ALIBABA CLOUD

阿里云

云原生数据仓库 AnalyticDB PostgreSQL 版

开发进阶

文档版本：20200910

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	**单击设置> 网络> 设置网络类型。**
粗体	表示按键、菜单、页面名称等UI元素。	**在结果确认页面，单击确定。**
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.PL/Python 使用

云原生数据仓库AnalyticDB PostgreSQL支持用户使用 PL/Python 过程语言自定义函数。

限制

- 不支持在 PL/Python 中使用触发器。
- 不支持可更新的游标（比如 `UPDATE...WHERE CURRENT OF` and `DELETE...WHERE CURRENT OF` ）。
- 只支持python2，暂不支持python3。

创建或删除 PL/Python 插件

在 AnalyticDB for PostgreSQL 中，执行如下命令，创建或删除 PL/Python 插件（每个数据库只需执行一次）。因为 PL/Python 是非授信语言，只支持使用数据库的高权限（初始）账号，执行该命令。以下示例中，高权限账号为admin，需要创建 PL/Python 所在的数据库名为 testdb：

创建 PL/Python 插件

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

删除 PL/Python 插件

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

使用 PL/Python 开发 Python 函数

> 注意 由于安全原因 PL/Python 没有直接开放，创建Python函数请提交工单，经ADB PG后台技术人员确认无安全风险后，由后台创建。

示例：创建 Python 函数

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

使用函数

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

参考文档

关于 Python 的更多用法，请参见Python官网。

关于 PL/Python 的更多用法，请参见PostgreSQL文档。

1.1. 扩展插件列表

云数据库 AnalyticDB for PostgreSQL 基于 Greenplum Database 开源数据库项目开发，由阿里云深度扩展，是一种在线的分布式云数据仓库服务，由多个计算节点组成，可提供大规模并行处理（MPP）数据仓库的服务。

插件类型

云数据库 AnalyticDB for PostgreSQL 支持如下插件(EXTENSION)：

- PostGIS：支持地理信息数据。可以参见使用PostGIS
- MADlib：机器学习方面的函数库。
- fuzzystrmatch：字符串模糊匹配。
- orafunc：Oracle部分函数兼容 使用orafunc *注：ADB PG 4.3版本支持， 6.0版本不支持*
- orafce：Oracle部分函数兼容 使用orafce *注：ADB PG 6.0版本支持，4.3版本不支持*
- oss_ext：支持从 OSS 读取数据。
- hll：支持用 HyperLogLog 算法进行统计。可以参见HyperLogLog 的使用
- pljava：支持使用 PL/Java 语言编写用户自定义函数（UDF）。可以参见使用 PL/Java UDF
- pgcrypto：在表或列级别，提供如下支加密算法函数
  MD5，SHA1，SHA224/256/384/512，Blowfish，AES128/256，Raw Encryption，PGP Symmetric-Key，PGP Public Key，实现数据安全加密存储。可以参见pgcrypto
- IntArray：整数数组相关的函数、操作符和索引支持。
- roaringbitmap：采用Roaring Bitmap高效压缩算法的位图运算插件。
- postgres_fdw：跨库 访问插件。*注：ADB PG 6.0版本支持，4.3版本 不支持*
- tablefunc：表函数差价。*注：ADB PG 6.0版本支持，4.3版本 不支持*

创建插件

创建插件的方法如下所示：

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

> 注意 创建 MADlib 插件时，需要先创建 plpythonu 插件，如下所示：
>
> ```
> CREATE EXTENSION plpythonu;
> CREATE EXTENSION madlib;
> ```

删除插件

删除插件的方法如下所示：

> 注意 如果插件被其它对象依赖，需要加入 CASCADE（级联）关键字，删除所有依赖对象。

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

1.2. Laser计算引擎的使用

Laser计算引擎是阿里巴巴自研的ADBPG计算引擎，对客户透明，可以提升复杂计算的性能，经实测，1GB、100GB、1TB、10TB数据规模下，TPCH 22条SQL的测试性能是原生引擎的2倍以上。

打开/关闭Laser

Laser 计算引擎通过GUC参数laser.enable打开或者关闭，on 表示使用Laser计算引擎，select查询会通过Laser返回结果，off表示计算通过原生引擎返回结果。默认状态为关闭状态。该参数可以设置为session级别、库级别和集群级别，session结束恢复到默认状态，库级别设置以后立即生效，集群级别重启后生效。可以通过下面SQL查看、修改：

```
--- 查看Laser是否开启
show laser.enable;
--- session级别关闭Laser
set laser.enable = off;
--- session级别开启Laser
set laser.enable = on;
--- 库级别关闭Laser
alter database ${DBNAME} set laser.enable = off;
--- 库级别开启Laser
alter database ${DBNAME} set laser.enable = on;
```

> 说明 ：集群级别的设置请联系阿里云管理员，建议使用库级别或者session级别的设置。

支持的数据类型和操作

Laser支持如下数据类型：

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

Laser支持如下操作符：

- =、<、<=、>、>=、<> or !=、BETWEEN、 IS NOT NULL、 IS NULL、LIKE
- 逻辑运算符：and、or、not

Laser的限制

- 推荐使用ORCA优化器
- 只支持ADBPG 6.0 及以上版本

1.3. HyperLogLog 的使用

阿里云深度优化云数据库 AnalyticDB for PostgreSQL，除原生 Greenplum Database 功能外，还支持 HyperLogLog，为互联网广告分析及有类似预估分析计算需求的行业提供解决方案，以便于快速预估 PV、UV 等业务指标。

创建 HyperLogLog 插件

执行如下命令，创建 HyperLogLog 插件：

```
CREATE EXTENSION hll;
```

基本类型

- 执行如下命令，创建一个含有 hll 字段的表：

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

- 执行如下命令，进行 int 转 hll_hashval：

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

基本操作符

- hll 类型支持 =、!=、<>、|| 和 #。

  ```
  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);
  ```

- hll_hashval 类型支持 =、!= 和 <>。

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

基本函数

- hll_hash_boolean、hll_hash_smallint 和 hll_hash_bigint 等 hash 函数。

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

- hll_add_agg：可以将 int 转 hll 格式。

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

- hll_union：hll 并集。

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

- hll_set_defaults：设置精度。

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

- hll_print：用于 debug 信息。

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

示例

```
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_ser
ies(1,10000) t(user_id);
insert into access_date select current_date-1, hll_add_agg(hll_hash_integer(user_id)) from generate_s
eries(5000,20000) t(user_id);
insert into access_date select current_date-2, hll_add_agg(hll_hash_integer(user_id)) from generate_s
eries(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. 使用PostGIS

PostGIS是数据库PostgreSQL的一个扩展，PostGIS提供如下空间信息服务功能：空间对象、空间索引、空间操作函数和空间操作符。

> 说明
> - PostGIS遵循Open Geospatial Consortium（OGC）规范。
> - AnalyticDB for PostgreSQL支持PostGIS扩展（4.3版本对应PostGIS v2.0.3，6.0版本对应PostGIS v2.5.3）。

通用操作

1）客户端连接实例

可参考客户端连接

2）初次装载PostGIS扩展模块

创建扩展：

```
create extension postgis;
```

查看版本：

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

3）空间数据写入数据库表

首先创建带Geometry字段的表，SQL参考：

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

该SQL表示插入的空间数据不区分几何类型，几何类型包括Point / MultiPoint / Linestring / MultiLinestring / Polygon / MultiPolygon等。如果在创建表时已知Geometry类型和SRID，SQL参考：

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

Geometry类型指定Point，SRID为4326。SRID不指定默认为0。

有关SRID可参考 SRID

数据插入，SQL参考：

```
-- 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));
```

JDBC Java程序参考：

```
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)', 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）空间索引管理

- 创建空间索引

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

idx_test_geom为自定义索引名，test为表名，geom为Geometry列名。

- 查看表有哪些索引

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

- 查看索引大小

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

- 索引重构

```
reindex index idx_test_geom;
```

- 删除索引

```
drop index idx_test_geom;
```

5）典型空间查询SQL

- 矩形范围查询

```
-- 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);
```

ST_MakeBox2D算子生成一个Envelope。

- 几何缓冲范围查询

```
-- 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);
```

ST_DWithin用法参考：ST_DWithin

- 多边形相交判定（在内部或在边界上）

```
-- 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))'), geo
m);

-- 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);
```

ST_*算子对大小写不敏感，更多用法可参考 PostGIS官方资料

> 说明
>
> AnalyticDB PG 6.0不完全兼容PostGIS功能集，例如不支持 create extension postgis_topology，不推荐用Geography类型创建表（非要用，SRID默认为0或4326）。

典型案例

1）电子围栏场景

某客运监控服务运营商，通过安装在客车上的GPS定位终端收集定位数据，常见的业务有偏航报警、常去的服务区频次、驶入特定区域提醒（例如易发事故地段、积水结冰地段）等，这类业务是比较典型的电子围栏应用场景。以驶入特定区域提醒业务为例，特定区域不会频繁变更且数据量偏少，可以一次采集定期更新，考虑区域表采用复制表，SQL参考：

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

插入Polygon / MultiPolygon类型的特定区域数据后，完成统计信息收集（Analyze 表名）并构建GIST索引。判定驶入区域，可以分为两种情况：一种完全在区域内，一种是到达边界就要提醒。两种情况用到的空间算子有所区别，SQL参考：

```
-- 完全在区划内
select rid, name from ky_region
where ST_Contains(geom, ST_GeomFromText('POINT(116 39)'));

-- 考虑边界情况
select rid, name from ky_region
where ST_Intersects(geom, ST_GeomFromText('POINT(116 39)'));
```

SQL解释：输入变化的经纬度，查询区域表geom字段包含或相交与输入点的记录，如果为0条记录表示未驶入任何区域，如果为1条记录表示驶入某个区域，如果大于1条记录表示驶入多个区域（说明区域表有空间重叠的区域，需要从业务上验证空间重叠的合理性）。

2）智慧交通场景

某智慧交通场景，数据库包含线型轨迹表和其他业务表，一业务功能为查找历史轨迹表中曾经驶入过某一区域的轨迹ID，相关轨迹表结构：

```
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'),
 ......
);
```

轨迹按照天创建分区表，每天导入数据后做统计信息收集，并对分区表创建GIST空间索引。SQL参考：

```
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);
```

亿级轨迹表做空间查询的响应时间在80ms内。

3）商业客流分析

某互联网生活服务运营商，基于AnalyticDB for PostgreSQL做店铺客流量分析，数据库有两张业务表：User签到表和Shop店铺区域表，表结构参考：

```
-- 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);
```

业务表比较巧的设计是用Geohash或ZOrder编码等方式将地理空间几何降维作为分布键，而不用构建空间索引。客流统计的SQL参考：

```
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. 使用压缩位图RoaringBitmap

位图（bitmap）是一种常用的数据结构，位图为每个列所有可能的值创建一个单独的位图（0和1的系列），每个位（bit）对应一个数据行是否存在对应的值。通过位图能够快速定位一个数值是否在存在，并利用计算机位级计算的快速特性（AND，OR和ANDNOT等运算），可以显著加快位图的相关计算，非常适合大数据查询和关联计算，常用于去重、标签筛选、时间序列等计算中。

传统的位图会占用大量内存，一般需要对位图进行压缩处理。Roaring Bitmap是一种高效优秀的位图压缩算法，目前已被广泛应用在各种语言和各种大数据平台上。

Roaring Bitmap压缩算法简介

Roaring Bitmap的算法是将整数的32-bit的范围 ([0, n]) 划分为 2^16 个数据块（Chunk），每一个数据块对应整数的高16位，并使用一个容器（Container）来存放一个数值的低16位。 Roaring Bitmap将这些容器保存在一个动态数组中，作为一级索引。容器使用两种不同的结构： 数组容器（Array Container）和 位图容器（Bitmap Container）。数组容器存放稀疏的数据，位图容器存放稠密的数据。如果一个容器里面的整数数量小于4096，就用数组容器来存储值。若大于4096，就用位图容器来存储值。

采用这种存储结构，Roaring Bitmap可以快速检索一个特定的值。在做位图计算（AND，OR，XOR）时，Roaring Bitmap提供了相应的算法来高效地实现在两种容器之间的运算。使得Roaring Bitmap无论在存储和计算性能上都变得优秀。

更多关于Roaring Bitmap的介绍信息，请参考官方网站

使用Roaring Bitmap位图计算

使用Roaring Bitmap位图计算功能之前，首先需要在数据库中创建RoaringBitmap扩展插件：

```
CREATE EXTENSION roaringbitmap;
```

创建带有RoaringBitmap数据类型的表：

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

使用rb_build函数插入roaringbitmap的数据：

```
--数组位置对应的BIT值为1
INSERT INTO t1 SELECT 1,RB_BUILD(ARRAY[1,2,3,4,5,6,7,8,9,200]);
--将输入的多条记录的值对应位置的BIT值设置为1，最后聚合为一个roaringbitmap
INSERT INTO t1 SELECT 2,RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
```

Bitmap计算(OR, AND, XOR, ANDNOT)

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

Bitmap聚合计算(OR, AND, XOR, BUILD)，并生成新的roaringbitmap类型

```
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;
```

统计基数（Cardinality）,即统计roaringbitmap中包含多少个位置为1的BIT位

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

从roaringbitmap中返回位置为1的BIT下标（位置值）。

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

Bitmap函数列表

函数名	输入	输出	描述	示例
rb_build	integer[]	roaringbitmap	通过数组创建一个Bitmap。	rb_build('{1,2,3,4,5}')
rb_and	roaringbitmap，roaringbitmap	roaringbitmap	And计算。	rb_and(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_or	roaringbitmap，roaringbitmap	roaringbitmap	Or计算。	rb_or(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_xor	roaringbitmap，roaringbitmap	roaringbitmap	Xor计算。	rb_xor(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_andnot	roaringbitmap，roaringbitmap	roaringbitmap	AndNot计算。	rb_andnot(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_cardinality	roaringbitmap	integer	统计基数	rb_cardinality(rb_build('{1,2,3,4,5}'))
rb_and_cardinality	roaringbitmap，roaringbitmap	integer	And计算并返回基数。	rb_and_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))

函数名	输入	输出	描述	示例
rb_or_cardinality	roaringbitmap，roaringbitmap	integer	Or计算并返回基数。	rb_or_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_xor_cardinality	roaringbitmap，roaringbitmap	integer	Xor计算并返回基数。	rb_xor_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_andnot_cardinality	roaringbitmap，roaringbitmap	integer	AndNot计算并返回基数。	rb_andnot_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_is_empty	roaringbitmap	boolean	判断是否为空的Bitmap。	rb_is_empty(rb_build('{1,2,3,4,5}'))
rb_equals	roaringbitmap，roaringbitmap	boolean	判断两个Bitmap是否相等。	rb_equals(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_intersect	roaringbitmap，roaringbitmap	boolean	判断两个Bitmap是否相交。	rb_intersect(rb_build('{1,2,3}'),rb_build('{3,4,5}'))

函数名	输入	输出	描述	示例
rb_remove	roaringbitmap,integer	roaringbitmap	从Bitmap移除特定的Offset。	rb_remove(rb_build('{1,2,3}'),3)
rb_flip	roaringbitmap,integer,integer	roaringbitmap	翻转Bitmap中特定的Offset段。	rb_flip(rb_build('{1,2,3}'),2,3)
rb_minimum	roaringbitmap	integer	返回Bitmap中最小的Offset，如果Bitmap为空则返回-1。	rb_minimum(rb_build('{1,2,3}'))
rb_maximum	roaringbitmap	integer	返回Bitmap中最大的Offset，如果Bitmap为空则返回0。	rb_maximum(rb_build('{1,2,3}'))
rb_rank	roaringbitmap,integer	integer	返回Bitmap中小于等于指定Offset的基数。	rb_rank(rb_build('{1,2,3}'),3)
rb_iterate	roaringbitmap	setof integer	返回Offset List。	rb_iterate(rb_build('{1,2,3}'))
rb_contains	roaringbitmap, integer	bool	判断Bitmap是否包含特定的Offset	rb_contains(rb_build('{1,2,3}'),1)

函数名	输入	输出	描述	示例
rb_contains	roaringbitmap, integer, integer	bool	判断Bitmap是否包含特定的Offset段(某个范围)	rb_contains(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_contains	roaringbitmap, roaringbitmap	bool	判断Bitmap是否包含另外一个bitmap	rb_contains(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_becontained	integer, roaringbitmap	bool	判断特定的Offset是否被Bitmap包含	rb_becontained(1, rb_build('{1,2,3}'))
rb_becontained	roaringbitmap, roaringbitmap	bool	判断Bitmap是否被另外一个包含。	rb_becontained(rb_build('{1}'), rb_build('{1,2,3}'))
rb_add	roaringbitmap, integer	roaringbitmap	添加特定的Offset到Bitma	rb_add(rb_build('{1,2,3,4}'), 5)
rb_add_2	integer, roaringbitmap	roaringbitmap	Add a specific offset to roaringbitmap.	rb_add_2(5, rb_build('{1,2,3,4}'))

函数名	输入	输出	描述	示例
rb_add	roaringbitmap, integer, integer	roaringbitmap	添加特定的Offset段到Bitmap	rb_add(rb_build('{1,2,3,4}'), 6,8)
rb_remove	roaringbitmap, integer, integer	roaringbitmap	从Bitmap移除特定的Offset	rb_remove(rb_build('{1,2,3,4,6,7,8}'), 6,8)
rb_jaccard_index	roaringbitmap, roaringbitmap	float8	计算两个Bitmap之间的jaccard相似系数	rb_jaccard_index(rb_build('{1,2,3,4}'), rb_build('{1,2}'))
rb_to_array	roaringbitmap	integer[]	Bitmap转数组	rb_to_array(rb_build('{1,2,3,4}'))
rb_iterate_decrement	roaringbitmap	integer[]	返回Offset List（从大到小）	rb_iterate_decrement(rb_build('{1,2,3,4}'))

Bitmap聚合函数列表

聚合函数名	输入	输出	描述	示例
rb_build_agg	integer	roaringbitmap	将Offset聚合成bitmap。	rb_build_agg(1)

聚合函数名	输入	输出	描述	示例
rb_or_agg	roaringbitmap	roaringbitmap	Or 聚合计算。	rb_or_agg(rb_build('{1,2,3}'))
rb_and_agg	roaringbitmap	roaringbitmap	And 聚合计算。	rb_and_agg(rb_build('{1,2,3}'))
rb_xor_agg	roaringbitmap	roaringbitmap	Xor 聚合计算。	rb_xor_agg(rb_build('{1,2,3}'))
rb_or_cardinality_agg	roaringbitmap	integer	Or 聚合计算并返回其基数。	rb_or_cardinality_agg(rb_build('{1,2,3}'))
rb_and_cardinality_agg	roaringbitmap	integer	And 聚合计算并返回其基数。	rb_and_cardinality_agg(rb_build('{1,2,3}'))
rb_xor_cardinality_agg	roaringbitmap	integer	Xor 聚合计算并返回其基数。	rb_xor_cardinality_agg(rb_build('{1,2,3}'))

操作符

操作符	left	right	output	描述	示例
&	roaringbitmap	roaringbitmap	roaringbitmap	两个Bitmap And 操作	rb_build('{1,2,3}') & rb_build('{1,2,4}')
|	roaringbitmap	roaringbitmap	roaringbitmap	两个Bitmap Or 操作	rb_build('{1,2}') | rb_build('{1,3}')
#	roaringbitmap	roaringbitmap	roaringbitmap	两个Bitmap Xor 操作	rb_build('{1,2}') # rb_build('{1,3}')
~	roaringbitmap	roaringbitmap	roaringbitmap	两个Bitmap Andnot 操作	rb_build('{2,3}') ~ rb_build('{2,4}')
+	roraingbitmap	integer	roaringbitmap	向Bitmap中添加特定的Offset	rb_build('{2,3}') + 1
-	roraingbitmap	integer	roaringbitmap	从Bitmap移除特定的Offset	rb_build('{1,2,3}') - 1

操作符	left	right	output	描述	示例
=	roaringbitmap	roaringbitmap	boolean	判断两个Bitmap是否相等	rb_build('{2,3}') = rb_build('{2,3}')
<>	roaringbitmap	roaringbitmap	boolean	判断两个Bitmap是否不相等	rb_build('{2,3}') <> rb_build('{1,2,3}')
&&	roaringbitmap	roaringbitmap	boolean	判断两个Bitmap是否相交	rb_build('{2,3}') && rb_build('{3,4}')
@>	roaringbitmap	roaringbitmap	boolean	判断Bitmap是否包含另外一个	rb_build('{2,3}') @> rb_build('{2}')
@>	roaringbitmap	integer	boolean	判断Bitmap是否包含特定的Offset	rb_build('{2,3}') @> 2
<@	roaringbitmap	roaringbitmap	boolean	判断Bitmap是否被另外一个包含	rb_build('{2,3}') <@ rb_build('{1,2,3}')

操作符	left	right	output	描述	示例
<@	integer	roaringbitmap	boolean	判断特定的Offset是否被Bitmap包含	rb_build('{2,3}') <@ rb_build('{3}')

1.6. 使用UUID-OSSP

本文介绍如何使用UUID-OSSP。

安装UUID-OSSP

安装UUID-OSSP代码如下所示：

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

> 说明 只有superuser或rds_superuser权限用户可以安装该插件。

插件说明

`"uuid-ossp"` extension创建后，会自动创建 `uuid` 数据类型且支持btree索引。

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

UUID数据类型用来存储RFC 4122、ISO/IEF 9834-8:2005以及相关标准定义的通用唯一标识符（UUID）。（部分系统认为这个数据类型为全球唯一标识符，或GUID。） 这个标识符是一个由算法产生的128位标识符，使它不可能在已知使用相同算法的模块中和其他方式产生的标识符相同。 因此，对分布式系统而言，这种标识符比序列能更好的提供唯一性保证，因为序列只能在单一数据库中保证唯一。

UUID被写成一个小写十六进制数字的序列，由分字符分成几组， 特别是一组8位数字+3组4位数字+一组12位数字，总共32个数字代表128位， 标准的UUID示例如下：

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

同样支持以其他方式输入：大写数字， 由花括号包围的标准格式，省略部分或所有连字符，在任意一组四位数字之后加一个连字符。如：

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

> 说明 当前版本不支持uuid类型字段作为分布键。

插件函数说明

- UUID生成函数

函数	描述
uuid_generate_v1()	此函数会生成v1版本的UUID。算法使用了计算机的MAC地址和时间戳。 说明 该UUID会泄露计算机标识和生成时间，不适合对安全性要求较高的应用。
uuid_generate_v1mc()	此函数会生成一个v1版本的UUID。和 uuid_generate_v1() 的区别在于 uuid_generate_v1mc() 使用的是一个随机多播MAC地址， uuid_generate_v1() 使用的是计算机的真实的MAC地址。
uuid_generate_v3(namespace uuid, name text)	此函数会生成一个v3版本的UUID。这个函数会使用指定输入名称 name 在指定的命名空间 namespace 中生成。 ○ 指定的命名空间应该是调用下表中的函数 uuid_ns_*() 返回的常量。 ○ 参数 name 是一个指定命名空间 namespace 中的标识符。 例如： `SELECT uuid_generate_v3(uuid_ns_url(), 'http://www.postgresql.org');` name 会使用MD5算法进行哈希，从产生的UUID中不能反向获得明文。利用这个方法生成的UUID不需要随机算法且不依赖任何运行相关的环境因素，生成过程是可重复的。
uuid_generate_v4()	此函数会生成一个v4版本的UUID。算法完全依靠随机数。
uuid_generate_v5(namespace uuid, name text)	此函数会生成一个v5版本的UUID。工作过程类似于v3版本的 UUID，但是v5版本使用的是SHA-1的哈希算法，因为SHA-1算法被认为比 MD5算法更安全，所有应该尽量使用v5版本而不是v3版本。

- 返回 UUID 常量的函数

函数	描述
uuid_nil()	代表nil的UUID常量，此处不应该看作一个真正的UUID。
uuid_ns_dns()	代表DNS命令空间的UUID常量。
uuid_ns_url()	代表URL命名空间的UUID常量。
uuid_ns_oid()	代表ISO对象标识符（OID）命名空间的UUID常量。 说明 此处OID是ASN.1的OID，和PostgreSQL用的OID没有关系。
uuid_ns_x500()	代表X.500识别名字（DN）命名空间的UUID常量。

示例

```
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)
```

参考文档

- 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.PL/Java 使用

2.1. 使用 PL/Java UDF

AnalyticDB for PostgreSQL 支持用户使用 PL/Java 语言，编写并上传 jar 软件包，并利用这些 jar 包创建用户自定义函数（UDF）。该功能支持的 PL/Java 语言版本为社区版 PL/Java 1.5.0，使用的 JVM 版本为 1.8。

本文介绍了创建 PL/Java UDF 的示例步骤。更多的 PL/Java 样例，请参见 PL/Java代码（查看 编译方法）。

操作步骤

1. 在 AnalyticDB for PostgreSQL 中，执行如下命令，创建 PL/Java 插件（每个数据库只需执行一次）。

```
create extension pljava;
```

2. 根据业务需要，编写自定义函数。例如，使用如下代码编写 `Test.java` 文件：

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

3. 编写 `manifest.txt` 文件。

```
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. 执行如下命令，将程序编译打包。

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

5. 将步骤 4 生成的 `analytics.jar` 文件，通过 OSS 控制台命令上传到 OSS。

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

6. 在 AnalyticDB for PostgreSQL 中，执行 Create Library 命令，将文件导入到 AnalyticDB for PostgreSQL 中。

   > 注意 Create Library 只支持 filepath，一次导入一个文件。另外，Create Library 还支持字节流形式，可以不通过 OSS 直接导入，详情请参见Create Library命令的使用 。

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

7. 在 AnalyticDB for PostgreSQL 中，执行如下命令，创建和使用相应 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. 使用 Create Library 命令

为支持用户导入自定义软件包，AnalyticDB for PostgreSQL 引入了 Create Library/Drop Library 命令。使用此命令创建 PL/Java 的 UDF 的示例，请参见 PL/Java UDF的使用。

本文介绍了 Create/Drop Library 命令的使用方法。

语法

```
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
```

参数说明：

- `library_name`：要安装的库的名称。若已安装的库与要安装的库的名称相同，必须先删除现有的库，然后再安装新库。
- `LANGUAGE [JAVA]`：要使用的语言。目前仅支持 PL/Java。
- `oss_location`：包文件的位置。您可以指定 OSS 存储桶和对象名称，仅可以指定一个文件，且不能为压缩文件。其格式为：

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

  您也可以使用阿里云 STS（Security Token Service）临时凭证来访问 OSS 存储桶，参见STS临时授权访问OSS。使用格式为：

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

- `file_content_hex`：文件内容，字节流为 16 进制。例如，`73656c6563742031`表示”select 1”的 16 进制字节流。借助这个语法，可以直接导入包文件，不必通过 OSS。
- `ownername`：指定用户。
- `DROP LIBRARY`：删除一个库。

示例

- 示例 1：安装名为 `analytics.jar` 的 jar 包。

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

- 示例 2：使用STS临时凭证安装名为 `analytics.jar` 的 jar 包。

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

- 示例 3：直接导入文件内容，字节流为 16 进制。

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

- 示例 4：删除一个库。

```
drop library example;
```

- 示例 5：查看已经安装的库。

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

3.PL/Python 使用

云原生数据仓库AnalyticDB PostgreSQL支持用户使用 PL/Python 过程语言自定义函数。

限制

- 不支持在 PL/Python 中使用触发器。
- 不支持可更新的游标（比如 `UPDATE...WHERE CURRENT OF` and `DELETE...WHERE CURRENT OF` ）。
- 只支持python2，暂不支持python3。

创建或删除 PL/Python 插件

在 AnalyticDB for PostgreSQL 中，执行如下命令，创建或删除 PL/Python 插件（每个数据库只需执行一次）。因为 PL/Python 是非授信语言，只支持使用数据库的高权限（初始）账号，执行该命令。以下示例中，高权限账号为admin，需要创建 PL/Python 所在的数据库名为 testdb：

创建 PL/Python 插件

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

删除 PL/Python 插件

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

使用 PL/Python 开发 Python 函数

> 注意 由于安全原因 PL/Python 没有直接开放，创建Python函数请提交工单，经ADB PG后台技术人员确认无安全风险后，由后台创建。

示例：创建 Python 函数

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

使用函数

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

参考文档

关于 Python 的更多用法，请参见Python官网。

关于 PL/Python 的更多用法，请参见PostgreSQL文档。