GaussDB
8.102

Vector Database Developer Guide for Primary+Standby Instances

Issue 01

Date 2024-04-30

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Product Description

Background

In the big data era, data volume increases explosively and data types become complex. Especially with the emergence of machine learning and artificial intelligence, requirements for effectively processing high-dimensional and unstructured data become increasingly urgent. The unstructured data (such as images, audios, videos, and complex texts), usually represented by high-dimensional vectors, poses a challenge to processing and retrieval of unstructured data in traditional relational databases. Against this backdrop, vector databases emerge to meet these new requirements by providing more efficient data processing and retrieval solutions.

Concept of Vector Database

A vector database is a database system designed for storing, managing, and retrieving high-dimensional vector data. Compared with traditional relational databases, they differ significantly in data model, indexing mechanism, and query processing. Vector databases use optimized indexing technologies, such as approximate nearest neighbor (ANN) search, to improve the retrieval efficiency of unstructured data. In addition, they can support complex data analysis and machine learning models, making data processing more efficient and intelligent.

Application Scenario

With the rapid development of big data and machine learning models, vector databases show their unique value in multiple fields. With the efficient retrieval capability and flexible data processing mechanism, vector databases are ideal to be applied in recommendation systems, image recognition, natural language processing, and complex data analysis. In these scenarios, the vector databases not only speed up data processing, but also improve the accuracy and depth of analysis, providing powerful support for data-driven decision-making.

In summary, vector databases are mainly used in the following scenarios:

- **Recommendation systems** in scenarios such as video recommendation, news recommendation, and shopping recommendation
- **Image recognition** in scenarios such as product search by image and facial recognition

- **Unstructured data processing** in scenarios such as audio, video, and image data conversion (to vector data) for criticality analysis
- **LLM-based intelligent question answering systems**, for example, vector databases are used to improve semantic search engines, and provide more accurate search results by storing and retrieving vector representations of text contents.

Working Principle

The vector database integrates the optimal disk-based vector retrieval algorithm into the GaussDB, enabling database users to use native SQL statements to create and import vector data, build indexes, generate query plans, and efficiently retrieve vectors. In the current version, you can use vector types FloatVector and BoolVector as native types. You can create GsIVFFLAT or GsDiskANN indexes to accelerate top K ANN queries. The GsIVFFLAT index divides the high-dimensional vector space into different buckets based on distances by using a clustering algorithm. Then, in a vector retrieval process, a candidate vector retrieval set may be selected based on a distance between a query vector and a vector bucket center point. Therefore, the retrieval cost is significantly reduced compared with that of a full scan. The GsDiskANN index searches for the nearest neighbor vector for all vector points, constructs a sparse graph structure, and uses the relaxation coefficient to generate some "short-circuit" edges to accelerate query. GaussDB optimizes the data deletion logic to implement real-time deletion and ensure performance with no accuracy loss during long-term running. GaussDB vector indexing supports high-concurrency retrieval and modification and two underlying storage engines: Astore and Ustore. In addition, it also supports MVCC. Log types are introduced to the GaussDB vector database to support complete HA capabilities, such as primary and standby synchronization in a centralized system, parallel replay, and ultimate RTO.

Figure 1-1 Working principle of IVFFLAT

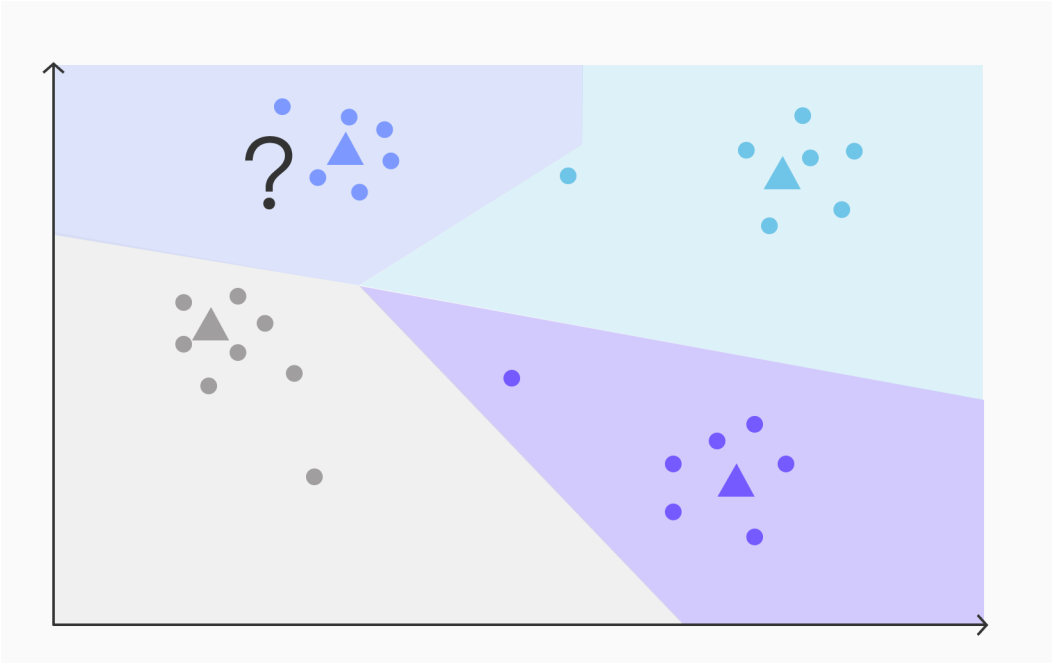
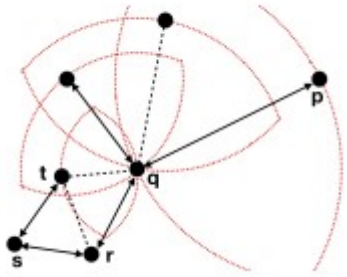
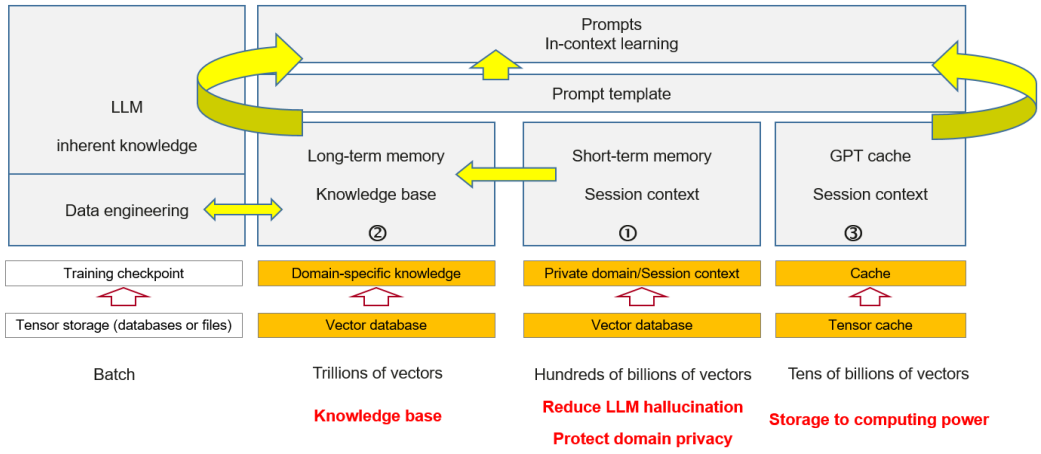


Figure 1-2 Working principle of DiskANN



In the LLM scenario, the vector database, like the "hippocampus" in the LLM technology stack, serves as the long-term and short-term memory. Specifically, there are three capabilities: long-term memory, short-term memory, and temporary cache.



- Long-term memory is mainly used in knowledge base scenarios where trillions of data records are involved and there are high requirements on the retrieval recall rate. In such scenarios, the vector database can supplement prompts through similar retrieval to improve the answer accuracy and efficiency and solve the LLM hallucination problem.
- Short-term memory is mainly used in session context scenarios where hundreds of billions of data records are involved and there are requirements for efficient data insertion and retrieval. In such scenarios, the vector database can supplement prompts for new questions through similar retrieval, generate corresponding answers, and save them in the database.
- Temporary cache is mainly used to accelerate answer generation. In this scenario, a large amount of data is involved and there are requirements for efficient data insertion and retrieval. In such scenario, the vector database temporarily stores the embeddings (that is, character prefixes) that have occurred and the next characters to be generated. In this way, the answer generation process can be accelerated through retrieval.

Constraints

1. In the current version, only deployment in standalone or centralized mode is supported. Distributed and lite deployment are not supported.

2. When a table is created, the table allows a maximum of 1024 dimensions. That is, the dimension range is [1,1024].
3. No global partitioned index (GPI) can be created for a distributed table.
4. A vector index cannot be a unique index.
5. Vector indexes cannot be sorted using CLUSTER BY.
6. A vector index cannot be a composite index.
7. Partition keys cannot be of the floatvector or boolvector type.
8. Primary keys cannot be created as the floatvector or boolvector type.
9. Unique keys cannot be created as the floatvector or boolvector type.
10. NULL values cannot be of the floatvector or boolvector type.
11. Currently, database links cannot be of the vector type. Therefore, vector indexes cannot be created for database links.
12. Vector indexes cannot be read on the standby node.
13. Vector indexes cannot be incrementally built or created online.
14. Indexes (B-tree and UB-tree indexes) other than vector indexes cannot be created as the floatvector or boolvector type.
15. Currently, vector indexes support top-K (sorted by distance in ascending order) query statements only. Other distance operations are not indexed.
16. It takes a long time to build indexes. Therefore, spare sufficient time for build in scenarios with low memory.
17. Graph indexes feature a high bloat rate and may fail to be built if the memory is low.
18. Vector retrieval is a resource-intensive operation. If multiple services are performed at the same time, the performance is affected.
19. Vector indexes cannot be created for unlogged or temp tables.
20. The Ustore indexes occupy space up to 10% more than that for Astore indexes and the retrieval delay increases by 5%.
21. Indexes are recycled depending on VACUUM. If frequent addition or deletion is performed between two VACUUM operations, vector retrieval efficiency decreases and space bloat occurs.
22. No statistics on index types are collected. For selectivity estimation, the default value is used; and cost estimation is supported. However, only index scan and sequential scan are supported currently.

2 Using a Vector Database

2.1 Getting Started

In machine learning and natural language processing, embedding refers to a process of mapping high-dimensional data (such as texts, images, and audios) to low-dimensional space. An embedding vector is usually a vector consisting of real numbers. It represents the input data as points in a continuous numerical space. Simply put, an embedding is an N-dimensional real-value vector that can be used to represent anything, such as text, music, and video.

This section describes the basic functions of the vector database, including creating vector tables, modifying vector attribute types, managing vector data, and clearing tables.

Step 1 Create a table that contains vector types and set data dimensions. When creating a table, you must specify dimensions for each vector type. If the dimensions of the inserted data are different from the specified dimensions, the database reports an error.

```
gaussdb=# CREATE TABLE t1(id int unique, repr floatvector(4));
gaussdb=# CREATE TABLE t2(id int unique, repr boolvector(3));
```

Step 2 Add a vector column. The default value must be specified. Otherwise, the database reports an error.

```
gaussdb=# ALTER TABLE t1 ADD COLUMN floatvec floatvector(3) default '[1,1,1]';
```

Step 3 Modify the vector data type attribute in the table. You can run the **ALTER TABLE** statement to modify the vector dimensions.

```
gaussdb=# ALTER TABLE t1 ALTER COLUMN repr type floatvector(4);
```

Note: If data has been inserted into the data table, the data in the table must comply with the dimension settings when the vector dimensions are modified. Otherwise, the database reports an error. If a vector index has been created for a vector attribute, the data type of the attribute cannot be changed and the database reports an error.

Step 4 Insert data.

```
gaussdb=# INSERT INTO t1 VALUES(0, '[30,12,12,25]');
gaussdb=# INSERT INTO t1 VALUES(1, '{7.5,62,1,235}');
```


The character data is implicitly converted to the vector type.

```
gaussdb=# copy t1 from 'data/floatvector_test.csv' csv header;
gaussdb=# INSERT INTO t2 VALUES(1, '[1, 0, 1]');
gaussdb=# INSERT INTO t2 VALUES(2, '[T, T, F]');
gaussdb=# INSERT INTO t2 VALUES(3, '[false, true, true]');
gaussdb=# INSERT INTO t2 VALUES(4, '[Y, N, N]');
gaussdb=# INSERT INTO t2 VALUES(5, '[no, yes, no]');
gaussdb=# INSERT INTO t2 VALUES(6, '[off, off, on]');
gaussdb=# INSERT INTO t2 VALUES(7, '[1, yes, on]');
```

Step 5 Delete data.

```
gaussdb=# DELETE t1 where repr <-> '[30,12,12,25]' < 0.2;
```

Step 6 Perform UPDATE and DELETE operations on vector and non-vector columns, as well as VACUUM operations.

```
gaussdb=# UPDATE t1 SET id = 38;
gaussdb=# UPDATE t1 SET repr = '[1000,152,132,5]';
gaussdb=# DELETE FROM t1 WHERE id=38;
gaussdb=# DELETE FROM t1 WHERE repr='[30,12,12,25]';
gaussdb=# DELETE FROM t1 WHERE repr <-> '[30,12,12,25]' < 0.8;
gaussdb=# VACUUM t1;
```

----End

2.2 Data Type Conversion

The vector database supports conversion between vector data types. In this process, data can be freely converted between vector data types and corresponding array types.

- array -> vector

  Explicit conversion using functions

  ```
  gaussdb=# SELECT floatvector(ARRAY[1,2,9.3]);
  ```

- vector -> array

  Explicit conversion using functions

  ```
  gaussdb=# SELECT vector_to_array(floatvector('[1,2,9.3]'));
  ```

- string -> vector

  Explicit conversion

  ```
  gaussdb=# CREATE TABLE t1 (id int, b varchar);
  gaussdb=# INSERT INTO t1 VALUES (1, '[2.3, 1.8, 3.6]');
  gaussdb=# SELECT '[1,2,3]' <-> b::floatvector from t1;
  gaussdb=# SELECT '[1,2,3]' <-> CAST(b AS floatvector) from t1;
  ```

  Implicit conversion

  ```
  gaussdb=# create table t2 (b floatvector(3));
  gaussdb=# INSERT INTO t2 VALUES ('[2.3, 1.8, 3.6]');
  gaussdb=# SELECT '[1,2,3]' <-> b from t2;
  ```


Table 2-1 Mapping between character types and vector types

Conversion Mode	Type Supporting Line Breaks
CAST(::)	• char(n) • character(n) • nchar(n) • varchar(n) • character varying(n) • varchar2(n) • nvarchar2 • clob • text • bpchar
Implicit conversion	unknown (If a type is not specified for a string literal, then the placeholder type **unknown** is assigned initially.) For details, see **Developer Guide** > **SQL Reference** > **Type Conversion**.)

2.3 Operators

The vector database supports binary operators and you can perform addition, subtraction, inner product, and comparison operations on vector data.

- Addition (+) and subtraction (–)
  ```
  gaussdb=# SELECT floatvector('[1,2,3]') - floatvector('[1,2,3]');
  ```
  Note: Only floatvector supports vector addition and subtraction.
- Inner product: No vector data type supports the multiplication (*) operator. Instead, you can use system functions for calculation.
  ```
  inner_product(floatvector, floatvector)
  gaussdb=# SELECT inner_product('[1,2,3]','[2,3,4]');
  ```
- Comparison (<, >, <=, >=, <>, and =)
  ```
  gaussdb=# SELECT floatvector('[1,2,3]') < floatvector('[1,2,3]');
  gaussdb=# SELECT boolvector('[T,F,T]') = boolvector('[T,T,T]');
  ```
  Note: The boolvector type applies only to the equal-to operation and does not support comparison operations.

2.4 Creating and Managing Indexes

2.4.1 GsIVFFLAT

GsIVFFLAT is used for vector retrieval in the case of a small amount (tens of thousands to millions) of data. You can create a single-column index on a column that contains vector data or delete an index. In addition, indexes can be automatically updated after data changes.

Prerequisites

The number of data records is not less than 1 x 10^4 or greater than 2 x 10^6. If the data volume is too small, you are advised not to create GsIVFFLAT indexes. If the data volume is too large, you are advised to use other vector indexes.

Usage Suggestions

- You are advised to set **nlist** to 4 x **sdrt***(N)* for a table with *N* data records.
- You are advised to set the GUC parameter **probes** to 10% of the index hyperparameter **nlist**. Increase or decrease the value of **probes** based on the required query accuracy and delay. The query time is linearly related to **probes**/**nlist**.
- You are advised to insert all data to be queried into the table in advance and then create indexes. If you decide to create indexes before importing data, you need to rebuild indexes after the data is imported.
- After an index is created, if the data change (including insertion, deletion, and update) exceeds 20%, data distribution drift occurs. As a result, the query accuracy decreases and the query time is unpredictable. You are advised to rebuild the index every time 10% to 20% data is updated.
- If the number of data records is less than the value of **nlist** during index creation, **nlist** is regarded as **1** and an error message is displayed. In this case, the query is no different from a full table scan, especially the accuracy and delay. To ensure that the index performance meets the expectation, you need to rebuild the index after the data import is complete. Note that, after the TRUNCATE operation is performed on the table, the same error message is displayed for the GsIVFFLAT index to make the index in the preceding state. To make the index available again, you need to rebuild the index after the data is imported again.

Parameters

Parameter	Value Range	Description
ivf_nlist	**Value range**: 1–65535 **Default value**: **100**	Number of inverted indexes
ivf_nlist2	**Value range**: 0-65535 **Default value**: **0**	Number of level-2 inverted indexes

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE "test" ("id" BIGINT PRIMARY KEY, "repr" floatvector (128)) with
(storage_type=astore);

-- Import data.
gaussdb=# \copy test from 'path/to/data.csv' WITH(format 'text', delimiter E'\t', ignore_extra_data 'true',
noescaping 'true');

-- Create an index.
gaussdb=# CREATE INDEX testivfflat ON test USING GSIVFFLAT(repr L2) WITH (IVF_NLIST = 16,
```



```
IVF_NLIST2  = 256 );

-- Retrieve vector data using an ANN algorithm.
gaussdb=# set gsivfflat_probes = 15;
gaussdb=# SELECT "id" FROM "test"  ORDER BY "repr" <-> '[
23.0,0.0,0.0,0.0,1.0,5.0,43.0,114.0,3.0,0.0,0.0,0.0,14.0,121.0,120.0,78.0,81.0,4.0,0.0,0.0,31.0,126.0,23.0,18.0,126.0
,12.0,0.0,0.0,0.0,1.0,0.0,10.0,0.0,0.0,0.0,0.0,8.0,29.0,96.0,43.0,0.0,0.0,0.0,0.0,1.0,81.0,126.0,44.0,126.0,1.0,0.0,0.0,
1.0,45.0,66.0,96.0,126.0,0.0,0.0,0.0,1.0,16.0,12.0,63.0,1.0,2.0,0.0,0.0,11.0,40.0,26.0,0.0,5.0,20.0,28.0,1.0,0.0,17.0,
36.0,5.0,126.0,45.0,10.0,1.0,0.0,2.0,12.0,29.0,126.0,6.0,0.0,0.0,2.0,110.0,96.0,46.0,18.0,13.0,0.0,0.0,3.0,5.0,1.0,2.0
,29.0,50.0,30.0,7.0,8.0,3.0,0.0,1.0,55.0,24.0,14.0,5.0,9.0,15.0,8.0,10.0,10.0,1.0,0.0,0.0,19.0,79.0,16.0,4.0
]'  LIMIT 10;

-- Insert data.
gaussdb=# INSERT INTO test(id,repr) VALUES(99999999,'[
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0
]');
gaussdb=# SELECT COUNT(*) FROM test;

-- Update data.
gaussdb=# UPDATE test SET repr = '[
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0
]' WHERE id = 99999999;
gaussdb=# SELECT * FROM test WHERE id = 99999999;

-- Delete data.
gaussdb=# DELETE FROM test WHERE id = 99999999;
gaussdb=# SELECT COUNT(*) FROM test;

-- Clear data.
gaussdb=# VACUUM test;

-- Drop an index.
gaussdb=# DROP INDEX testivfflat;

Drop a table.
gaussdb=# DROP TABLE test;
```

2.4.2 GsDiskANN

GsDiskANN is used for vector retrieval in the case of a large amount (tens of millions or hundreds of millions) of data. You can create a single-column index on a column that contains vector data or delete an index. In addition, indexes can be automatically updated after data changes.

Prerequisites

- The database is running properly, and users can log in to the database.
- A table of the vector type has been created, and the table contains a small number of rows (less than 10,000 rows).
- An index of the GsDiskANN type has been created.

Usage Suggestions

- Create a GsDiskANN index on the vector attribute column where query acceleration is required.
- The **pq_nseg** parameter controls the number of segments created by the PQ module. The number of vector dimensions must be exactly divided by the number of segments. The larger the number of segments, the larger the

occupied space, but the more accurate the distance calculation result. Therefore, the recommended setting rules are as follows:

  - When the number of vector dimensions is less than 512, you are advised to set the number of segments to the same value as the dimensions.
  - When the number of vector dimensions is greater than 512 and less than or equal to 1024, you are advised to set it to half of the number of dimensions (if the number of dimensions can be exactly divided).

- The **pq_nclus** parameter controls the number of categories in each PQ segment. This parameter specifies the number of mapping clusters in each segment. The value of this parameter must be less than the number of data rows. A larger value of this parameter indicates a higher distance calculation accuracy. If the number of vector dimensions is less than 1024, the recommended value is **16**.
- The **num_parallels** parameter controls the concurrency during index creation. If the memory is sufficient (all indexes can be stored) and Intel Xeon 5 or later processors are used, the **num_parallels** parameter can be set to **50** when the degree of concurrency is less than **30**. The **num_parallels** parameter is restricted by the maximum number of bgworkers in the database. This maximum value is **64**. If the number of bgworkers that are allocated to the database process exceeds this value, an error is reported. Note that in the scenario where indexes are created for multiple sessions at the same time, do not create too many threads.
- When indexes are built from existing data tables and the data volume is sufficient, it is recommended **enable_pq** be set to **true** so that PQ can be used to calculate the distance between neighboring nodes, avoiding a large number of disk (or shared memory) accesses and accelerating the build.

CAUTION

If the table data is insufficient and you try to create an index and set **enable_pq** to **true**, an alarm is reported and the index is automatically created with **enable_pq** set to **false**. If a GsDiskANN index without PQ compression is used, the performance deteriorates. If you want to create an index with **enable_pq** set to **true**, ensure that the data volume is sufficient and try to reindex. During table truncation and table-level backup and restoration in the database, data may be insufficient for index creation. After the operation is complete, you are advised to rebuild the index to ensure that PQ compression is used to optimize the indexing efficiency.

- When indexes are built from existing data tables and the data volume is insufficient, the value of **enable_pq** must be set to **false**. In this scenario, PQ is not used to calculate the distance.
- When indexes are created from existing data tables, enabling the **using_clustering_for_parallel** parameter can alleviate concurrency conflicts. However, long tail may occur due to unbalanced clustering. Therefore, this parameter is disabled by default.
- The types of the new data must be the same as those of the created data table, and the vector dimensions must be the same.
- To delete data, you must specify the search criteria for the data to be deleted. Deletion one by one and batch deletion are supported.

- Due to the limited storage length, the output of each node in the GsDiskANN graph is limited. Therefore, the connectivity of the entire graph cannot be ensured. When a user queries outliers, the returned query results may be less than the specified top K results. To ensure the output quantity, use the GsIVFFLAT index instead.
- The size of the GsDiskANN index is estimated to be 10 to 50 times of the disk occupied by the original data. Determine the disk capacity before creating the index.

Parameters

Parameter	Value Range	Description
pq_nseg	**Value range**: 1–65535 **Default value**: **1**	Number of original vector segments.
pq_nclus	**Value range**: 2–256 **Default value: 16**	Number of central points of each vector segment.
queue_size	**Value range**: 64–1000 **Default value**: **100**	Queue size specified during disk build.
num_parallels	**Value range**: 1–64 **Default value**: **10**	Number of parallel computing tasks for building indexes.
using_clustering_for_parallel	**Value range**: **T** and **F** **Default value**: **F**	Specifies whether to use clusters for parallel build (**T**: yes; **F**: no).
lambda_for_balance	**Value range**: 0–*DBL_MAX* **Default value**: **0.00001**	Balance among parallel computing tasks.
enable_pq	**Value range**: **T** and **F** **Default value**: **T**	Specifies whether to use PQ for acceleration during build and search (**T**: yes; **F**: no).

CAUTION

Each time GsDiskANN updates a vector, data on multiple index pages needs to be updated. Therefore, a large number of logs are recorded, and "LOG: checkpoints are occurring too frequently" logs may be frequently printed, which is normal. If you do not want to print them, increase the value of **checkpoint_segments** when resources are sufficient. For details, see **checkpoint_segments** in **Administrator Guide** > **Configuring Running Parameters** > **GUC Parameters** > **WALs** > **Checkpoints**.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE "test" ("id" BIGINT PRIMARY KEY, "repr" floatvector (128)) with
(storage_type=astore);

-- Import data.
gaussdb=# \copy test from 'path/to/data.csv' WITH(format 'text', delimiter E'\t', ignore_extra_data 'true',
noescaping 'true');

-- Create an index.
gaussdb=# CREATE INDEX diskann_sift1m_l2 on test USING GsDiskANN (repr L2) with (pq_nseg=128,
pq_nclus=16, num_parallels=50, enable_pq=true, using_clustering_for_parallel=false);

-- Retrieve vector data using an ANN algorithm.
gaussdb=# SELECT "id" FROM "test"  ORDER BY "repr" <-> '[
23.0,0.0,0.0,0.0,1.0,5.0,43.0,114.0,3.0,0.0,0.0,0.0,14.0,121.0,120.0,78.0,81.0,4.0,0.0,0.0,31.0,126.0,23.0,18.0,126.0
,12.0,0.0,0.0,0.0,1.0,0.0,10.0,0.0,0.0,0.0,0.0,8.0,29.0,96.0,43.0,0.0,0.0,0.0,0.0,1.0,81.0,126.0,44.0,126.0,1.0,0.0,0.0,
1.0,45.0,66.0,96.0,126.0,0.0,0.0,0.0,1.0,16.0,12.0,63.0,1.0,2.0,0.0,0.0,11.0,40.0,26.0,0.0,5.0,20.0,28.0,1.0,0.0,17.0,
36.0,5.0,126.0,45.0,10.0,1.0,0.0,2.0,12.0,29.0,126.0,6.0,0.0,0.0,2.0,110.0,96.0,46.0,18.0,13.0,0.0,0.0,3.0,5.0,1.0,2.0
,29.0,50.0,30.0,7.0,8.0,3.0,0.0,1.0,55.0,24.0,14.0,5.0,9.0,15.0,8.0,10.0,10.0,1.0,0.0,0.0,19.0,79.0,16.0,4.0
]'  LIMIT 10;

-- Insert data.
gaussdb=# INSERT INTO test(id,repr) VALUES(99999999,'[
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0
]');
gaussdb=# SELECT COUNT(*) FROM test;

-- Update data.
gaussdb=# UPDATE test SET repr = '[
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0
]' WHERE id = 99999999;
gaussdb=# SELECT * FROM test WHERE id = 99999999;

-- Delete data.
gaussdb=# DELETE FROM test WHERE id = 99999999;
gaussdb=# SELECT COUNT(*) FROM test;

-- Clear data.
gaussdb=# VACUUM test;

-- Drop an index.
gaussdb=# DROP INDEX diskann_sift1m_l2 ;

Drop a table.
gaussdb=# DROP TABLE test;
```

2.5 Similarity Search

Distance is usually used to describe the similarity between vectors. Currently, the vector database provides three distance calculation methods: Euclidean distance, Cosine distance, and Hamming distance. Euclidean distance is affected by the angle difference and length difference between vectors. Cosine distance focuses only on the angle difference between vectors. Hamming distance is used only to calculate the boolvector and the number of different elements in the vector.

Example of Similarity Search Using Euclidean Distance

When calculating the Euclidean distance between vectors, the operator <-> and the system function l2_distance apply only to floatvector vectors.

- l2_distance operator (<->)
```
gaussdb=# SELECT id, repr<-> '[1,1,3,2]' AS s FROM t1;
gaussdb=# SELECT floatvector('[1,2,3]')<->floatvector('[5,-1,3.5]');
```
- System function (l2_distance)
```
gaussdb=# SELECT id, l2_distance(repr, '[1,1,3,2]') AS s FROM t1;
gaussdb=# SELECT l2_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
```
  Undefined character types (**unknown**) can be implicitly converted.
```
gaussdb=# SELECT l2_distance('[1,2,3]',  '[5,-1,3.5]');
```
- Similarity search is performed for data fields with the same dimensions. The output is as follows:

  Vector indexes can be sorted only in ascending order. Currently, feature vector indexes cannot be sorted in descending order.
```
gaussdb=# SELECT id FROM t1 ORDER BY repr<-> '[1,1,3,2]' LIMIT 2;
```
  Multi-table join is supported.
```
gaussdb=# SELECT t1.repr FROM t1 inner join t2 ON t1.repr <-> t2.repr < 0.8;
```
  Subqueries are supported.
```
gaussdb=# SELECT t1.id FROM t1 WHERE t1.repr <-> (SELECT t2.repr FROM t2 LIMIT 1) < 0.8;
```

Example of Similarity Search Using Cosine Distance

When calculating the Cosine distance between vectors, the operator <+> and the system function cosine_distance apply only to floatvector vectors.

- Operator (<+>)
```
gaussdb=# select id, repr<+> '[1,1,3,2]' as s from t1;
gaussdb=# select floatvector('[1,2,3]')<+>floatvector('[5,-1,3.5]');
```
- System function (cosine_distance)
```
gaussdb=# select id, cosine_distance(repr, '[1,1,3,2]') as s from t1;
gaussdb=# select cosine_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
```
  Undefined character types (**unknown**) can be implicitly converted.
```
gaussdb=# select cosine_distance('[1,2,3]', '[5,-1,3.5]');
```
- Similarity search is performed for data fields with the same dimensions. The output is as follows:

  Vector indexes can be sorted only in ascending order. Currently, feature vector indexes cannot be sorted in descending order.
```
gaussdb=# select id from t1 order by repr<+> '[1,1,3,2]' limit 2;
```
  Multi-table join is supported.
```
gaussdb=# select t1.repr from t1 inner join t3 on t1.repr <+> t3.repr < 0.8;
```
  Subqueries are supported.
```
gaussdb=# select t1.id from t1 where t1.repr <+> (select t2.repr from t2 limit 1) < 0.8;
```

Example of Similarity Search Using Hamming Distance

When calculating the Hamming distance between vectors, the operator <#> and the system function hamming_bool_distance apply only to boolvector vectors.

- Operator (<#>)
```
gaussdb=# select id, repr<#> '[1,1,0]' as s from t2;
gaussdb=# select boolvector('[T,F,F]')<#>boolvector('[T,F,T]');
```
- System function (hamming_bool_distance)
```
gaussdb=# select id, hamming_bool_distance(repr, '[1,1,0]') as s from t2;
gaussdb=# select hamming_bool_distance(boolvector('[T,F,F]'), boolvector('[T,F,T]'));
```
  Undefined character types (**unknown**) can be implicitly converted.


```
gaussdb=# select hamming_bool_distance('[T,F,F]', '[T,F,T]');
```

- Similarity search is performed for data fields with the same dimensions. The output is as follows:

  Vector indexes can be sorted only in ascending order. Currently, feature vector indexes cannot be sorted in descending order.

  ```
  gaussdb=# select id from t2 order by repr<#> '[1,1,0]' limit 2;
  ```

  Multi-table join is supported.

  ```
  gaussdb=# select t2.repr from t2 inner join t4 on t2.repr <#> t4.repr < 3;
  ```

  Subqueries are supported.

  ```
  gaussdb=# select t2.id from t2 where t2.repr <#> (select t4.repr from t4 limit 1) < 3;
  ```


3 Helpful Links

3.1 Vector Data Types

Vector data types include floatvector and boolvector.

- The floatvector data type indicates that multidimensional data contains data of the float type, for example, [1.0,3.0,11.0,110.0,62.0,22.0,4.0].

  NOTE

  - The floatvector member supports only single precision. Single-precision range: -3.402E+38 to +3.402E+38, 6-digit decimal digits.
  - The floatvector does not support NULL, Nan, or Inf as elements. If a vector contains NULL values, the database reports an error.
  - The floatvector cannot be NULL. The database reports an error when a NULL value is inserted, updated, or converted as vector data.

- The boolvector data type indicates that the multidimensional data contains data of the Boolean type, for example, [0,0,0,0,0,1,0,0,1,0,0,0].

  NOTE

  - Members of the boolvector data type can express Boolean data in **t(T)**/**f(F)**, **y(Y)**/**n(N)**, **1**/**0**, **true**/**false**, **yes**/**no**, or **on**/**off** mode.
  - The boolvector does not support NULL, Nan, or Inf as elements. If a vector contains NULL values, the database reports an error.
  - The boolvector cannot be NULL. The database reports an error when a NULL value is inserted, updated, or converted as vector data.

Usage of Vector Types

An example of using the vector type is as follows:

```
-- Create a table that contains vector types and set data dimensions. Dimensions must be specified for vector types during table creation.
gaussdb=# CREATE TABLE t1(id int unique, repr floatvector(4));
gaussdb=# CREATE TABLE t2(id int unique, repr boolvector(3));
-- Insert data.
gaussdb=# INSERT INTO t1 VALUES(0, '[30,12,12,25]');
gaussdb=# INSERT INTO t2 VALUES(1, '[1, 0, 1]');
```


3.2 Vector Calculation Interfaces and Functions

3.2.1 Vector Distance Calculation Interface

l2_distance

Function: Calculates the Euclidean distance between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT l2_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT l2_distance('[1,2,3]',  '[5,-1,3.5]');
```

vector_l2_squared_distance

Function: Calculates the square of the Euclidean distance between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT vector_l2_squared_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_l2_squared_distance('[1,2,3]',  '[5,-1,3.5]');
```

cosine_distance

Function: Calculates the cosine distance between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# select cosine_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# select cosine_distance('[1,2,3]', '[5,-1,3.5]');
```

vector_spherical_distance

Function: Calculates the spherical distance between two normalized vectors (represented by the radian system of the cosine angle).

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# gaussdb=# select vector_spherical_distance('[1,0,0,0]', '[0,0,0,1]');
```

NOTE

If the input vector is not normalized, the correct calculation result cannot be obtained.

3.2.2 Vector Operation Function Interface

Functions implemented by the vector operation function include: vector size comparison, vector addition, vector subtraction, and vector bitwise multiplication.

inner_product

Function: Calculates the inner product of two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT inner_product(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT inner_product('[1,2,3]', '[5,-1,3.5]');
```

vector_negative_inner_product

Function: Calculates the negative inner product of two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT vector_negative_inner_product(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_negative_inner_product('[1,2,3]', '[5,-1,3.5]');
```

vector_dims

Function: Returns the dimension value of the floatvector vector.

Input parameter type: floatvector

Output parameter type: int4

Code example:

```
gaussdb=# SELECT vector_dims(floatvector('[1,2,3]'));
gaussdb=# SELECT vector_dims('[1,2,3]');
```

vector_norm

Function: Returns the L2 norm of a vector.

Input parameter type: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT vector_norm(floatvector('[1,2,3]'));
gaussdb=# SELECT vector_norm('[1,2,3]');
```

vector_add

Function: Calculates the sum of two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT vector_add(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_add('[1,2,3]', '[5,-1,3.5]');
```

vector_sub

Function: Calculates the subtraction between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT vector_sub(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_sub('[1,2,3]', '[5,-1,3.5]');
```

vector_lt

Function: Compares the vector size and checks whether vector 1 is less than vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_lt(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_lt('[1,2,3]', '[5,-1,3.5]');
```

vector_le

Function: Compares the vector size and checks whether vector 1 is less than or equal to vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_le(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_le('[1,2,3]', '[5,-1,3.5]');
```

vector_eq

Function: Compares whether vectors are equal.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_eq(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_eq('[1,2,3]', '[5,-1,3.5]');
```

vector_ne

Function: Compares whether vectors are unequal.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_ne(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_ne('[1,2,3]', '[5,-1,3.5]');
```

vector_ge

Function: Compares the vector size and checks whether vector 1 is greater than or equal to vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_ge(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_ge('[1,2,3]', '[5,-1,3.5]');
```

vector_gt

Function: Compares the vector size and checks whether vector 1 is greater than vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_gt(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_gt('[1,2,3]', '[5,-1,3.5]');
```

vector_cmp

Function: Compares vector sizes.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: int4

Code example:

```
gaussdb=# SELECT vector_cmp(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_cmp('[1,2,3]', '[5,-1,3.5]');
```

vector_accum

Function: Returns the vector accumulation.

Type of input parameter 1: anyarray

Type of input parameter 2: floatvector

Output parameter type: anyarray

Code example:

```
-- This is a system function and is not recommended. If this function is required, the array element type must be float8.
gaussdb=# SELECT vector_accum(array[cast(3 as float8),1,2,3],  floatvector('[5,-1,3.5]'));
```

vector_combine

Function: Combines vectors.

Type of input parameter 1: anyarray

Type of input parameter 2: anyarray

Output parameter type: anyarray

Code example:

```
-- This is a system function and is not recommended. If this function is required, the array element type must be float8.
gaussdb=# SELECT vector_combine(array[cast(1 as float8),2,3], array[cast(1 as float8),2,3]);
```

vector_avg

Function: Averages vectors.

Input parameter type: anyarray

Output parameter type: floatvector

Code example:


```
-- This is a system function and is not recommended. If this function is required, the array element type must be float8.
gaussdb=# SELECT vector_avg(array[cast(1 as float8),2,3]);
```

bool_vector_dims

Function: Returns the dimension value of the boolvector vector.

Input parameter type: boolvector

Output parameter type: int4

Code example:

```
gaussdb=# SELECT bool_vector_dims(boolvector('[1,1,1]'));
```

bool_vector_eq

Function: Compares whether Boolean vectors are equal.

Type of input parameter 1: boolvector

Type of input parameter 2: boolvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT bool_vector_eq(boolvector('[1,1,1]'), boolvector('[1,1,1]'));
gaussdb=# SELECT bool_vector_eq('[1,1,1]', '[1,1,1]');
```

3.2.3 Vector Functions and Operators

floatvector supports data conversion between the vector type and array type. In addition, strings in specific formats can be converted to the vector type.

array<->floatvector: During data type conversion, the vector data type can be freely converted to the corresponding array type. The member data type of the floatvector vector is floating point.

string ->floatvector: You need to pay attention to the character string format. You need to contain arrays using square brackets ([]) or braces ({}) and separate elements using commas (,).

floatvector

Function: Converts array data into vector data.

Scenario 1:

Input parameter type: anyarray

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT floatvector(ARRAY[1,2,9.3]);
```

Scenario 2:

Type of input parameter 1: floatvector

Type of input parameter 2: integer

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT floatvector(floatvector('[1,2,9.3]'),3);
```

vector_to_array

Function: Converts vector data into array data.

Input parameter type: floatvector

Output parameter type: real[]

Code example:

```
gaussdb=# SELECT vector_to_array(floatvector('[1,2,3]'));
```

text_to_vector

Function: Converts character data into vector data.

Input parameter type: cstring

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT text_to_vector('[1,2,9.3]');
```

vector_in

Function: Specifies the floatvector input conversion function (text format).

Type of input parameter 1: cstring

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT vector_in('[1,2,3]',701,3);
```

vector_out

Function: Specifies the floatvector output conversion function (text format).

Input parameter type: floatvector

Output parameter type: cstring

Code example:

```
gaussdb=# SELECT vector_out(floatvector('[1,2,3]'));
```


vector_send

Function: Specifies the floatvector input conversion function (binary format).

Input parameter type: floatvector

Output parameter type: bytea

Code example:

```
gaussdb=# SELECT vector_send(floatvector('[1,2,3]'));
```

vector_recv

Function: Specifies the floatvector output conversion function (binary format).

Type of input parameter 1: integer

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: floatvector

Code example:

```
-- This is a system function and cannot be invoked by SQL statements.
```

vector_typmod_in

Function: Specifies the floatvector input type modifier function.

Input parameter type: cstring[]

Output parameter type: integer

Code example:

```
gaussdb=# SELECT vector_typmod_in( '{1}' );
```

NOTE

- Vector data type members support only single precision.
- Only the same dimension is supported for vector calculation. If the dimensions are different, an error is reported.
- floatvector supports vector addition and subtraction. The dot product operation is performed by the inner_product function.
- When creating a table, you must specify dimensions for each vector type. If the dimensions of the inserted data are different from the specified dimensions, the database reports an error.
- If data has been inserted into the data table, the data in the table must comply with the dimension settings when the vector dimensions are changed. Otherwise, the database reports an error. If a vector index has been created for a vector attribute, the data type of the attribute cannot be switched to another data type, and the database reports an error.

boolvector

Scenario 1:

Function: Converts array data into vector data.

Input parameter type: anyarray

Output parameter type: boolvector

Code example:

```
gaussdb=# SELECT boolvector(ARRAY[1,0,1]);
```

Scenario 2:

Function: Converts to boolvector and detects dimensions.

Type of input parameter 1: boolvector

Type of input parameter 2: integer

Output parameter type: boolvector

Code example:

```
gaussdb=# SELECT boolvector(boolvector('[1,0,1]'),3);
```

boolvector_to_array

Function: Converts vector data into array data.

Input parameter type: boolvector

Output parameter type: anyarray

Code example:

```
gaussdb=# SELECT boolvector_to_array(boolvector('[1,0,1]'));
```

text_to_boolvector

Function: Converts character data into vector data.

Input parameter type: cstring

Output parameter type: boolvector

Code example:

```
gaussdb=# SELECT text_to_boolvector('[1,1,1]');
```

bool_vector_in

Function: Specifies the boolvector input conversion function (text format).

Type of input parameter 1: cstring

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: boolvector

Code example:

```
gaussdb=# SELECT bool_vector_in('[1,1,1]',701,3);
```


bool_vector_out

Function: Specifies the boolvector output conversion function (text format).

Input parameter type: boolvector

Output parameter type: cstring

Code example:

```
gaussdb=# SELECT bool_vector_out(boolvector('[1,1,1]'));
```

bool_vector_send

Function: Specifies the boolvector input conversion function (binary format).

Input parameter type: boolvector

Output parameter type: bytea

Code example:

```
gaussdb=# SELECT bool_vector_send(boolvector('[1,1,1]'));
```

bool_vector_recv

Function: Specifies the boolvector output conversion function (binary format).

Type of input parameter 1: internal

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: boolvector

Code example:

```
-- This is a system function and cannot be invoked by SQL statements.
```

bool_vector_typmod_in

Function: Specifies the boolvector input type modifier function.

Input parameter type: cstring[]

Output parameter type: integer

Code example:

```
gaussdb=# SELECT bool_vector_typmod_in( '{1}' );
```

NOTE

- The vector data type does not support null, nan, or inf as elements. If a vector contains **NULL** values, the database reports an error.
- When a vector data type is inserted, **NULL** cannot be used as the inserted value. When a **NULL** value is inserted as the vector data, the database reports an error.
- Elements of the boolvector type can express Boolean data in **t(T)**/**f(F)**, **y(Y)**/**n(N)**, **1**/**0**, **true**/**false**, **yes**/**no**, or **on**/**off** mode.
- The boolvector type applies only to the equal-to operation and does not support comparison operations.

gs_vector_index_options

Function: Displays the hyperparameter values of related vector indexes.

Input parameter type: text

Output parameter type: text

Code example:

```
-- Create a table.
gaussdb=# CREATE TABLE t1 (id int unique,repr floatvector(960)) with (storage_type=astore);
Insert data.
gaussdb=# copy t1 from '/data/gist1w.txt' delimiter '^';
Create an index.
gaussdb=# CREATE INDEX test1v on t1 using gsdiskann (repr l2) with
(pq_nseg=120,pq_nclus=64,queue_size=120,num_parallels=30,enable_pq=true,using_clustering_for_parallel=f
alse);

gaussdb=# SELECT gs_vector_index_options('test1v');
```

Operators for vector calculation focus on measuring similarity between vectors and binary operators of vectors.

<->

Function: Calculates the Euclidean distance (L2) between two vectors (floatvector).

Left parameter type: floatvector

Right parameter type: floatvector

Return type: double precision

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <->  floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<-> floatvector('[1,1,3,2]');
```

<+>

Function: Calculates the cosine distance between two vectors (floatvector).

Left parameter type: floatvector

Right parameter type: floatvector

Return type: double precision

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <+>  floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<+> floatvector('[1,1,3,2]');
```

<#>

Function: Calculates the Hamming distance between two vectors (floatvector).

Left parameter type: boolvector

Right parameter type: boolvector

Return type: double precision

Code example:

```
gaussdb=# SELECT boolvector('[1,0,1,0]') <#>  boolvector('[1,1,1,0]');
gaussdb=# SELECT '[1,0,1,0]'<#> boolvector('[1,1,1,0]');
```

+

Function: Calculates the bitwise addition of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: floatvector

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') + floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'+ floatvector('[1,1,3,2]');
```

-

Function: Calculates the bitwise subtraction of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: floatvector

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') + floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'+ floatvector('[1,1,3,2]');
```

<

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') < floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'< floatvector('[1,1,3,2]');
```

<=

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <= floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<= floatvector('[1,1,3,2]');
```

>

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') > floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'> floatvector('[1,1,3,2]');
```

>=

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') >= floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'>= floatvector('[1,1,3,2]');
```

=

Scenario 1:

Function: Determines whether two vectors with the same dimension are equal.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') = floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'= floatvector('[1,1,3,2]');
```

Scenario 2:

Function: Determines whether two Boolean vectors with the same dimension are consistent.

Left parameter type: boolvector

Right parameter type: boolvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT boolvector('[1,0,1,0]') =  boolvector('[1,1,1,0]');
gaussdb=# SELECT '[1,0,1,0]' =  boolvector('[1,1,1,0]');
```

<>

Function: Determines whether two vectors with the same dimension are unequal.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <> floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<> floatvector('[1,1,3,2]');
```

NOTE

If the vector element value is large, the intermediate calculation result may overflow the single-precision range during distance calculation. As a result, the distance result is NAN or INF.

3.3 Vector Indexes

Vector indexes can be created, maintained, or retrieved. For details about how to create an index, see **Developer Guide** > **SQL Reference** > **SQL Syntax** > **C** > **CREATE INDEX**. Vector indexes do not support UNIQUE or CONCURRENTLY.

- For details about how to rebuild an index, see **Developer Guide** > **REINDEX**.
- For details about how to delete an index, see **Developer Guide** > **DROP INDEX**.
- **GLOBAL**

  Global indexes cannot be created on a partitioned table.
- **DESC**

  Not supported.
- **CONCURRENTLY**

  Not supported.
- **UNIQUE**

  Not supported.
- **COLLATE collation**

  Not supported.
- **DEDUPLICATION(only yun)**

  Composite indexes are not supported.
- **USING method**

  Specifies the method of creating an index.

Value range: Add an index of any of the following types:

gsivfflat: GsIVFFLAT index, which is an inverted index for vector data.

gsdiskann: GsDiskANN index, which is a graph index for vector data.

- **WITH ({storage_parameter = value} [, ...])**

  Specifies the storage parameter used for an index.

  Value range: as shown in **Table 3-1** and **Table 3-2**.

GsIVFFLAT

Table 3-1 GsIVFFLAT

Vector Index Type	Function Name	Description
GsIVFFLAT	ivfflatbeginscan	Reads the index root node information and stores it in the memory. For IVFFLAT indexes, it reads the metadata information and information about the center point of each cluster.
	ivfflatbuild	Creates an inverted index by data cluster.
	ivfflatbuildempty	Reports an empty index.
	ivfflatbulkdelete	Deletes dead tuples in all indexes, and updates the insertion position of each cluster to the first free page.
	ivfflatcostestimate	Estimates the cost of IVFFLAT index scan.
	ivfflatendscan	Clears the variables in the index scan structure and releases them.
	ivfflatgettuple	Provides an executor to obtain an index data record and implements the core IVFFLAT index retrieval step.
	ivfflatinsert	Inserts new tuples into existing indexes and maintains IVF indexes.
	ivfflatoptions	Parses and validates the reloptions array for the index.
	ivfflatrescan	Resets the variables in the index scan structure.
	ivfflatvacuumclean-up	Updates IVFFLAT index statistics.

GsDiskANN

Table 3-2 GsDiskANN

Vector Index Type	Function Name	Description
GsDiskANN	diskannbuild	Creates a vector index based on the Vamana graph.
	diskannbuildempty	Reports an empty index.
	diskannoptions	Parses and validates the reloptions array for the index.
	diskanninsert	Inserts new tuples into existing indexes and maintains DiskANN indexes.
	diskannbeginscan	Reads the index root node information and stores it in the memory, loads the center point information of the PQ table, specifies the distance function, and creates key structures.
	diskanngettuple	Provides an executor to obtain an index data record and implements the core GsDiskANN index retrieval step.
	diskannrescan	Resets the variables in the index scan structure.
	diskannendscan	Clears the variables in the index scan structure and releases them.
	diskannbulkdelete	Resets dead tuples in all indexes and places them in FSM.
	diskannvacuumcleanup	Updates GsDiskANN index statistics.
	diskanncostestimate	Estimates the cost of GsDiskANN index scan.

NOTE

- If the memory required for creating an index exceeds the limit, an error is reported. In this case, you need to set the GUC parameter **maintenance_work_mem** to a proper value.
- The GsIVFFLAT index supports the floatvector and boolvector vector types, while the GsDiskANN index supports only the floatvector vector type.

4 FAQs

The error message "dimensions for type vector cannot exceed 1024" is displayed during table creation.

Answer: When creating a table, you need to specify the vector dimensions and ensure that the number of dimensions does not exceed 1024.

```
gaussdb=#  CREATE TABLE t2 (a int, repr floatvector(100));
```

The error message "The "boolvector" type declaration for column must have length set" is displayed during table creation.

Answer: When creating a table, you need to specify the vector dimensions.

```
gaussdb=#  CREATE TABLE t1 (a int, repr floatvector(2));
gaussdb=#  CREATE TABLE t2 (a int, repr boolvector(2));
```

If a table is defined as the vector type and the inserted data is not of this type, an error is reported.

Answer: The data to be inserted must comply with the data type requirement.

```
gaussdb=#  INSERT INTO t1 VALUES(0, '[10.1, 0.5]');
```

When data is inserted, if the string format of the inserted vector is incorrect, an error is reported.

Answer: The string of the vector data to be inserted must comply with the format requirements.

```
gaussdb=#  INSERT INTO t1 VALUES(1, '[1,2]');
gaussdb=#  INSERT INTO t1 VALUES(2, '{1,2}');
```

If the vector dimensions of the data to be inserted are inconsistent with those defined in a table, an error is reported.

Answer: The dimensions of the data to be inserted must be the same as the preset dimensions.

```
gaussdb=#  INSERT INTO t1 VALUES(1, '[1,0]');
gaussdb=#  INSERT INTO t2 VALUES(0, '[T,F]');
```


An error is reported when similarity search is performed on data with different dimensions.

Answer: The vector dimensions for calculating the similarity must be the same.

```
gaussdb=#  SELECT '[2,3,1]'<-> '[1,2,3]'::floatvector AS s;
```

If an incorrect index parameter value is specified during index creation, an error is reported.

Answer: Specify parameter values within the range when creating an index.

```
gaussdb=#  CREATE INDEX diskann_l2_idx on sift1m using gsdiskann (repr L2) WITH (pq_nseg=128,
pq_nclus=16, queue_size=100, num_parallels=30, enable_pq=true, using_clustering_for_parallel=false);
```

During index creation, if the index type does not match the attribute type, an error is reported.

Answer: The index type must match the attribute type. For details, see the vector index section.

```
gaussdb=#  CREATE INDEX h1 ON t1 USING GSIVFFLAT(repr cosine) WITH (IVF_NLIST = 1024);
```

