ALIBABA CLOUD

Alibaba Cloud

AnalyticDB for PostgreSQL
Advanced Developer Guide

Document Version: 20220308

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Use advanced extensions

1.1. Manage extensions

AnalyticDB for PostgreSQL is an online, distributed cloud data warehousing service that consists of multiple compute nodes and provides massively parallel processing (MPP) data warehousing solutions. AnalyticDB for PostgreSQL is developed by Alibaba Cloud based on open source Greenplum Database with enhanced, in-depth extensions.

Supported extensions

AnalyticDB for PostgreSQL supports the following extensions:

- PostGIS: processes geographic data. For more information, see Use PostGIS.
- MADlib: provides a machine learning function library.
- fuzzystrmatch: implements fuzzy match of strings.
- orafunc: provides compatibility with some Oracle functions. For more information, see Oracle Compatibility Functions. *Note: AnalyticDB for PostgreSQL V4.3 supports this extension, whereas AnalyticDB for PostgreSQL V6.0 does not.*
- orafce: provides compatibility with some Oracle functions. For more information, see orafce. *Note: AnalyticDB for PostgreSQL V6.0 supports this extension, whereas AnalyticDB for PostgreSQL V4.3 does not.*
- oss_ext: reads data from Object Storage Service (OSS).
- HyperLogLog: collects statistics. For more information, see Use HyperLogLog.
- PL/Java: compiles user-defined functions (UDFs) in PL/Java. For more information, see Create and use PL/Java UDFs.
- pgcrypto: provides cryptography functions that use encryption algorithms to ensure data security. Algorithms include MD5 message-digest (MD5), Secure Hash Algorithm 1 (SHA-1), SHA-224, SHA-256, SHA-384, SHA-512, Blowfish, Advanced Encryption Standard 128 (AES-128), AES-256, Raw Encryption, Pretty Good Privacy (PGP) symmetric keys, and PGP public keys. For more information, see pgcrypto.
- IntArray: provides integer array-related functions, operators, and indexes.
- RoaringBitmap: uses the RoaringBitmap efficient compression algorithm for bitmap operations.
- postgres_fdw: enables data access across databases. *Note: AnalyticDB for PostgreSQL V6.0 supports this extension, whereas AnalyticDB for PostgreSQL V4.3 does not.*
- tablefunc: includes various functions that return tables. *Note: AnalyticDB for PostgreSQL V6.0 supports this extension, whereas AnalyticDB for PostgreSQL V4.3 does not.*
- zhparser: provides full-text search of Chinese language. *Note: AnalyticDB for PostgreSQL V6.0 supports this extension, whereas AnalyticDB for PostgreSQL V4.3 does not.*

Create an extension

Execute the following statements to create an extension:

```
CREATE EXTENSION <extension name>;
CREATE SCHEMA <schema name>;
CREATE EXTENSION IF NOT EXISTS <extension name> WITH SCHEMA <schema name>;
```

Notice Before you create a MADlib extension, you must create a plpythonu extension.

```
CREATE EXTENSION plpythonu;
CREATE EXTENSION madlib;
```

Delete an extension

Execute the following statements to delete an extension.

Notice If an object depends on an extension that you want to delete, you must add the CASCADE keyword to delete the object.

```
DROP EXTENSION <extension name>;
DROP EXTENSION IF EXISTS <extension name> CASCADE;
```

1.2. Use the Laser computing engine

Laser is a computing engine developed by Alibaba Cloud. Laser is transparent to you and can improve the performance of complex computing. Laser provides more than twice the performance of the open source Greenplum Database for 1 GB, 100 GB, 1 TB, or 10 TB of data.

Limits

- We recommend that you use the ORCA optimizer.
- Laser is supported only in AnalyticDB for PostgreSQL V6.0 and later.

Enable or disable Laser

To enable or disable Laser, you can set the global user configuration (GUC) parameter **laser.enable** to on or off. Laser can be enabled or disabled for sessions, databases, or instances. For sessions, Laser automatically enters its default status when the sessions are ended. For databases, Laser immediately takes effect. For instances, Laser takes effect after you restart the instances. This section describes how to view and modify the status of Laser.

- Execute the following statement to view the status of Laser:

  ```
  SHOW laser.enable;
  ```

  Sample query result:

  ```
   laser.enable
  --------------
   on
  (1 row)
  ```

- Execute the following statement to enable Laser for sessions:

```
SET laser.enable = on;
```

- Execute the following statement to disable Laser for sessions:

```
SET laser.enable = off;
```

- Execute the following statement to enable Laser for a database:

```
ALTER DATABASE ${DBNAME} SET laser.enable = on;
```

- Execute the following statement to disable Laser for a database:

```
ALTER DATABASE ${DBNAME} SET laser.enable = off;
```

> **Note**
>
> - You are not allowed to directly enable or disable Laser for instances. We recommend that you enable or disable Laser for sessions or databases. To enable or disable Laser for instances, Submit a ticket.
> - By default, Laser is disabled if the minor version is earlier than 6.3.4.0, and enabled if the minor version is 6.3.4.0 or later. For information about how to query and update the minor version, see Query the minor engine version and Upgrade the engine version.

Supported data types and operators

Laser supports the following data types:

- INT2, INT4, and INT8
- FLOAT4, FLOAT8, and NUMERIC
- DATE, TIME, TIMETZ, TIMESTAMP, and TIMESTAMPTZ
- VARCHAR, TEXT, and BPCHAR

Laser supports the following operators:

- =, <, <=, >, >=, <>, !=, BETWEEN, IS NOT NULL, IS NULL, and LIKE
- Logical operators: AND, OR, and NOT

1.3. Use HyperLogLog

Alibaba Cloud Database AnalyticDB for PostgreSQL is deeply optimized. In addition to the native Greenplum Database function, HyperLogLog is also supported to provide solutions for Internet advertisement analysis and industries with similar estimation analysis and computing requirements. In order to quickly estimate the PV, UV and other business indicators.

Create a HyperLogLog plug-in

Execute the following command to create HyperLogLog plug-in:

```
CREATE EXTENSION hll;
```

Basic types

- Execute the following statement to create a table containing the hll field:

```
create table agg (id int primary key,userids hll);
```

- Run the following command to convert int to hll_hashval:

```
select 1::hll_hashval;
```

Basic operators

- The hll type supports=,!=, =, <>, ||, and#.

```
select hll_add_agg(1::hll_hashval) = hll_add_agg(2::hll_hashval);
select hll_add_agg(1::hll_hashval) || hll_add_agg(2::hll_hashval);
select #hll_add_agg(1::hll_hashval);
```

- The hll_hashval type supports=,!=, =and <>.

```
select 1::hll_hashval = 2::hll_hashval;
select 1::hll_hashval <> 2::hll_hashval;
```

Basic functions

- hll_hash_boolean, hll_hash_smallint, and hll_hash_bigint.

```
select hll_hash_boolean(true);
select hll_hash_integer(1);
```

- hll_add_agg: converts the int format to the hll format.

```
select hll_add_agg(1::hll_hashval);
```

- hll_union: Union of hll.

```
select hll_union(hll_add_agg(1::hll_hashval),hll_add_agg(2::hll_hashval));
```

- hll_set_defaults: sets the precision.

```
select hll_set_defaults(15,5,-1,1);
```

- hll_print: displays debug information.

```
select hll_print(hll_add_agg(1::hll_hashval));
```

Examples

```
create table access_date (acc_date date unique, userids hll);
insert into access_date select current_date, hll_add_agg(hll_hash_integer(user_id)) from ge
nerate_series(1,10000) t(user_id);
insert into access_date select current_date-1, hll_add_agg(hll_hash_integer(user_id)) from
generate_series(5000,20000) t(user_id);
insert into access_date select current_date-2, hll_add_agg(hll_hash_integer(user_id)) from
generate_series(9000,40000) t(user_id);
postgres=# select #userids from access_date where acc_date=current_date;
     ? column?
------------------
 9725.85273370708
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-1;
     ? column?
------------------
 14968.6596883279
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-2;
     ? column?
------------------
 29361.5209149911
(1 row)
```

1.4. Use PostGIS

PostGIS is an extension of PostgreSQL and provides spatial features including objects, indexes, functions, and operators.

> **Note**
>
> - PostGIS conforms to Open Geospatial Consortium (OGC) standards.
> - AnalyticDB for PostgreSQL V4.3 and V6.0 support PostGIS 2.0.3 and PostGIS 2.5.3 respectively.

Common operations

1) Connect to an AnalyticDB for PostgreSQL instance

For more information, see Use client tools to connect to an instance.

2) Install the PostGIS extension for the first time

Execute the following statement to install the PostGIS extension:

```
create extension postgis;
```

Execute the following statements to query the version of PostGIS that is installed:

```
select postgis_version();
select postgis_full_version();
```

3) Create a table and insert spatial data into the table

Execute the following SQL statement to create a table with a geometry field:

```
create table testg ( id int, geom geometry )
distributed by (id);
```

The preceding statement indicates that the inserted spatial data is insensitive to geometry types such as Point, MultiPoint, Linestring, MultiLineString, Polygon, and MultiPolygon. Execute the following SQL statement to create a table with the required geometry type and spatial reference identifier (SRID):

```
create table test ( id int, geom geometry(point, 4326) )
distributed by (id);
```

The following SQL statements show how to insert data into a table with or without an SRID:

```
-- without srid
insert into testg values (1, ST_GeomFromText('point(116 39)'));
-- with srid
insert into test values (1, ST_GeomFromText('point(116 39)', 4326));
```

The following code shows how to use the Java Database Connectivity (JDBC) API to insert data:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class PGJDBC {
    public static void main(String args[]) {
        Connection conn = null;
        Statement stmt = null;
        try{
            Class.forName("org.postgresql.Driver");
            //conn = DriverManager.getConnection("jdbc:postgresql://<host>:3432/<database>"
,"<user>", "<password>");
            conn.setAutoCommit(false);
            stmt = conn.createStatement();
            String sql = "INSERT INTO test VALUES (1001, ST_GeomFromText('point(116 39)', 4
326) )";
            stmt.executeUpdate(sql);
            stmt.close();
            conn.commit();
            conn.close();
        } catch (Exception e) {
            System.err.println(e.getClass().getName() + " : " + e.getMessage());
            System.exit(0);
        }
        System.out.println("insert successfully");
    }
}
```

4) Manage spatial indexes

- Create a spatial index

```
create index idx_test_geom on test using gist(geom);
```

idx_test_geom is a custom index name, in which test is the table name and geom is the geometry column name.

- Query indexes in a table

```
select * from pg_stat_user_indexes
where relname='test';
```

- Query the size of an index

```
select pg_indexes_size('idx_test_geom');
```

- Rebuild an index

```
reindex index idx_test_geom;
```

- Delete an index

```
drop index idx_test_geom;
```

5) Use SQL statements to execute typical spatial queries

- Execute rectangular range queries

```
-- without srid
select st_astext(geom) from testg
where ST_Contains(ST_MakeBox2D(ST_Point(116, 39),ST_Point(117, 40)), geom);
-- with srid
select st_astext(geom) from test
where ST_Contains(ST_SetSRID(ST_MakeBox2D(ST_Point(116, 39),ST_Point(117, 40)), 4326), geom
);
```

The ST_MakeBox2D function creates an envelope that is also known as a rectangular polygon.

- Execute range queries based on geometry buffers

```
-- without srid
select st_astext(geom) from testg
where ST_DWithin(ST_GeomFromText('POINT(116 39)'), geom, 0.01);
-- with srid
select st_astext(geom) from test
where ST_DWithin(ST_GeomFromText('POINT(116 39)', 4326), geom, 0.01);
```

For more information about ST_DWithin, visit ST_DWithin.

- Check the intersections of polygons either inside or on the boundary

```
-- without srid
select st_astext(geom) from testg
where ST_Intersects(ST_GeomFromText('POLYGON((116 39, 116.1 39, 116.1 39.1, 116 39.1, 116 39))'), geom);
-- with srid
select st_astext(geom) from test
where ST_Intersects(ST_GeomFromText('POLYGON((116 39, 116.1 39, 116.1 39.1, 116 39.1, 116 39))', 4326), geom);
```

The ST_* operator is case-insensitive. For more information, visit Introduction to PostGIS.

> **Note**
>
> AnalyticDB for PostgreSQL V6.0 is not compatible with certain features of PostGIS such as **create extension postgis_topology**. Therefore, we recommend that you do not create table columns of the geography type. If you need to create table columns of the geography type, the default SRID value is 0 or 4326.

Scenarios

1) Electronic fence

A monitoring service provider of passenger traffic gathers positioning data by using GPS terminals on passenger cars. Common services include deviation alarm, frequency reminders for popular service areas, and warnings about driving conditions in certain areas such as accidents or puddly and icy roads. Such services are typical scenarios of electronic fence. Take driving condition warnings as an example. The region table can be replicated because specific regions have small amounts of fixed data. Data can be collected once and updated on a regular basis. The corresponding SQL statement is as follows:

```
CREATE TABLE ky_region (
  rid     serial,
  name    varchar(256),
  geom    geometry)
DISTRIBUTED REPLICATED;
```

After specific region data of the Polygon or MultiPolygon type is inserted, AnalyticDB for PostgreSQL collects the data by using the ANALYZE TABLE statement and creates Generalized Search Tree (GiST) indexes. Warnings can be classified into two categories: warnings triggered when the car is fully enclosed within the region, and warnings triggered when the car touches the boundary. Each warning type uses different spatial operators. The corresponding SQL statements are as follows:

```
-- Fully enclosed within the region
select rid, name from ky_region
where ST_Contains(geom, ST_GeomFromText('POINT(116 39)'));
-- Contact with boundary
select rid, name from ky_region
where ST_Intersects(geom, ST_GeomFromText('POINT(116 39)'));
```

Description: After the latitude and longitude of a specified point are provided, the SQL statement queries the records that contain or intersect with the specified point in the geom field. If no record is returned, the car has not entered into any regions. One record indicates that the car has entered once. Multiple records indicate that the car has entered into a specific region. If multiple records are returned, some regions are overlapped, and you need to check whether the overlapped regions are valid.

2) Smart transportation

In a smart traffic scenario, a database contains linetype trace tables and other business tables. AnalyticDB for PostgreSQL queries the ID of a historical driving trace in the historical trace table. The schema of the trace table is as follows:

```
create table vhc_trace_d (
 stat_date        text,
 trace_id         text,
 vhc_id           text,
 rid_wkt          geometry)
Distributed by (vhc_id) partition by LIST(stat_date)
(
 PARTITION p20191008 VALUES('20191008'),
 PARTITION p20191009 VALUES('20191009'),
 ......
);
```

AnalyticDB for PostgreSQL creates partitions for the trace table by day, collects statistics of data imported each day, and creates GiST spatial indexes for the partitions. The corresponding SQL statement is as follows:

```
SELECT trace_id FROM vhc_trace_d
WHERE ST_Intersects(
  ST_GeomFromText('Polygon((118.732461  29.207363,118.732366  29.207198,118.732511  29.205951,118.732296  29.205644,
                  118.73226  29.205469,118.732350  29.20470,118.731708  29.203399,118.731701  29.202401, 118.754689 29.213488,
                  118.750827 29.21316,118.750272 29.213337,118.749677 29.213257,118.748699 29.213388,118.747715 29.213206,
                  118.746580 29.213831,118.74639 29.213872,118.744989 29.213858,118.743442 29.213795,118.74174 29.213002,
                  118.735633 29.208167,118.734422 29.207699,118.733045 29.207450,118.732803 29.207342,118.732461  29.207363))'), rid_wkt);
```

Hundreds of millions of spatial queries on the trace table can be responded within less than 80 ms.

3) Shop traffic analysis

An Internet service provider analyzes their shop traffic by using AnalyticDB for PostgreSQL. A database has two business tables: an attendance table named User and a shop table named Shop. The schema of the tables is as follows:

```
-- user
create table user_label (
  ghash7          int,
  uid             int,
  workday_geo     geometry,
  weekend_geo     geometry)
distributed by (ghash7);
-- shop
create table user_shop (
  ghash7          int,
  sid             int,
  shop_poly       geometry)
distributed by (ghash7);
```

A skilled design of the business tables is to use Geohash or ZOrder coding to reduce geospatial dimensions to partition keys instead of creating geospatial indexes. The SQL statement to collect statistics of customer traffic is as follows:

```
SELECT COUNT(1)
FROM (
  SELECT DISTINCT T0.uid FROM user_label T0 JOIN user_shop T1
  ON T1.ghash7 = T0.ghash7
  WHERE T1.sid IN (1,2,3) AND (ST_Intersects(T0.workday_geo, T1.shop_poly)
                               OR ST_Intersects(T0.weekend_geo, T1.shop_poly))
) c;
```

1.5. Use compressed bitmap index

This topic describes how to use Roaring bitmaps in AnalyticDB for PostgreSQL. A bitmap is a common data structure that consists of values 0 and 1. A separate bitmap is created to house all possible values for each column. Each bit indicates whether a data row has a value in that column. A bitmap helps you check whether a value exists. It also enables you to expedite bitmap-related computing by using bitwise operations such as AND, OR, and ANDNOT. In the big data discipline, bitmaps are suitable for data queries and correlated computing workloads such as removing duplicate data, filtering data based on tags, and generating time series.

A conventional bitmap occupies a large amount of memory resources. Therefore, we recommend that you use compressed bitmaps. Roaring bitmaps are efficient compressed bitmaps that are used in almost all popular programming languages on various big data platforms.

Overview

In a Roaring bitmap, a 32-bit integer within the range of [0, n] is divided into two parts: the most significant 16 bits comprise a 2^16 chunk and the least significant 16 bits are stored in a container. The containers of a Roaring bitmap are stored in a dynamic array as the primary index. AnalyticDB for PostgreSQL supports two types of containers: array containers and bitmap containers. An array container is used to store sparse data while a bitmap container is used to store dense data. An array container can store up to 4,096 integers. A bitmap container can store more than 4,096 integers.

Based on the preceding storage structure, AnalyticDB for PostgreSQL can rapidly retrieve a specific value from a Roaring bitmap. Roaring bitmaps provide algorithms suitable for bitwise operations such as AND, OR, and XOR between the two types of containers. This enables Roaring bitmaps to deliver excellent storage and computing performance.

For more information, visit Roaring Bitmaps.

Procedure

Create the RoaringBitmap extension.

```
CREATE EXTENSION roaringbitmap;
```

Create a table with the roaringbitmap data type.

```
CREATE TABLE t1 (id integer, bitmap roaringbitmap);
```

Call the rb_build function to insert data of the roaringbitmap type.

```
-- Set the bit value of a data record in the array to 1.
INSERT INTO t1 SELECT 1,RB_BUILD(ARRAY[1,2,3,4,5,6,7,8,9,200]);
-- Set the bit values of multiple elements to 1 and aggregate the bit values into a Roaring bitmap.
INSERT INTO t1 SELECT 2,RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
```

Perform bitwise operations such as OR, AND, XOR, and ANDNOT.

```
SELECT RB_OR(a.bitmap,b.bitmap) FROM (SELECT bitmap FROM t1 WHERE id = 1) AS a,(SELECT bitmap FROM t1 WHERE id = 2) AS b;
```

Perform bitwise operations such as OR, AND, XOR, and BUILD to aggregate data and generate a new Roaring bitmap.

```
SELECT RB_OR_AGG(bitmap) FROM t1;
SELECT RB_AND_AGG(bitmap) FROM t1;
SELECT RB_XOR_AGG(bitmap) FROM t1;
SELECT RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
```

Calculate the cardinality of the Roaring bitmap. The cardinality indicates how many bit values are 1 in the Roaring bitmap.

```
SELECT RB_CARDINALITY(bitmap) FROM t1;
```

Obtain the subscripts of the bits whose values are 1.

```
SELECT RB_ITERATE(bitmap) FROM t1 WHERE id = 1;
```

Bitmap calculation functions

Function name	Input data	Message	Description	Examples
rb_build	integer[]	roaringbitmap	Creates a Roaring bitmap from an integer array.	`rb_build('{1,2,3,4,5}')`
rb_and	roaringbitmap,roaringbitmap	roaringbitmap	Performs an AND operation.	`rb_and(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_or	roaringbitmap,roaringbitmap	roaringbitmap	Performs an OR operation.	`rb_or(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_xor	roaringbitmap,roaringbitmap	roaringbitmap	Performs an XOR operation.	`rb_xor(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_andnot	roaringbitmap,roaringbitmap	roaringbitmap	Performs an ANDNOT operation.	`rb_andnot(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_cardinality	roaringbitmap	integer	Calculates the cardinality of a Roaring bitmap.	`rb_cardinality(rb_build('{1,2,3,4,5}'))`
rb_and_cardinality	roaringbitmap,roaringbitmap	Integer	Calculates the cardinality from an AND operation on two Roaring bitmaps.	`rb_and_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`

Function name	Input data	Message	Description	Examples
rb_or_cardinality	roaringbitmap,roaringbitmap	Integer	Calculates the cardinality from an OR operation on two Roaring bitmaps.	`rb_or_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_xor_cardinality	roaringbitmap,roaringbitmap	Integer	Calculates the cardinality from an XOR operation on two Roaring bitmaps.	`rb_xor_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_andnot_cardinality	roaringbitmap,roaringbitmap	Integer	Calculates the cardinality from an ANDNOT operation on two Roaring bitmaps.	`rb_andnot_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_is_empty	roaringbitmap	boolean	Checks whether a Roaring bitmap is empty.	`rb_is_empty(rb_build('{1,2,3,4,5}'))`
rb_equals	roaringbitmap,roaringbitmap	boolean	Checks whether two Roaring bitmaps are the same.	`rb_equals(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_intersect	roaringbitmap,roaringbitmap	boolean	Checks whether two Roaring bitmaps intersect.	`rb_intersect(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`

Function name	Input data	Message	Description	Examples
rb_remove	roaringbitmap,integer	roaringbitmap	Removes an offset from a Roaring bitmap.	`rb_remove(rb_build('{1,2,3}'),3)`
rb_flip	roaringbitmap,integer,integer	roaringbitmap	Flips specific offsets in a Roaring bitmap.	`rb_flip(rb_build('{1,2,3}'),2,3)`
rb_minimum	roaringbitmap	integer	Returns the smallest offset in a Roaring bitmap. If the Roaring bitmap is empty, the value -1 is returned.	`rb_minimum(rb_build('{1,2,3}'))`
rb_maximum	roaringbitmap	integer	Returns the largest offset in a Roaring bitmap. If the Roaring bitmap is empty, the value 0 is returned.	`rb_maximum(rb_build('{1,2,3}'))`
rb_rank	roaringbitmap,integer	Integer	Returns the number of elements that are smaller than or equal to a specified offset in a Roaring bitmap.	`rb_rank(rb_build('{1,2,3}'),3)`
rb_iterate	roaringbitmap	setof integer	Returns a list of offsets from a Roaring bitmap.	`rb_iterate(rb_build('{1,2,3}'))`

Bitmap aggregate functions

Function	Input data	Message	Description	Examples
rb_build_agg	Integer	roaringbitmap	Creates a Roaring bitmap from a group of offsets.	`rb_build_agg(1)`

Function	Input data	Message	Description	Examples
rb_or_agg	roaringbitmap	roaringbitmap	Performs an OR aggregate operation.	`rb_or_agg(rb_build('{1,2,3}'))`
rb_and_agg	roaringbitmap	roaringbitmap	Performs an AND aggregate operation.	`rb_and_agg(rb_build('{1,2,3}'))`
rb_xor_agg	roaringbitmap	roaringbitmap	Performs an XOR aggregate operation.	`rb_xor_agg(rb_build('{1,2,3}'))`
rb_or_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an OR aggregate operation on two Roaring bitmaps.	`rb_or_cardinality_agg(rb_build('{1,2,3}'))`
rb_and_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an AND aggregate operation on two Roaring bitmaps.	`rb_and_cardinality_agg(rb_build('{1,2,3}'))`
rb_xor_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an XOR aggregate operation on two Roaring bitmaps.	`rb_xor_cardinality_agg(rb_build('{1,2,3}'))`

1.6. Use the uuid-ossp extension

This topic describes how to use the uuid-ossp extension in AnalyticDB for PostgreSQL.

Introduction

The UUID data type is used to store universally unique identifiers (UUIDs). UUIDs are more unique than sequences in distributed systems.

A UUID consists of 32 hexadecimal digits. The standard format is a group of eight characters followed by three groups of four digits followed by a group of 12 digits. The groups are separated by hyphens (-). Example:

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

You can also use uppercase letters or braces ({}) to enclose a UUID in standard format, omit some or all hyphens (-), or add a hyphen after any group of four digits. Example:

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
{a0eebc99-9c0b4ef8-bb6d6bb9-bd380a11}
```

Note None of the available versions of AnalyticDB for PostgreSQL allow you to use a field of the UUID data type as the distribution key.

Installation

Execute the following statement to install uuid-ossp:

```
CREATE EXTENSION "uuid-ossp";
```

Note The account that is used to install uuid-ossp must be granted the rds_superuser permission.

Functions

- Functions for UUID generation

Function	Description
`uuid_generate_v1()`	This function generates a version 1 UUID. This involves the MAC address of a computer and a timestamp. **Note** This type of UUID reveals the identity of the computer that created the identifier and the time at which the identifier was created. Therefore, this type of UUID is unsuitable for security-sensitive applications.
`uuid_generate_v1mc()`	This function generates a version 1 UUID. This function differs from the `uuid_generate_v1()` function in that the `uuid_generate_v1mc()` function uses a random multicast MAC address to generate a UUID, whereas the `uuid_generate_v1()` function uses the real MAC address of a computer to generate a UUID.

Function	Description
`uuid_generate_v3(namespace uuid, name text)`	This function generates a version 3 UUID in the specified `namespace` by using the specified `name`. ○ The namespace is one of the constants that are returned by the `uuid_ns_*()` functions described in the Functions returning UUID constants table. ○ The `name` is an identifier in the specified `namespace`. Example: `SELECT uuid_generate_v3(uuid_ns_url(), 'example.com');` The `name` parameter is hashed based on the MD5 hashing algorithm. Therefore, no plaintext can be derived from the generated UUID. A UUID that is generated by using this function does not require a random algorithm. The UUID is not generated based on the environment variables that are required to run the system. As a result, the UUID can be reproduced.
`uuid_generate_v4()`	This function generates a version 4 UUID, which is generated based on random numbers.
`uuid_generate_v5(namespace uuid, name text)`	This function generates a version 5 UUID, which is similar to a version 3 UUID except that the SHA-1 hashing algorithm is used. Version 5 is preferred over version 3 because SHA-1 is considered to be more secure than MD5.

- Functions returning UUID constants

Function	Description
`uuid_nil()`	This function returns a nil UUID constant, which is not considered as a real UUID.
`uuid_ns_dns()`	This function returns a constant that designates the DNS namespace for UUIDs.
`uuid_ns_url()`	This function returns a constant that designates the URL namespace for UUIDs.
`uuid_ns_oid()`	This function returns a constant that designates the ISO object identifier (OID) namespace for UUIDs. **Note** This pertains to ASN.1 OIDs, which are unrelated to the OIDs used in PostgreSQL.
`uuid_ns_x500()`	This function returns a constant that designates the X.500 distinguished name (DN) namespace for UUIDs.

Examples

- Execute the following statement to generate a version 1 UUID:

```
SELECT uuid_generate_v1();
```

The following result is returned:

```
           uuid_generate_v1
--------------------------------------
 c7f83ba4-bd93-11e9-8674-40a8f01ec4e8
(1 row)
```

- Execute the following statement to generate a version 3 UUID:

```
SELECT uuid_generate_v3(uuid_ns_url(), 'example.com');
```

The following result is returned:

```
           uuid_generate_v3
--------------------------------------
 a0473a67-27a1-3c05-a2d1-5c134639347f
(1 row)
```

- Execute the following statement to generate a version 4 UUID:

```
SELECT uuid_generate_v4();
```

The following result is returned:

```
           uuid_generate_v4
--------------------------------------
 d7a8d47e-58e3-4bd9-9340-8553ac03d144
(1 row)
```

- Execute the following statement to generate a version 5 UUID:

```
SELECT uuid_generate_v5(uuid_ns_url(), 'example.com');
```

The following result is returned:

```
           uuid_generate_v5
--------------------------------------
 a5cf6e8e-4cfa-5f31-a804-6de6d1245e26
(1 row)
```

References

- OSSP official website
- PostgreSQL official website

1.7. Use zhparser to perform Chinese word segmentation

This topic describes how to use zhparser to perform Chinese word segmentation during a full-text search in AnalyticDB for PostgreSQL.

> **Note** Only AnalyticDB for PostgreSQL V6.0 supports full-text search.

Full-text search overview

By default, PostgreSQL performs word segmentation based on spaces and punctuation marks. PostgreSQL does not support Chinese word segmentation. AnalyticDB for PostgreSQL can be integrated with zhparser to support Chinese word segmentation.

In most cases, you can use one of the following methods to perform a full-text search:

- Query data in a table:

```
SELECT name FROM <table...>
WHERE to_tsvector('english', name) @@ to_tsquery('english', 'friend');
```

- Create a Generalized Inverted Index (GIN) index:

```
CREATE INDEX <idx_...> ON <table...> USING gin(to_tsvector('english', name));
```

Configure zhparser

1. Execute the following statement to install zhparser:

   ```
   CREATE EXTENSION zhparser;
   ```

   > **Note** The account that is used to install zhparser must be granted the rds_superuser permission.

2. Execute the following statement to configure zhparser as the Chinese text parser, and then set the name to zh_cn.

   ```
   CREATE TEXT SEARCH CONFIGURATION zh_cn (PARSER = zhparser);
   ```

   After the configuration is complete, you can run the `\dF` or `\dFp` command to view the configuration. Custom dictionaries are not supported.

3. Query the token types that are used for word segmentation.
   - Execute the following statement to query the dictionary configuration of zhparser:

     ```
     SELECT ts_token_type('zhparser');
     ```

     The following result is returned:

```
          ts_token_type
--------------------------------
 (97,a,"adjective,形容词")
 (98,b,"differentiation,区别词")
 (99,c,"conjunction,连词")
 (100,d,"adverb,副词")
 (101,e,"exclamation,感叹词")
 (102,f,"position,方位词")
 (103,g,"root,词根")
 (104,h,"head,前连接成分")
 (105,i,"idiom,成语")
 (106,j,"abbreviation,简称")
 (107,k,"tail,后连接成分")
 (108,l,"tmp,习用语")
 (109,m,"numeral,数词")
 (110,n,"noun,名词")
 (111,o,"onomatopoeia,拟声词")
 (112,p,"prepositional,介词")
 (113,q,"quantity,量词")
 (114,r,"pronoun,代词")
 (115,s,"space,处所词")
 (116,t,"time,时语素")
 (117,u,"auxiliary,助词")
 (118,v,"verb,动词")
 (119,w,"punctuation,标点符号")
 (120,x,"unknown,未知词")
 (121,y,"modal,语气词")
 (122,z,"status,状态词")
(26 rows)
```

- Execute the following statement to query the configuration of zh_cn:

```
SELECT * FROM pg_ts_config_map
 WHERE mapcfg=(SELECT oid FROM pg_ts_config WHERE cfgname='zh_cn');
```

4. Add or remove token types.
   - Add token types.

     Execute the following statement to add nouns, verbs, adjectives, idioms, exclamations, and temporary idioms as token types that are used for word segmentation:

     ```
     ALTER TEXT SEARCH CONFIGURATION zh_cn ADD MAPPING FOR n,v,a,i,e,l WITH simple;
     ```

   - Remove token types.

     Execute the following statement to remove nouns, verbs, adjectives, idioms, exclamations, and temporary idioms as token types that are used for word segmentation:

     ```
     ALTER TEXT SEARCH CONFIGURATION zh_cn DROP MAPPING IF exists FOR n,v,a,i,e,l;
     ```

5. Use the following two functions to test the Chinese word segmentation feature during a full-text search:
   - to_tsvector:

```
SELECT to_tsvector('zh_cn', '有两种方法进行全文检索');
```

The following result is returned:

```
              to_tsvector
---------------------------------------
 '全文检索':4 '方法':2 '有':1 '进行':3
(1 row)
```

- to_tsquery:

```
SELECT to_tsquery('zh_cn', '有两种方法进行全文检索');
```

The following result is returned:

```
             to_tsquery
-------------------------------------
 '有' & '方法' & '进行' & '全文检索'
(1 row)
```

References

- For more information about full-text search, see Full Text Search.
- For more information about the functions and operators that can be used for full-text search, see Text Search Functions and Operators.

1.8. Use hints

AnalyticDB for PostgreSQL provides the pg_hint_plan plug-in for you to modify and optimize execution plans by using hints. The pg_hint_plan plug-in can also be used to generate SQL patterns and register hints. Therefore, hint-optimized execution plans are automatically generated when SQL statements that use the same SQL pattern are executed.

Limits

The pg_hint_plan plug-in is available only in AnalyticDB for PostgreSQL minor engine version 6.3.7.0 or later. For information about how to query and update the minor engine version of an AnalyticDB for PostgreSQL instance, see Query the minor engine version and Upgrade the engine version.

Features

AnalyticDB for PostgreSQL uses a cost-based optimizer that utilizes data statistics instead of static rules. The optimizer estimates the cost of each possible execution plan for an SQL statement and chooses the minimum-cost plan for execution. The optimizer does its best to select the best execution plan, but the final execution plan may not be suitable for your scenario because it does not count the possible correlation between data.

The pg_hint_plan plug-in can use hints to modify and optimize SQL execution plans and register optimized SQL patterns and hints. This way, optimized execution plans can be automatically generated when SQL statements that use the same registered SQL pattern are executed, which improves the execution efficiency.

Supported hints

Type	Format	Description
Hints for Grand Unified Configuration (GUC) parameter setting	`Set(GUC-param value)`	Sets GUC parameters while the optimizer is running. For more information about the GUC parameters, see the "GUC parameters" section of this topic. GUC parameters take effect only when the optimizer is running and not in other phases such as the rewrite and execute phases. • To disable the ORCA optimizer for a statement, we recommend that you add `SET(optimizer off)` to the statement. • To enable the ORCA optimizer for a statement, we recommend that you add `SET(<ORCA parameter><value>)` to the statement.
Hints for scan methods	`SeqScan(table)`	Forces sequential scan on the table.
	`TidScan(table)`	Forces TID scan on the table.
	`IndexScan(table[ index...])`	Forces index scan on the table. You can specify an index.
	`IndexOnlyScan(table[ index...])`	Forces index-only scan on the table. You can specify an index.
	`BitmapScan(table[ index...])`	Forces bitmap index scan on the table.
	`NoSeqScan(table)`	Forbids sequential scan on the table.
	`NoTidScan(table)`	Forbids TID scan on the table.
	`NoIndexScan(table)`	Forbids index scan on the table.
	`NoIndexOnlyScan(table)`	Forbids index-only scan on the table.
	`NoBitmapScan(table)`	Forbids bitmap index scan on the table.
Hints for join methods	`NestLoop(table table[ table...])`	Forces nested loop for the joins that consist of the specified tables.
	`HashJoin(table table[ table...])`	Forces hash join for the joins that consist of the specified tables.
	`MergeJoin(table table[ table...])`	Forces merge join for the joins that consist of the specified tables.
	`NoNestLoop(table table[ table...])`	Forbids nested loop for the joins that consist of the specified tables.

Type	Format	Description
Note The hints for join methods must be used together with the **hints for join order**.	`NoHashJoin(table table[ table...])`	Forbids hash join for the joins that consist of the specified tables.
	`NoMergeJoin(table table[ table...])`	Forbids merge join for the joins that consist of the specified tables.
Hints for join order	`Leading(table table[ table...])`	Forces join order as specified.
	`Leading(<join pair>)`	Forces join order and directions as specified.
Hints for row number correction	`Rows(table table[ table...] correction)`	Corrects the row number of a result of the joins that consist of the specified tables. The available correction methods are absolute `#<n>`, addition `+ <n>`, subtract `-<n>`, and multiplication `* <n>`. `<n>` is a string that strtod can read. **Note** The hint Rows corrects the total number of rows, whereas the query result shows the average number of rows per node (total number of rows/number of nodes).

Note

- Hints other than the hints for GUC parameter settings take effect only on the PostgreSQL query optimizer, not on the ORCA optimizer.
- The hints cannot be used to modify the degree of parallelism (DOP).

Examples:

- Hints for GUC parameter settings

  GUC parameter settings made when the optimizer is running take effect on both the ORCA and query optimizers.

  - Disable the ORCA optimizer.

    ```
    /*+ SET(optimizer off) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
    ```

    After the ORCA optimizer is disabled, it is not used.

- Enable the ORCA optimizer.

```
/*+ SET(optimizer on) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  After the ORCA optimizer is enabled, it is used. The ORCA optimizer is used in most cases and not used only in specific scenarios such as single-table queries and excessive partitioned tables.

- Forcefully enable the ORCA optimizer.

```
/*+ SET(optimizer on) SET(rds_optimizer_options 0) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  After the ORCA optimizer is forcefully enabled, it is always used. The ORCA optimizer is not used only when it cannot create plans.

- Forcefully enable the ORCA optimizer and disable HashJoin of the ORCA optimizer.

```
/*+ SET(optimizer on) SET(rds_optimizer_options 0) SET(optimizer_enable_hashjoin off) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

- Hints for scan methods

  Hints for scan methods apply only to the query optimizer. You must execute the following statement to disable the ORCA optimizer:

```
SET optimizer to off;
```

  - Forces index scan on table t1.

```
/*+ Indexscan(t1) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  - Forbids index scan on table t1.

```
/*+ NoIndexscan(t1) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  - Forces bitmap index scan on table t1 by using the t1_val bitmap index.

```
/*+ Bitmapscan(t1 t1_val) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  - Forces index-only scan on table t1.

```
/*+ Indexonlyscan(t1) */EXPLAIN SELECT t2.*, t1.val FROM t1 JOIN t2 ON t1.val = t2.val;
```

    > **Note** Index-only scan can be used only on index-only columns.

  - Forces TID scan on table t1.

```
/*+ Tidscan(t1) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val where t1.ctid = '(1,2)';
```

    > **Note** TID scan can be used only on tables that contain TID conditions.

- Hints for join methods and join order

  Hints for join methods and join order apply only to the query optimizer. You must execute the following statement to disable the ORCA optimizer:

```
SET optimizer to off;
```

- Perform merge join in which table t1 is the left table.

```
/*+ Leading((t1 t2)) MergeJoin(t1 t2) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2
.val;
```

- Perform nested loop join in which table t1 is the left table.

```
/*+ Leading((t1 t2)) NestLoop(t1 t2) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.
val;
```

- Do not perform hash join in which table t1 is the left table.

```
/*+ Leading((t1 t2)) NoHashJoin(t1 t2) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t
2.val;
```

- Perform hash join for t2 and t3 and then perform nest loop join with t1.

```
/*+ Leading(((t2 t3) t1)) HashJoin(t2 t3) NestLoop(t2 t3 t1) */EXPLAIN SELECT * FROM t1
, t2, t3 WHERE t1.val = t2.val and t2.val = t3.val;
```

- Hints for row number correction

  Hints for row number correction apply only to the query optimizer. You must execute the following statement to disable the ORCA optimizer:

```
SET optimizer to off;
```

  - Increase the total number of rows by 100 times in the table obtained by joining t1 and t2.

```
/*+ Rows(t1 t2 *100) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  - Decrease the total number of rows by 100 times in the table obtained by joining t1 and t2.

```
/*+ Rows(t1 t2 *0.01) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  - Increase the total number of rows by 100 in the table obtained by joining t1 and t2.

```
/*+ Rows(t1 t2 +100) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  - Subtract 100 from the total number of rows in the table obtained by joining t1 and t2.

```
/*+ Rows(t1 t2 -100) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

  - Corrects the total number of rows to 100 in the table obtained by joining t1 and t2.

```
/*+ Rows(t1 t2 #100) */EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.val = t2.val;
```

GUC parameters

Parameter	Default value	Description

Parameter	Default value	Description
pg_hint_plan.enable_hint	on	Specifies whether to enable the hint plan. Valid values: • on: enables the hint plan. • off: disables the hint plan.
pg_hint_plan.enable_hint_table	off	Specifies whether to enable the hint registration feature. Valid values: • on: enables the hint registration feature. • off: disables the hint registration feature.
pg_hint_plan.jumble_mode	off	Specifies whether to use object identifiers (OIDs) to identify objects in parameterized SQL statements, such as tables, functions, and operators. Valid values: • on: uses OIDs to identify objects. After an object is deleted, another object created by using the same name as the deleted object is considered a different object. • off: uses schemas and object names to identify objects. Objects with the same name in the same schema are considered the same. Note We recommend that you do not change this parameter frequently. If you change this parameter, previous registered hints cannot be used.
pg_hint_plan.parse_messages	info	Specifies the log level of the hint parse error. Valid values: error, warning, notice, info, log, and debug[1-5].
pg_hint_plan.message_level	log	Specifies the log level of errors in phases other than parse hint. Valid values: error, warning, notice, info, log, and debug[1-5].

Enable hints

1. Load and install the pg_hint_plan plug-in.
   - For temporary use of hints, perform the following steps to load and install the pg_hint_plan plug-in:
     a. Connect to the database. For more information, see Use client tools to connect to an instance.
     b. Execute the following statement to install the plug-in:

```
CREATE EXTENSION pg_hint_plan;
```

Note The preceding operations take effect only on the current session.

- For permanent use of hints, perform the following steps to load and install the pg_hint_plan plug-in:
  a. Submit a ticket to contact the technical support personnel to load the pg_hint_plan plug-in.
  b. Execute the following statement to install the plug-in:

```
CREATE EXTENSION pg_hint_plan;
```

> **Note** Hints can be enabled only for databases that have the pg_hint_plan plug-in installed.

2. (Optional)Enable hint registration.
   - For temporary use, execute the following statement to enable hint registration for the current session:

```
SET pg_hint_plan.enable_hint_table to on;
```

   - For permanent use, Submit a ticket contact the technical support personnel to enable hint registration.

> **Note** After the technical support personnel enables hint registration, you must reconnect to the database to make hint registration take effect.

3. (Optional)Register hints. You can modify the hint_plan.hints table by using functions to register hints. For more information, see the "Register hints" section of this topic.

   After hints are registered, hint-optimized execution plans are automatically generated when SQL statements that use the same SQL pattern are executed.

Register hints

If you want hints to be automatically applied to SQL statements that use the same SQL pattern or if hints can not be added to SQL statements, you can register the hints by adding them to the hint_plan.hints system table. After hints are registered, the hints are automatically applied to SQL statements that use the same SQL pattern.

The following table describes the structure of the hint_plan.hints table.

Column	Type	Description
id	integer	The unique number that identifies a hint. This column is filled in automatically by sequence.
norm_query_string	text	The SQL pattern to which the hint applies. SQL patterns are SQL statements that do not contain parameters or constants.
application_name	text	The application registered with the hint. The default value is an empty string (`''`), which indicates that the hint applies to all applications. In the following examples, the value of this column is an empty string (`''`). The application_name column has the unique key constraint.

Column	Type	Description
hints	text	The hint to be registered. The hints column has the unique key constraint.
query_hash	bigint	The hash value of the parameterized SQL pattern, which is the unique identifier of standard SQL statements. The query_hash column has the unique key constraint.
enable	boolean	Specifies whether to enable the hint. You can apply only a single hint to an SQL pattern.
prepare_param_strings	text	Records the parameters if the PREPARE statement is used.

Note You can query the hint_plan.hints table but we recommend that you do not directly modify it. We recommend that you modify the table by means of modifying functions.

The following section describes the functions used for hint registration.

- Function used to obtain the parameters of SQL statements

```
hint_plan.gp_hint_query_parameterize(<query>, <application_name>)
```

Parameter	Description
`query`	The SQL statement that contains the hint.
`application_name`	The application registered with the hint. The value is an empty string (`''`).

This function is used to obtain parameters of the SQL statement that contains the hint. The following table describes the parameters that can be obtained.

Parameter	Description
`query_hash`	The hash value of the parameterized SQL pattern, which is the unique identifier of standard SQL statements.
`norm_query_string`	The SQL pattern.
`comment_hints`	The hint.
`first_matched_hint_in_table`	The hints in the hint_plan.hints table that match the SQL pattern.
`prepare_param_strings`	The parameters in the SQL statement.

Example:

```
SELECT * FROM hint_plan.gp_hint_query_parameterize('/*+ MergeJoin(t1 t2) Leading((t1 t2))
*/SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < 100 and t2.val > 20;');
```

The following information is returned:

```
-[ RECORD 1 ]---------------+-------------------------------------------------------------
--------------
query_hash                  | -4733464863014584191
norm_query_string           | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 an
d t2.val > $2;
comment_hints               | MergeJoin(t1 t2) Leading((t1 t2))
first_matched_hint_in_table | HashJoin(t1 t2) Leading((t1 t2))
prepare_param_strings       | {}
```

- Function used to register hints

```
hint_plan.insert_hint_table(<query>, <application_name>)
```

Parameter	Description
`query`	The SQL statement that contains the hint.
`application_name`	The application registered with the hint. The value is an empty string (`''`).

This function can be used to register different hints for the same SQL pattern. When you insert hints with the same SQL pattern, hint name, and application name, no new hints are added to the hint_plan.hints table. The inserted hint is enabled while existing hints are disabled.

Example:

```
SELECT hint_plan.insert_hint_table('/*+ MergeJoin(t1 t2) Leading((t1 t2)) */SELECT * FROM
t1, t2 WHERE t1.id = t2.id and t1.val < 100 and t2.val > 1;');
```

The following information is returned:

```
                                                                    insert_hint_table
-------------------------------------------------------------------------------------------
-------------------------------------------------------------
 (1,"SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2;","","Merge
Join(t1 t2) Leading((t1 t2)) ",-4733464863014584191,t,{})
(1 row)
```

- Function used to modify hints

```
hint_plan.upsert_hint_table(<query>, <application_name>)
```

Parameter	Description
`query`	The SQL statement that contains the hint.
`application_name`	The application registered with the hint. The value is an empty string (`''`).

If the SQL pattern used by an SQL statement uses an existing hint, the existing hint in the hint_plan.hints table is replaced with the hint contained in the SQL statement specified by `query`.

Example:

i. Query the existing hint in the hint_plan.hints table.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id |                            norm_query_string                             | app
lication_name |               hints                |      query_hash      | enable |
prepare_param_strings
----+--------------------------------------------------------------------------+----
-------------+------------------------------------+----------------------+--------+-
---------------------
  1 | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2; |
| MergeJoin(t1 t2) Leading((t1 t2))  | -4733464863014584191 | f      | {}
  2 | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2; |
| Nestloop(t1 t2) Leading((t1 t2))   | -4733464863014584191 | t      | {}
(2 rows)
```

ii. Invoke the function to modify the hint.

```
SELECT hint_plan.upsert_hint_table('/*+ HashJoin(t1 t2) Leading((t1 t2)) */SELECT * F
ROM t1, t2 WHERE t1.id = t2.id and t1.val < 100 and t2.val > 1;');
```

The following information is returned:

```
                                                                        upsert_hint_table
------------------------------------------------------------------------------------------
------------------------------------------------------------
 (2,"SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2;","","H
ashJoin(t1 t2) Leading((t1 t2)) ",-4733464863014584191,t,{})
(1 row)
```

iii. Query the modified hint in the hint_plan.hints table.

```
SELECT * FROM hint_plan.hints;
```

The hint of the SQL pattern changes `Nestloop(t1 t2) Leading((t1 t2))` from to `HashJoin(t1 t2) Leading((t1 t2))`. The following information is returned:

```
 id |                            norm_query_string                             | app
lication_name |               hints                |      query_hash      | enable |
prepare_param_strings
----+--------------------------------------------------------------------------+----
-------------+------------------------------------+----------------------+--------+-
---------------------
  1 | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2; |
| MergeJoin(t1 t2) Leading((t1 t2))  | -4733464863014584191 | f      | {}
  2 | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2; |
| HashJoin(t1 t2) Leading((t1 t2))   | -4733464863014584191 | t      | {}
(2 rows)
```

- Function used to delete hints
  - Delete the hint with the specified ID.

    ```
    hint_plan.delete_hint_table(<id>)
    ```

  - Delete the hint defined by the specified SQL statement, hint name, and application name.

    ```
    hint_plan.delete_hint_table(<query>, <hint>, <application_name>)
    ```

  - Delete the hint defined by the specified SQL statement and application name.

    ```
    hint_plan.delete_all_hint_table(<query>, <application_name>)
    ```

Parameter	Description
`id`	The hint ID in the hint_plan.hints table.
`query`	The SQL statement that may or may not contain the hint.
`hint`	The hint.
`application_name`	The application registered with the hint. The value is an empty string (`''`).

Examples:

Query the original hint_plan.hints table.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id |                            norm_query_string                             | application_name |                      hints                       |      query_hash      | enable | prepare_param_strings
----+--------------------------------------------------------------------------+------------------+--------------------------------------------------+----------------------+--------+-----------------------
  1 | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2; |                  | MergeJoin(t1 t2) Leading((t1 t2))                | -4733464863014584191 | f      | {}
  2 | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2; |                  | HashJoin(t1 t2) Leading((t1 t2))                 | -4733464863014584191 | t      | {}
  3 | select * from t1 join t2 on t1.val = t2.val;                             |                  | set(optimizer on) set(rds_optimizer_options 0)   | -2169095602568752481 | f      | {}
  4 | select * from t1 join t2 on t1.val = t2.val;                             |                  | set(optimizer off)                               | -2169095602568752481 | t      | {}
(4 rows)
```

- Delete the hint with the specified ID.

```
SELECT hint_plan.delete_hint_table(1);
```

The following information is returned:

```
WARNING:  "max_appendonly_tables": setting is deprecated, and may be removed in a futur
e release.
                                                                    delete_hint_table
------------------------------------------------------------------------------------------
------------------------------------------------------------
 (1,"SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2;","","Mer
geJoin(t1 t2) Leading((t1 t2)) ",-4733464863014584191,f,{})
(1 row)
```

Query the hint_plan.hints table with the specified hint deleted.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id |                            norm_query_string                             | appli
cation_name |                      hints                       |      query_hash      |
enable | prepare_param_strings
----+--------------------------------------------------------------------------+------
-----------+--------------------------------------------------+----------------------+-
-------+-----------------------
  2 | SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2; |
| HashJoin(t1 t2) Leading((t1 t2))                | -4733464863014584191 | t      | {}
  3 | select * from t1 join t2 on t1.val = t2.val;                             |
| set(optimizer on) set(rds_optimizer_options 0)  | -2169095602568752481 | f      | {}
  4 | select * from t1 join t2 on t1.val = t2.val;                             |
| set(optimizer off)                              | -2169095602568752481 | t      | {}
(3 rows)
```

- Delete the hint defined by the specified SQL statement, hint name , and application name.

```
SELECT hint_plan.delete_hint_table('SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val
< 5 and t2.val > 1;', 'HashJoin(t1 t2) Leading((t1 t2))');
```

The following information is returned:

```
                                                                        delete_hint_table
------------------------------------------------------------------------------------------
------------------------------------------------------------
 (2,"SELECT * FROM t1, t2 WHERE t1.id = t2.id and t1.val < $1 and t2.val > $2;","","Has
hJoin(t1 t2) Leading((t1 t2)) ",-4733464863014584191,t,{})
(1 row)
```

Query the hint_plan.hints table with the specified hint deleted.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id |              norm_query_string               | application_name |
hints                       |      query_hash      | enable | prepare_param_strings
----+----------------------------------------------+------------------+----------------
---------------------------------+----------------------+--------+---------------------
--
  3 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
n) set(rds_optimizer_options 0)  | -2169095602568752481 | f      | {}
  4 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
ff)                              | -2169095602568752481 | t      | {}
(2 rows)
```

- Delete the hint defined by the specified SQL statement and application name.

```
SELECT hint_plan.delete_all_hint_table('select * from t1 join t2 on t1.val = t2.val;');
```

The following information is returned:

```
                                                    delete_all_hint_table
------------------------------------------------------------------------------------------------------------------------
 (3,"select * from t1 join t2 on t1.val = t2.val;","","set(optimizer on) set(rds_optimizer_options 0) ",-2169095602568752481,f,{})
 (4,"select * from t1 join t2 on t1.val = t2.val;","","set(optimizer off) ",-2169095602568752481,t,{})
(2 rows)
```

Query the hint_plan.hints table with the specified hint deleted.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id | norm_query_string | application_name | hints | query_hash | enable | prepare_param_strings
----+-------------------+------------------+-------+------------+--------+----------------------
(0 rows)
```

- Function used to enable hints
  - Enable the hint with the specified ID. After the specified hint is enabled, other hints for the same SQL pattern are not applied.

```
hint_plan.enable_hint_table(<id>)
```

  - Enable the hint defined by the specified SQL statement, hint name, and application name. After the specified hint is enabled, other hints for the same SQL pattern are disabled.

```
hint_plan.enable_hint_table(<query>, <hint>, <application_name>)
```

  - Disable the hint with the specified ID.

```
hint_plan.disable_hint_table(<id>)
```

  - Disable the hint defined by the specified SQL statement, hint name, and application name.

```
hint_plan.disable_hint_table(<query>, <hint>, <application_name>)
```

  - Disable the hint defined by the specified SQL statement and application name.

```
hint_plan.disable_all_hint_table(<query>, <application_name>)
```

Parameter	Description
`id`	The hint ID in the hint_plan.hints table.
`query`	The SQL statement that may or may not contain the hint.

Parameter	Description
`hint`	The hint.
`application_name`	The application registered with the hint. The value is an empty string (`''`).

Examples:

Query the original hint_plan.hints table.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id |              norm_query_string               | application_name |
hints                       |      query_hash      | enable | prepare_param_strings
----+----------------------------------------------+------------------+------------------
-------------------------------+----------------------+--------+----------------------
  5 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer off
)                             | -2169095602568752481 | f      | {}
  6 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer on)
set(rds_optimizer_options 0)  | -2169095602568752481 | t      | {}
(2 rows)
```

- Disable the hint with the specified ID.

  ```
  SELECT hint_plan.disable_hint_table(6);
  ```

  The following information is returned:

  ```
                                                                  disable_hint_table
  ------------------------------------------------------------------------------------------------
  ------------------------------------------
   (6,"select * from t1 join t2 on t1.val = t2.val;","","set(optimizer on) set(rds_optimi
  zer_options 0) ",-2169095602568752481,f,{})
  (1 row)
  ```

  Query the hint_plan.hints table with the specified hint disabled.

  ```
  SELECT * FROM hint_plan.hints;
  ```

  The following information is returned:

  ```
   id |              norm_query_string               | application_name |
  hints                       |      query_hash      | enable | prepare_param_strings
  ----+----------------------------------------------+------------------+----------------
  -------------------------------+----------------------+--------+----------------------
  --
    5 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
  ff)                           | -2169095602568752481 | f      | {}
    6 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
  n) set(rds_optimizer_options 0)  | -2169095602568752481 | f      | {}
  (2 rows)
  ```

- Enable the hint with the specified ID.

```
SELECT hint_plan.enable_hint_table(5);
```

The following information is returned:

```
                                        enable_hint_table
-----------------------------------------------------------------------------------------
----------------
 (5,"select * from t1 join t2 on t1.val = t2.val;","","set(optimizer off) ",-2169095602
568752481,t,{})
(1 row)
```

Query the hint_plan.hints table with the specified hint enabled.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id |              norm_query_string               | application_name |
hints                       |      query_hash      | enable | prepare_param_strings
----+----------------------------------------------+------------------+----------------
---------------------------------+----------------------+--------+---------------------
--
  6 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
n) set(rds_optimizer_options 0)  | -2169095602568752481 | f      | {}
  5 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
ff)                              | -2169095602568752481 | t      | {}
(2 rows)
```

- Enable the hint defined by the specified SQL statement and application name.

```
SELECT hint_plan.enable_hint_table('select * from t1 join t2 on t1.val = t2.val;', 'set
(optimizer off)');
```

The following information is returned:

```
                                        enable_hint_table
------------------------------------------------------------------------------------------
----------------
 (5,"select * from t1 join t2 on t1.val = t2.val;","","set(optimizer off) ",-2169095602
568752481,t,{})
(1 row)
```

Query the hint_plan.hints table with the specified hint enabled.

```
SELECT * FROM hint_plan.hints;
```

The following information is returned:

```
 id |              norm_query_string               | application_name |
hints                       |      query_hash      | enable | prepare_param_strings
----+----------------------------------------------+------------------+----------------
---------------------------------+----------------------+--------+---------------------
--
  6 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
n) set(rds_optimizer_options 0)  | -2169095602568752481 | f      | {}
  5 | select * from t1 join t2 on t1.val = t2.val; |                  | set(optimizer o
ff)                              | -2169095602568752481 | t      | {}
(2 rows)
```

Uninstall the pg_hint_plan plug-in

If you no longer need hints, you can execute the following statement to uninstall the pg_hint_plan plug-in:

```
DROP EXTENSION pg_hint_plan;
```

References

Use hints to modify execution plans

2.Use PL/Java

2.1. Create and use PL/Java UDFs

AnalyticDB for PostgreSQL allows you to compile and upload JAR software packages written in the PL/Java language, and use these JAR packages to create user-defined functions (UDFs). AnalyticDB for PostgreSQL supports PL/Java Community Edition 1.5.0 and JVM 1.8.

This topic describes how to create a PL/Java UDF. For more examples of PL/Java code, see PL/Java code. For the compiling method, see PL/Java documentation.

Procedure

1. In AnalyticDB for PostgreSQL, execute the following statement to create a PL/Java extension. You need to execute the statement only once for each database.

```
create extension pljava;
```

2. Compile the UDF. For example, you can use the following code to compile the `Test.java` file:

```
public class Test
{
    public static String substring(String text, int beginIndex,
              int endIndex)
            {
                    return text.substring(beginIndex, endIndex);
            }
}
```

3. Compile the `manifest.txt` file.

```
Manifest-Version: 1.0
Main-Class: Test
Specification-Title: "Test"
Specification-Version: "1.0"
Created-By: 1.7.0_99
Build-Date: 01/20/2016 21:00 AM
```

4. Run the following commands to compile and package the program:

```
javac Test.java
jar cfm analytics.jar manifest.txt Test.class
```

5. Upload the `analytics.jar` file that is generated in Step 4 to OSS by using the following OSS console command:

```
osscmd put analytics.jar oss://zzz
```

6. In AnalyticDB for PostgreSQL, execute the CREATE LIBRARY statement to import the file to AnalyticDB for PostgreSQL:

> **Notice** You can only use the filepath variable in the CREATE LIBRARY statement to import files. You can import only one file at a time. You can also execute the CREATE LIBRARY statement to import files as byte streams without the need to use OSS. For more information, see Use the CREATE LIBRARY statement.

```
create library example language java from 'oss://oss-cn-hangzhou.aliyuncs.com filepath=
analytics.jar id=xxx key=yyy bucket=zzz';
```

7. In AnalyticDB for PostgreSQL, execute the following statements to create and use the UDF:

```
 create table temp (a varchar) distributed randomly;
 insert into temp values ('my string');
 create or replace function java_substring(varchar, int, int) returns varchar as 'Test.
substring' language java;
 select java_substring(a, 1, 5) from temp;
```

2.2. Use the CREATE LIBRARY statement

AnalyticDB for PostgreSQL introduces the CREATE LIBRARY and DROP LIBRARY statements to allow you to import custom software packages. For more information, see Use PL/Java UDF.

This topic describes how to use the CREATE LIBRARY and DROP LIBRARY statements.

Syntax

```
CREATE LIBRARY library_name LANGUAGE [JAVA] FROM oss_location OWNER ownername
CREATE LIBRARY library_name LANGUAGE [JAVA] VALUES file_content_hex OWNER ownername
DROP LIBRARY library_name
```

Parameter description:

- `library_name`: the name of the library to be installed. If the library to be installed has the same name as an existing library, you must delete the existing library before you install the new library.
- `LANGUAGE [JAVA]`: the programming language to be used. Only PL/Java is supported.
- `oss_location`: the location of the package. You can specify an Object Storage Service (OSS) bucket and an object name. Only one object can be specified and the specified object cannot be a compressed file. Sample format:

  ```
  oss://oss_endpoint filepath=[folder/[folder/]...]/file_name id=userossid key=userosskey b
  ucket=ossbucket
  ```

  You can also use a temporary Security Token Service (STS) credential to access an OSS bucket. For more information, see . Sample format:

  ```
  oss://oss_endpoint filepath=[folder/[folder/]...]/file_name id=userossid key=userosskey t
  oken=usersecuritytoken bucket=ossbucket
  ```

- `file_content_hex`: the content of the file. The byte stream is in hexadecimal notation. For example, `73656c6563742031` indicates the hexadecimal byte stream of "select 1". You can use this syntax to import packages without the need to use OSS.
- `ownername`: specifies the name of the user.

- `DROP LIBRARY`: deletes a library.

Examples

- Example 1: Install a JAR package that is named `analytics.jar`.

```
create library example language java from 'oss://oss-cn-hangzhou.aliyuncs.com filepath=analytics.jar id=xxx key=yyy bucket=zzz';
```

- Example 2: Use a temporary STS credential to install a JAR package that is named `analytics.jar`.

```
create library example language java from 'oss://oss-cn-hangzhou.aliyuncs.com filepath=analytics.jar id=xxx key=yyy token=ttt bucket=zzz';
```

- Example 3: Import file content of which the byte stream is in hexadecimal notation.

```
create library  pglib LANGUAGE java VALUES '73656c6563742031' OWNER "myuser";
```

- Example 4: Delete a library.

```
drop library example;
```

- Example 5: View installed libraries.

```
select name, lanname from pg_library;
```

3.Use PL/Python

AnalyticDB for PostgreSQL allows you to create user-defined functions (UDFs) in the PL/Python procedural language.

Limits

- AnalyticDB for PostgreSQL does not support trigger functions in PL/Python.
- You cannot use cursors to update data. For example, you cannot use `UPDATE...WHERE CURRENT OF` or `DELETE...WHERE CURRENT OF`.
- AnalyticDB for PostgreSQL supports Python 2 only.

Create or delete a PL/Python plug-in

To create or delete a PL/Python plug-in, execute one of the following statements on an AnalyticDB for PostgreSQL instance. You do not have to execute the same statement twice on each database. PL/Python is considered an untrusted language. In this case, you must execute the statements by using a privileged account. In the following statements, the privileged account is admin, and the database on which you want to execute statements is testdb.

Create a PL/Python plug-in

```
$ psql -U admin -d testdb -c 'CREATE EXTENSION plpythonu;'
```

Delete a PL/Python plug-in

```
$ psql -U admin -d testdb -c 'DROP EXTENSION plpythonu;'
```

Use PL/Python to create functions

> **Notice** For security considerations, you do not have the permissions to create functions in PL/Python. To create such a function, submit a ticket. After your request is accepted, technical engineers help you create the function.

Create a function in PL/Python

```
CREATE FUNCTION return_int_array()
  RETURNS int[]
AS $$
  return [1, 2, 3, 4]
$$ LANGUAGE plpythonu;
```

Call the function

```
SELECT return_int_array();
return_int_array
---------------------
{1,11,21,31}
(1 row)
```

References

For more information about how to use Python, visit the Python official website.

For more information about how to use PL/Python, see PostgreSQL documentation.

4.AnalyticDB for PostgreSQL error codes

Each message that is sent by the AnalyticDB for PostgreSQL server is assigned a five-character error code that follows the conventions of the SQL standard for SQLSTATE codes. This topic describes all the error codes that are defined in PostgreSQL 8.1.

Error code	Description
Class 00: successful operation.	
00000	The error code returned because the operation is successful.
Class 01: warning.	
01000	The error code returned to indicate a warning.
0100C	The error code returned because dynamic result sets are returned.
01008	The error code returned because the value is padded with 0-bits.
01003	The error code returned because the NULL value is eliminated in the set function.
01007	The error code returned because the required permissions are not granted.
01006	The error code returned because the specified permissions are not revoked.
01004	The error code returned because the string is truncated on the right.
01P01	The error code returned because the feature is deprecated.
Class 02: no data. This is also a warning class as per the SQL standard.	
02000	The error code returned because no data is returned.

Error code	Description
02001	The error code returned because no additional dynamic result sets are returned.
Class 03: The execution of the SQL statement is not complete.	
03000	The error code returned because the execution of the SQL statement is not complete.
Class 08: connection exception.	
08000	The error code returned because a connection exception occurs.
08003	The error code returned because a connection does not exist.
08006	The error code returned because a connection failure occurs.
08001	The error code returned because the SQL client fails to establish a connection.
08004	The error code returned because the SQL server rejects the request from the SQL client to establish a connection.
08007	The error code returned because the resolution for the transaction is unknown.
08P01	The error code returned because the protocol is violated.
Class 09: trigger action exception.	
09000	The error code returned because an exception is thrown when an action is triggered.
Class 0A: feature not supported.	
0A000	The error code returned because the feature is not supported.

Error code	Description
Class 0B: invalid initialization of transactions.	
0B000	The error code returned because the transaction initialization is invalid.
Class 0F: locator exception.	
0F000	The error code returned because a locator exception occurs.
0F001	The error code returned because the locator specification is invalid.
Class 0L: invalid grantor.	
0L000	The error code returned because the grantor is invalid.
0LP01	The error code returned because the grant operation is invalid.
Class 0P: invalid role specification.	
0P000	The error code returned because the role specification is invalid.
Class 21: cardinality violation.	
21000	The error code returned because cardinality violations occur.
Class 22: data exception.	
22000	The error code returned because a data exception occurs.
2202E	The error code returned because the array subscript is invalid.
22021	The error code returned because characters are not in repertoire.
22008	The error code returned because the datetime field overflows.

Error code	Description
22012	The error code returned because a number is divided by zero.
22005	The error code returned because an error occurs in assignment.
2200B	The error code returned because an escape character conflict occurs.
22022	The error code returned because the indicator overflows.
22015	The error code returned because an internal field overflows.
2201E	The error code returned because the logarithm operation has invalid arguments.
2201F	The error code returned because the power function has invalid arguments.
2201G	The error code returned because the WIDTH_BUCKET function has invalid arguments.
22018	The error code returned because the cast specification has invalid character values.
22007	The error code returned because the datetime format is invalid.
22019	The error code returned because the escape character is invalid.
2200D	The error code returned because the escape octet is invalid.
22025	The error code returned because the escape sequence is invalid.
22P06	The error code returned because a non-standard escape character is used.
22010	The error code returned because the indicator parameter value is invalid.

Error code	Description
22020	The error code returned because the limit value is invalid.
22023	The error code returned because the parameter value is invalid.
2201B	The error code returned because the regular expression is invalid.
22009	The error code returned because the time zone displacement value is invalid.
2200C	The error code returned because an invalid escape character is used.
2200G	The error code returned because the most relevant types do not match.
22004	The error code returned because NULL values are not allowed.
22002	The error code returned because NULL values cannot be used for indicator parameters.
22003	The error code returned because the numeric value is out of range.
22026	The error code returned because lengths of strings do not match.
22001	The error code returned because the string is truncated on the right.
22011	The error code returned because a substring error occurs.
22027	The error code returned because a truncation error occurs.
22024	The error code returned because the C string is not terminated.
2200F	The error code returned because a zero-length character string is returned.
22P01	The error code returned because a floating point exception occurs.

Error code	Description
22P02	The error code returned because the text representation is invalid.
22P03	The error code returned because the binary representation is invalid.
22P04	The error code returned because the copy file format is invalid.
22P05	The error code returned because untranslatable characters are found.
Class 23: integrity constraint violation.	
23000	The error code returned because the integrity constraint is violated.
23001	The error code returned because the limit is violated.
23502	The error code returned because the NOT NULL rule is violated.
23503	The error code returned because the foreign key constraint is violated.
23505	The error code returned because the constraint on uniqueness is violated.
23514	The error code returned because the check constraint is violated.
Class 24: invalid cursor state.	
24000	The error code returned because the cursor state is invalid.
Class 25: invalid transaction state.	
25000	The error code returned because the transaction state is invalid.
25001	The error code returned because the SQL transaction is active.
25002	The error code returned because the branch transaction is active.

Error code	Description
25008	The error code returned because the held cursors require the same isolation level.
25003	The error code returned because the access mode for the branch transaction is inappropriate.
25004	The error code returned because the isolation level for the branch transaction is inappropriate.
25005	The error code returned because the branch transaction does not have an active SQL transaction.
25006	The error code returned because the SQL transaction is read-only.
25007	The error code returned because the schema and data statement cannot be mixed.
25P01	The error code returned because no active SQL transactions exist.
25P02	The error code returned because the SQL transaction is in the failed state.
Class 26: invalid SQL statement name.	
26000	The error code returned because the SQL statement name is invalid.
Class 27: triggered data change violation.	
27000	The error code returned because the triggered data change violates constraints.
Class 28: invalid authorization specification.	
28000	The error code returned because the authorization specification is invalid.

Error code	Description
Class 2B: Dependent privilege descriptors still exist.	
2B000	The error code returned because dependent privilege descriptors still exist.
2BP01	The error code returned because dependent objects still exist.
Class 2D: invalid transaction termination.	
2D000	The error code returned because the transaction termination is invalid.
Class 2F: SQL routine exception.	
2F000	The error code returned because an SQL routine exception occurs.
2F005	The error code returned because the executed function does not return statements.
2F002	The error code returned because the SQL data cannot be modified.
2F003	The error code returned because the SQL statements to be used are prohibited.
2F004	The error code returned because the SQL data cannot be read.
Class 34: invalid cursor name.	
34000	The error code returned because the cursor name is invalid.
Class 38: external routine exception.	
38000	The error code returned because an external routine exception occurs.
38001	The error code returned because prohibited SQL statements are contained.

Error code	Description
38002	The error code returned because the SQL data cannot be modified.
38003	The error code returned because the SQL statements to be used are prohibited.
38004	The error code returned because the SQL data cannot be read.
Class 39: exception in external routine invocation.	
39000	The error code returned because an exception occurs during external routine invocation.
39001	The error code returned because the returned SQL statement is invalid.
39004	The error code returned because NULL values are not allowed.
39P01	The error code returned because the trigger protocol is violated.
39P02	The error code returned because the SRF protocol is violated.
Class 3B: savepoint exception.	
3B000	The error code returned because a savepoint exception occurs.
3B001	The error code returned because the savepoint specification is invalid.
Class 3D: invalid database name.	
3D000	The error code returned because the database name is invalid.
Class 3F: invalid schema name.	
3F000	The error code returned because the schema name is invalid.

Error code	Description
Class 40: transaction rollback.	
40000	The error code returned because the transaction is rolled back.
40002	The error code returned because the constraint on transaction integrity is violated.
40001	The error code returned because the serialization fails.
40003	The error code returned because whether the statement execution is complete is unknown.
40P01	The error code returned because a deadlock is detected.
Class 42: syntax error or access rule violation	
42000	The error code returned because a syntax error occurs or the access rule is violated.
42601	The error code returned because a syntax error occurs.
42501	The error code returned because the required permissions are not granted.
42846	The error code returned because the data type cannot be converted.
42803	The error code returned because a grouping error occurs.
42830	The error code returned because the foreign key is invalid.
42602	The error code returned because the name is invalid.
42622	The error code returned because the length of the name exceeds the limit.

Error code	Description
42939	The error code returned because the name is reserved.
42804	The error code returned because data types do not match.
42P18	The error code returned because the data type is indeterminate.
42809	The error code returned because the object type is invalid.
42703	The error code returned because the column is undefined.
42883	The error code returned because the function is undefined.
42P01	The error code returned because the table is undefined.
42P02	The error code returned because the parameter is undefined.
42704	The error code returned because the object is undefined.
42701	The error code returned because the column is duplicate.
42P03	The error code returned because the cursor is duplicate.
42P04	The error code returned because the database is duplicate.
42723	The error code returned because the function is duplicate.
42P05	The error code returned because the prepared statement is duplicate.
42P06	The error code returned because the schema is duplicate.
42P07	The error code returned because the table is duplicate.
42712	The error code returned because the alias is duplicate.

Error code	Description
42710	The error code returned because the object is duplicate.
42702	The error code returned because no specific column is specified.
42725	The error code returned because no specific function is specified.
42P08	The error code returned because no specific parameter is specified.
42P09	The error code returned because no specific alias is specified.
42P10	The error code returned because the referenced column is invalid.
42611	The error code returned because the column definition is invalid.
42P11	The error code returned because the cursor definition is invalid.
42P12	The error code returned because the database definition is invalid.
42P13	The error code returned because the function definition is invalid.
42P14	The error code returned because the definition of a prepared statement is invalid.
42P15	The error code returned because the schema definition is invalid.
42P16	The error code returned because the table definition is invalid.
42P17	The error code returned because the object definition is invalid.
Class 44: WITH CHECK option violation.	
44000	The error code returned because the WITH CHECK option is violated.
Class 53: insufficient resources.	

Error code	Description
53000	The error code returned because the resources are insufficient.
53100	The error code returned because the disk is full.
53200	The error code returned because the memory is exhausted.
53300	The error code returned because the number of connections exceeds the limit.
Class 54: program limit exceeded.	
54000	The error code returned because the program limit is exceeded.
54001	The error code returned because the statement is complex.
54011	The error code returned because the number of columns exceeds the limit.
54023	The error code returned because the number of arguments exceeds the limit.
Class 55: object not in the required state.	
55000	The error code returned because the object is not in the required state.
55006	The error code returned because the object is in use.
55P02	The error code returned because the runtime parameters cannot be modified.
55P03	The error code returned because the lock is unavailable.
Class 57: operator intervention.	
57000	The error code returned because operator intervention occurs.

Error code	Description
57014	The error code returned because the query is canceled.
57P01	The error code returned because the system is shut down by an administrator.
57P02	The error code returned because the system is shut down when a crash occurs.
57P03	The error code returned because the system cannot be connected.
Class 58: system error.	
58030	The error code returned because an I/O error occurs.
58P01	The error code returned because the file is undefined.
58P02	The error code returned because the file is duplicate.
Class F0: configuration file error.	
F0000	The error code returned because an error occurs in the configuration file.
F0001	The error code returned because a lock file exists.
Class P0: Procedural Language/PostgreSQL (PL/pgSQL) error.	
P0000	The error code returned because a PL/pgSQL error occurs.
P0001	The error code returned because an exception is thrown.
Class XX: internal error.	
XX000	The error code returned because an internal error occurs.

Error code	Description
XX001	The error code returned because the data is corrupted.
XX002	The error code returned because the index is corrupted.