ALIBABA CLOUD

Alibaba Cloud

AnalyticDB for PostgreSQL
Advanced Developer Guide

Document Version: 20201204

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 services. It is developed by Alibaba Cloud based on 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 the fuzzy match of strings.
- orafunc: compatible with some Oracle functions,see Use orafunc

  .

  > **Note** AnalyticDB for PostgreSQL V4.3 supports this extension, while AnalyticDB for PostgreSQL V6.0 does not.

- orafce: compatible with some Oracle functions,see Use orafce.

  > **Note** AnalyticDB for PostgreSQL V4.3 supports this extension, while AnalyticDB for PostgreSQL V6.0 does not.

- oss_ext: reads data from OSS.
- HyperLogLog: collects statistics. For more information, see Use HyperLogLog.
- PL/Java: compiles user-defined functions (UDFs) in PL/Java. For more information, see Use PL/Java UDF.
- 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, while AnalyticDB for PostgreSQL V4.3 does not.

- tablefunc: includes various functions that return tables.

  > **Note** AnalyticDB for PostgreSQL V6.0 supports this extension, while AnalyticDB for PostgreSQL V4.3 does not.

Create an extension

Execute the following statements:

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

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

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

Delete an extension

Execute the following statements:

Notice If an object depends on an extension, 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 Odyssey compute engine

The Odyssey compute engine for AnalyticDB for PostgreSQL is independently developed by Alibaba Cloud and transparent to users to improve sophisticated computing performance. Compared with the native engine, Odyssey provides two-fold performance when the 22 SQL queries of the TPC-H benchmark are executed on 1 GB, 100 GB, 1 TB, and 10 TB of data.

Enable and disable Odyssey

You can use the enable_odyssey parameter of global user configuration (GUC) to enable and disable Odyssey. To enable Odyssey, set the parameter to on, and SELECT statements will be processed by Odyssey. To enable the native engine, set the parameter to off. The default value is off. You can enable Odyssey for sessions, databases, and clusters. When a session is ended, Odyssey is automatically disabled. When the parameter is set to on, Odyssey is immediately enabled for a database and will be enabled for a cluster after you restart the cluster. You can execute the following SQL statements to view the Odyssey status and modify the status for different levels.

```
--- View whether Odyssey is enabled:
show enable_odyssey;
--- Disable Odyssey for a session:
set enable_odyssey =off;
--- Enable Odyssey for a session:
set enable_odyssey = on;
--- Disable Odyssey for a database:
alter database ${DBNAME} set enable_odyssey=off;
--- Enable Odyssey for a database:
alter database ${DBNAME} set enable_odyssey=on;
```

Note We recommend that you enable Odyssey for databases and sessions. To enable Odyssey for your clusters, contact the Alibaba Cloud administrator.

Supported data types and operators

Odyssey supports the following data types:

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

Odyssey supports the following operators:

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

Note

- We recommend that you use the ORCA query optimizer.
- Odyssey is only supported in AnalyticDB for PostgreSQL 6.0 and later versions.

1.3. Use HyperLogLog

AnalyticDB for PostgreSQL is nested with native features of Greenplum Database, and also supports HyperLogLog. It provides solutions for industries with the Internet advertisement analysis requirements and requirements similar to estimation analysis computing to facilitate quick estimation of PV, UV, and other business metrics.

Create a HyperLogLog extension

Run the following command to create a HyperLogLog extension:

```
CREATE EXTENSION hll;
```

Basic types

- Run the following command 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

- The supported functions include hll_hash_boolean, hll_hash_smallint, hll_hash_bigint, and other hash functions.

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

- hll_add_agg: Used to convert int to the hll format.

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

- hll_union: The union of hll.

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

- hll_set_defaults: Used to set the precision.

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

- hll_print: Used for 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 generate_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 Connect to an AnalyticDB for PostgreSQL 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>", "<pass
word>");
      conn.setAutoCommit(false);
      stmt = conn.createStatement();
      String sql = "INSERT INTO test VALUES (1001, ST_GeomFromText('point(116 39)', 4326) )";
      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 bitmaps

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. If the number of integers to be stored is less than 4,096, use an array container. Otherwise, use a bitmap container.

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 value of more than one data record 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	Input	Output	Description	Example
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}'))

Function	Input	Output	Description	Example
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}'))
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}'))

Function	Input	Output	Description	Example
rb_intersect	roaringbitmap, roaringbitmap	boolean	Checks whether two Roaring bitmaps intersect.	rb_intersect(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
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	Negatives an offset 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	set of integer	Returns a list of offsets from a Roaring bitmap.	rb_iterate(rb_build('{1,2,3}'))

Bitmap aggregate functions

Function	Input	Output	Description	Example
rb_build_agg	integer	roaringbitmap	Creates a Roaring bitmap from a group of offsets.	`rb_build_agg(1)`
rb_or_agg	roaringbitmap	roaringbitmap	Performs an OR operation.	`rb_or_agg(rb_build('{1,2,3}'))`
rb_and_agg	roaringbitmap	roaringbitmap	Performs an AND operation.	`rb_and_agg(rb_build('{1,2,3}'))`
rb_xor_agg	roaringbitmap	roaringbitmap	Performs an XOR operation.	`rb_xor_agg(rb_build('{1,2,3}'))`
rb_or_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an OR 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 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 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.

Installation

Execute the following statement to install the uuid-ossp extension:

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

> **Note** You can only install the uuid-ossp extension as the superuser or rds_superuser user.

Overview

After you install the `uuid-ossp` extension, the system automatically creates the `UUID` data type and supports B-tree indexes.

```
mydb=> create extension "uuid-ossp";
CREATE EXTENSION
mydb=> \dT
   List of data types
 Schema | Name | Description
--------+------+-------------
 public | uuid |
(1 row)
```

The UUID data type is used to store universally unique identifiers (UUIDs) defined by standards such as RFC 4122 and ISO/IEF 9834-8:2005. UUIDs are also known as globally unique identifiers (GUIDs) in some systems. No identical UUIDs can be generated by the same algorithm or any other methods. In a distributed database system, UUIDs ensure uniqueness better than sequences, because sequences can only ensure uniqueness in a single database.

A standard UUID is a 128-bit hexadecimal sequence in lowercase. It consists of 32 hexadecimal digits displayed in five groups that are separated with hyphens (-). These five groups consist of one 8-digit group, three 4-digit groups, and one 12-digit group. Example:

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

In addition to standard UUIDs, the uuid-ossp extension supports UUIDs in other formats such as UUIDs written in uppercase, UUIDs with some or all hyphens (-) deleted, UUIDs in which every four digits are separated with a hyphen (-), and UUIDs enclosed in a pair of braces {}. Examples:

```
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 choose a column of the UUID data type as the partition key.

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 when the creation occurred. Therefore, this type of UUID is unsuitable for certain security-sensitive applications.
`uuid_generate_v1mc()`	This function generates a version 1 UUID. It differs from the `uuid_generate_v1()` function in the following aspect: 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.
`uuid_generate_v3(namespace uuid, name text)`	This function generates a version 3 UUID in the given `namespace` by using the specified `name`. ◦ The namespace must be one of the special constants that are generated by the `uuid_ns_*()` function described in the "Functions returning UUID constants" table. ◦ The `name` is an identifier in the specified `namespace`. For example: `SELECT uuid_generate_v3(uuid_ns_url(), 'http://www.postgresql.org');` The `name` parameter is MD5-hashed. Therefore, no plaintext can be derived from the generated UUID. A UUID generated by using this function does not require a random algorithm or depend on any environment elements related to system running and is therefore reproducible.
`uuid_generate_v4()`	This function generates a version 4 UUID, which is derived entirely from random numbers.
`uuid_generate_v5(namespace uuid, name text)`	This function generates a version 5 UUID, which works like a version 3 UUID except that SHA-1 is used as a hash algorithm. 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 generates a nil UUID constant, which is not considered as a real UUID.
uuid_ns_dns()	This function generates a constant that designates the DNS namespace for UUIDs.
uuid_ns_url()	This function generates a constant that designates the URL namespace for UUIDs.
uuid_ns_oid()	This function generates 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 generates a constant that designates the X.500 distinguished name (DN) namespace for UUIDs.

Examples

```
mydb=# create extension "uuid-ossp";
CREATE EXTENSION
mydb=# SELECT uuid_generate_v1();
      uuid_generate_v1
--------------------------------------
 c7f83ba4-bd93-11e9-8674-40a8f01ec4e8
(1 row)
mydb=# SELECT uuid_generate_v3(uuid_ns_url(), 'http://www.postgresql.org');
      uuid_generate_v3
--------------------------------------
 cf16fe52-3365-3a1f-8572-288d8d2aaa46
(1 row)
mydb=# SELECT uuid_generate_v4();
      uuid_generate_v4
--------------------------------------
 d7a8d47e-58e3-4bd9-9340-8553ac03d144
(1 row)
mydb=# SELECT uuid_generate_v5(uuid_ns_url(), 'http://www.postgresql.org');
      uuid_generate_v5
--------------------------------------
 e1ee1ad4-cd4e-5889-962a-4f605a68d94e
(1 row)
mydb=# create table x(id uuid, value float4) distributed by (value);
```

```
CREATE TABLE
mydb=# insert into x select uuid_generate_v4(),(r*random()) from generate_series(1,100000)r;
INSERT 0 100000
mydb=# CREATE INDEX idx_x_id ON x USING btree (id);
CREATE INDEX
mydb=# ANALYZE x;
ANALYZE
mydb=# explain SELECT count(1) from x where id = '9f830fc9-a1ee-425f-844b-9c73290a91ad';
                        QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=200.44..200.45 rows=1 width=8)
   -> Gather Motion 3:1  (slice1; segments: 3)  (cost=200.37..200.42 rows=1 width=8)
      -> Aggregate  (cost=200.37..200.38 rows=1 width=8)
         -> Index Scan using idx_x_id on x  (cost=0.00..200.37 rows=1 width=0)
            Index Cond: id = '9f830fc9-a1ee-425f-844b-9c73290a91ad'::uuid
 Settings:  enable_seqscan=off; optimizer=off
 Optimizer status: legacy query optimizer
(7 rows)
```

References

- http://www.ossp.org/pkg/lib/uuid/
- https://www.postgresql.org/docs/9.4/uuid-ossp.html
- http://www.postgres.cn/docs/9.4/uuid-ossp.html

2.Use PL/Java

2.1. Use PL/Java UDF

AnalyticDB for PostgreSQL supports compiling and uploading JAR software packages written in PL/Java languages, and using these JAR packages to create user-defined functions (UDF). The PL/Java language supported in this feature is Community Edition PL/Java 1.5.0 and the JVM version is 1.8.

This document describes how to create a PL/Java UDF. For more PL/Java examples, see PL/Java Code. You can also view How to Compile.

Procedure

1. In AnalyticDB for PostgreSQL, run the following command to create a PL/Java plug-in. The command only needs to be ran once for the database.

```
create extension pljava;
```

2. Compile the UDF based on your business requirements. 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 command to compile and package the program.

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

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

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

6. In AnalyticDB for PostgreSQL, run the "Create Library" command to import the file to AnalyticDB for PostgreSQL.

   > **Note** The `Create Library` command only supports filepath and you can import one file a time. In addition, the "Create Library" command also supports byte streams to directly import files without using the OSS. For more information, see Use the Create Library Command.

   ```
   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, run the following command 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

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

AnalyticDB for PostgreSQL introduces the CREATE LIBRARY and DROP LIBRARY statements to allow you to import custom software packages. For more information about how to create PL/Java UDFs by executing the statements, see Use PL/Java UDFs.

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 installing the new one.
- `LANGUAGE [JAVA]`: the programming language to be used. Only PL/Java is supported.
- `oss_location`: the location of the package. You can specify the Object Storage Service (OSS) bucket and object names. Only one object can be specified and the specified object cannot be a compressed file. The format is as follows:

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

You can also use a temporary Security Token Service (STS) credential to access an OSS bucket. For more information, see Access OSS with a temporary access credential provided by STS. The format is as follows:

```
oss://oss_endpoint filepath=[folder/[folder/]...]/file_name id=userossid key=userosskey token=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 using OSS.
- `ownername` : the name of the user.
- `DROP LIBRARY` : deletes a library.

Examples

- Example 1: Install a JAR package 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 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 with the byte stream 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.