ALIBABA CLOUD

阿里云

云原生数据仓库 AnalyticDB PostgreSQL 版

开发进阶

文档版本：20220121

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.高级扩展插件使用 05
1.1. 扩展插件列表 05
1.2. Laser计算引擎的使用 06
1.3. HyperLogLog 的使用 07
1.4. 使用PostGIS 08
1.5. 使用压缩位图RoaringBitmap 13
1.6. 使用UUID-OSSP 21
1.7. 使用Zhparser支持中文分词 24
1.8. 使用Hint 26
2.PL/Java 使用 42
2.1. 使用PL/Java UDF 42
2.2. 使用 Create Library 命令 43
3.PL/Python 使用 45
4.AnalyticDB PostgreSQL版错误码 46
5.数据写入方式概述 60

1.高级扩展插件使用

1.1. 扩展插件列表

云原生数据仓库AnalyticDB PostgreSQL版基于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

云原生数据仓库AnalyticDB PostgreSQL版支持如下插件（EXTENSION）：

- PostGIS：支持地理信息数据。请参见使用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 PG4.3版本支持， 6.0版本不支持。*
- orafce：Oracle部分函数兼容 使用orafce。 *注：ADB PG6.0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：表函数插件。*注：ADB PG 6.0版本支持，4.3版本不支持。*
- zhparser：中文分词插件。*注：ADB PG 6.0版本支持，4.3版本不支持。*
- pg_hint_plan：支持通过Hint干预和调优查询SQL，加速查询速度。具体信息，请参见使用Hint。

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计算引擎，对用户透明，可以提升复杂计算的性能，经实测，在1 GB、100 GB、1 TB、10 TB数据规模下，性能是原生Greenplum的2倍以上。

功能限制

- 建议使用ORCA优化器。
- 仅支持 AnalyticDB PostgreSQL 6.0版及以上版本。

开启或关闭Laser

Laser计算引擎可以通过GUC参数**laser.enable**开启或关闭，on表示开启；off表示关闭。该参数可以设置Session级别、库级别和集群级别，Session结束后恢复到默认状态，库级别设置后立即生效，集群级别设置后重启生效，以下内容将为您介绍如何查看或修改Laser计算引擎状态：

- 查看Laser计算引擎的状态，示例如下：

  ```
  SHOW laser.enable;
  ```

  返回示例如下：

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

- 开启Session级别Laser，示例如下：

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

- 关闭Session级别Laser，示例如下：

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

- 开启库级别Laser，示例如下：

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

- 关闭库级别Laser，示例如下：

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

说明

- 目前不支持修改集群级别的Laser，建议您使用库级别或Session级别的设置。如果需要修改集群级别Laser，请提交工单联系技术支持进行修改。
- 内核版本为6.3.4.0以前版本，Laser计算引擎状态默认为关闭；6.3.4.0及以后版本，Laser计算引擎状态默认为开启。如何查看和升级内核小版本，请参见查看内核小版本和版本升级。

支持的数据类型和操作

Laser支持的数据类型如下：

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

Laser支持的操作符如下：

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

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 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. 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)', 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）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 3
9))'), 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 3
9))', 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  29.2059
51,118.732296  29.205644,
                  118.73226  29.205469,118.732350  29.20470,118.731708  29.203399,118.73170
1  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 bitm
ap FROM 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

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}'))`
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)`

函数名	输入	输出	描述	示例
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)`
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}'))`

函数名	输入	输出	描述	示例
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}'))`
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}')`

操作符	left	right	output	描述	示例
|	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`
=	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}')`

操作符	left	right	output	描述	示例
@>	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}')`
<@	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。

UUID-OSSP简介

UUID数据类型用于储存全局唯一标识符UUID。对分布式系统来说，UUID比序列更能保证唯一性。

UUID由32个16进制数字组成，标准格式为一组8位字符+一组4位字符+一组4位字符+一组4位字符+12组字符，每组字符由连字符连接，示例如下：

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

UUID还支持以大写字母、花括号包围标准格式、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的方式输入，例如：

```
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。

安装UUID-OSSP

安装UUID-OSSP代码如下所示：

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

> **说明** 需要rds_superuser权限的账号安装该插件。

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(), 'example.com');` `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，所以应该尽量使用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。

函数	描述
`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

- 生成一个V1版本的UUID。

```
SELECT uuid_generate_v1();
```

返回信息如下：

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

- 生成一个V3版本的UUID。

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

返回信息如下：

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

- 生成一个V4版本的UUID。

```
SELECT uuid_generate_v4();
```

返回信息如下：

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

- 生成一个V5版本的UUID

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

返回信息如下：

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

参考文档

- OSSP官网
- PostgreSQL官网

1.7. 使用Zhparser支持中文分词

AnalyticDB PostgreSQL版数据库安装Zhparser插件后可以实现全文检索时的中文分词。

> **说明** 全文检索功能仅支持AnalyticDB PostgreSQL 6.0版。

全文检索概述

PostgreSQL默认按照空格及各种标点符号来分词，不支持中文分词。AnalyticDB PostgreSQL版通过集成Zhparser扩展来支持中文分词。

一般情况下，全文检索可以采用如下两种方法：

- 搜索表：

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

- 创建GIN索引：

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

配置Zhparser

1. 安装Zhparser插件。

```
CREATE EXTENSION zhparser;
```

> **说明** 需要使用rds_superuser权限账号安装。

2. 配置中文解析器，取名为zh_cn。

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

配置完成后，您可以通过 `\dF` 和 `\dFp` 命令查看配置。目前暂不支持用户自定义词典。

3. 查看分词策略。
   - 查看Zhparser的词典配置：

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

返回信息如下：

```
          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)
```

- 查看zh_cn的结构（Configuration）：

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

4. 添加或删除分词策略：
   - 添加分词策略：

     添加名词（n）、动词（v）、形容词（a）、成语（i）、叹词（e）和习用语（l）六种分词策略：

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

   - 删除分词策略：

     删除名词（n）、动词（v）、形容词（a）、成语（i）、叹词（e）和习用语（l）六种分词策略：

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

5. 通过以下下两个函数测试全文检索中文分词功能：
   - to_tsvector:

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

     返回信息如下：

```
                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');
```

返回信息如下：

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

相关文档

- 全文检索总体介绍，请参见全文检索。
- 全文检索函数和操作符，请参见文本检索函数和操作符。

1.8. 使用Hint

AnalyticDB PostgreSQL版提供了pg_hint_plan插件，插件可以通过Hint（注释语句提示）干预和调优执行计划，提升SQL的执行能力。该插件还支持为SQL语句生成SQL语句模板并注册对应Hint规则，后续执行同SQL语句模板的SQL语句时会自动生成Hint干预调优后的执行计划。

版本限制

目前pg_hint_plan插件仅内核小版本为V6.3.7.0及以上的AnalyticDB PostgreSQL版实例以上版本支持，如何查看和升级内核小版本，请参见查看内核小版本和版本升级。

功能介绍

AnalyticDB PostgreSQL版优化器，基于统计数据而非固定的规则，评估SQL语句各种可行的执行算子的代价并选择代价最低的组合执行。虽然优化器会尽可能选择最好的执行计划，但是由于数据潜在的内连关系，最终给出的执行计划未必适用于当前场景。

pg_hint_plan插件可以用Hint来强制干预和优化SQL语句的执行计划，并将优化后的SQL语句模板和Hint进行注册，后续遇到相同SQL语句模板（常数参数数值不同，其它值都相同）的SQL语句将自动生成调优后的执行计划，从而提高执行效率。

支持的Hint

类别	格式	说明

类别	格式	说明
设置语句级GUC参数	`Set(GUC-param value)`	设置优化器阶段的GUC参数。具体信息，请参见GUC参数。 目前GUC参数仅在优化器阶段生效，暂时在其它阶段（例如Rewrite和Execute等阶段）不生效。 • 如需语句级别关闭ORCA优化器，建议与 `SET(optimizer off)` 语句共同使用。 • 如需语句级别使用ORCA优化器，建议与 `SET(<ORCA相关干预参数> <参数对应值>)` 语句共同使用。
扫描方法提示	`SeqScan(table)`	强制序列扫描。
	`TidScan(table)`	强制TID扫描。
	`IndexScan(table[ index...])`	强制索引扫描，且允许指定一个索引。
	`IndexOnlyScan(table[ index...])`	强制使用仅索引扫描，且允许指定一个索引。
	`BitmapScan(table[ index...])`	强制使用位图索引（Bitmap）扫描。
	`NoSeqScan(table)`	禁用序列扫描。
	`NoTidScan(table)`	禁用TID扫描。
	`NoIndexScan(table)`	禁用索引扫描。
	`NoIndexOnlyScan(table)`	禁用仅索引扫描。
	`NoBitmapScan(table)`	禁用位图索引扫描。
联接方法提示 说明 需要配合**联接顺序提示**共同使用。	`NestLoop(table table[ table...])`	强制使用嵌套循环联接。
	`HashJoin(table table[ table...])`	强制使用散列联接。
	`MergeJoin(table table[ table...])`	强制使用合并联接。
	`NoNestLoop(table table[ table...])`	禁用嵌套循环联接。
	`NoHashJoin(table table[ table...])`	禁用散列联接。

类别	格式	说明
	`NoMergeJoin(table table[ table...])`	禁用合并联接。
联接顺序提示	`Leading(table table[ table...])`	强制定义联接的顺序。
	`Leading(<join pair>)`	强制定义联接的顺序和方向。
行号纠正提示	`Rows(table table[ table...] correction)`	纠正由指定表组成的联接结果的行号。 可用的校正方法为绝对值 `#<n>` 、加法 `+<n>` 、减法 `-<n>` 和乘法 `* <n>` 。 `<n>` 是strtod函数可以读取的数字。 **说明** ROWS修改的是总行数，返回的查询计划中显示的是每个节点平均行数（总行数/节点数量）。

说明

- 当前GUC参数之外的其它Hint仅对查询优化器（Postgres query optimizer）生效，对ORCA优化器不生效。
- 当前暂时不支持并行程度相关的干预能力。

示例如下：

- 设置语句级GUC参数

  优化器阶段的GUC参数配置，对ORCA优化器和查询优化器均生效。

  - 关闭ORCA优化器：

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

    关闭后将不会使用ORCA优化器。

  - 启用ORCA优化器：

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

    启用后将默认使用ORCA优化器。数据库在大多数情况都会使用ORCA优化器，只有部分情况（例如单表查询、过多分区表等）不会使用ORCA优化器。

  - 强制启用ORCA优化器：

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

    强制启用后所有情况均使用ORCA优化器。数据库只有在ORCA优化器无法创建计划时不使用ORCA优化器。

- 强制启用ORCA优化器并关闭ORCA优化器的HashJoin能力：

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

- 扫描方法提示

  以下内容仅适用查询优化器，使用前请执行以下命令关闭ORCA优化器：

```
SET optimizer to off;
```

  - 强制t1表进行索引扫描：

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

  - 禁止对t1表进行索引扫描：

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

  - 强制t1表使用t1_val进行位图索引扫描：

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

  - 强制t1表进行仅索引扫描（IndexOnlyScan）：

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

> **说明** 仅索引扫描只有仅扫描索引列时才能使用。

  - 强制t1表进行TID扫描：

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

> **说明** TID扫描只有在表中有TID条件时才能使用。

- 联接方法提示和联接顺序提示

  以下内容仅适用查询优化器，使用前请执行以下命令关闭ORCA优化器：

```
SET optimizer to off;
```

  - 联接时t1为左表，且联接类型为MergeJoin：

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

  - 联接时t1为左表，且联接类型为NestLoopJoin：

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

  - 联接时t1为左表，且联接时禁止HashJoin：

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

- 指定t2与t3先进行HashJoin，随后与t1进行NestLoopJoin：

```
/*+ 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;
```

- 行号纠正提示

  以下内容仅适用查询优化器，使用前请执行以下命令关闭ORCA优化器：

```
SET optimizer to off;
```

  - 将t1与t2联接后的总行数扩大100倍：

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

  - 将t1与t2联接后的总行数缩小100倍：

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

  - 将t1与t2联接后的总行数增加100行：

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

  - 将t1与t2联接后的总行数减少100行：

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

  - 将t1与t2联接后的总行数修正为100：

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

GUC参数

参数名称	默认值	说明
pg_hint_plan.enable_hint	on	是否使用Hint干预计划。取值说明： • on：开启Hint干预计划。 • off：关闭Hint干预计划。
pg_hint_plan.enable_hint_table	off	是否使用Hint注册功能。取值说明： • on：开启Hint注册功能。 • off：关闭Hint注册功能。
pg_hint_plan.jumble_mode	off	定义是否使用OID来标识SQL参数化语句中的对象，例如表、函数、操作符等。取值说明： • on：使用OID区别各个对象，在对象删除用同名称重建后会认为是不同对象。 • off：使用Schema与对象名称区别各个对象，只要是同Schema下同一个名称的该类型对象会识别为相同。 **说明** 该参数不建议经常切换，切换后将无法识别切换前注册的规则。

参数名称	默认值	说明
pg_hint_plan.parse_messages	info	控制Parse Hint阶段的错误信息的日志等级。取值如下： error、warning、notice、info、log、debug[1-5]。
pg_hint_plan.message_level	log	控制Hint其它阶段的错误信息的日志等级。取值如下： error、warning、notice、info、log、debug[1-5]。

启用Hint功能

1. 载入和安装插件。
   - 临时使用Hint功能，请参考以下步骤载入和安装插件：
     a. 连接数据库，具体操作，请参见客户端连接。
     b. 执行以下命令安装插件：

     ```
     CREATE EXTENSION pg_hint_plan;
     ```

     说明　该方式仅对当前会话生效。
   - 持续使用Hint功能，请参考以下步骤载入和安装插件。
     a. 请提交工单联系技术支持载入pg_hint_plan插件。
     b. 执行以下命令安装插件：

     ```
     CREATE EXTENSION pg_hint_plan;
     ```

   说明　仅安装插件的库可以使用Hint功能。
2. （可选）启用Hint注册能力。
   - 临时使用Hint注册能力，可以使用以下语句在当前会话中启用该功能：

     ```
     SET pg_hint_plan.enable_hint_table to on;
     ```
   - 持续使用Hint注册能力，请提交工单联系技术启用Hint注册能力。

     说明　技术支持启用Hint注册能力后，您需要重新连接数据库Hint注册能力才会生效。
3. （可选）注册Hint。可以通过函数修改hint_plan.hints表，完成Hint注册。具体操作，请参见注册Hint。

   注册后，后续执行相同SQL语句模板的SQL语句时，会自动生成Hint干预调优后的执行计划。

注册Hint

当希望相同SQL模板的SQL语句自动应用Hint或者当出现SQL语句不方便添加Hint时，可以将Hint注册信息添加至系统表hint_plan.hints中。注册后，相同SQL语句模板的SQL语句会自动应用Hint。

hint_plan.hints表结构如下：

列名	类型	内容
id	integer	注册Hint规则的标号，默认递增。
norm_query_string	text	SQL语句模板，即去除参数（Param）和常数（Const）的SQL语句。
application_name	text	注册该Hint规则的应用标识字符串，用于多个应用间隔离规则，默认为 `''` ，表示不隔离。下文中的函数示例中该参数均为 `''` 。 application_name列拥有唯一键约束。
hints	text	为SQL语句模板注册的Hint。 hints列拥有唯一键约束。
query_hash	bigint	SQL语句模板参数化后的Hash值，为标准化SQL的唯一标识。 query_hash列拥有唯一键约束。
enable	boolean	控制Hint规则是否可用。同一SQL语句模板只能使用一个Hint规则。
prepare_param_strings	text	注册的查询SQL语句为Prepare语句时，记录其参数。

说明 您可以直接查询hint_plan.hints表，但不建议直接修改该表，如需修改建议使用对应函数进行修改。

以下内容将介绍Hint注册函数：

- SQL语句参数化函数

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

参数	说明
`query`	包含Hint的SQL语句。
`application_name`	注册该Hint规则的应用标识字符串，此处默认留空（ `''` ）。

该函数用于获取带有Hint的SQL语句的各种参数信息，返回信息如下：

参数	说明
`query_hash`	SQL语句模板参数化后的Hash值，为标准化SQL的唯一标识。
`norm_query_string`	SQL语句模板。
`comment_hints`	语句的注释。

参数	说明
`first_matched_hint_in_table`	在hint_plan.hints表中与SQL语句模板匹配的注释。
`prepare_param_strings`	SQL语句中提取出的参数。

示例如下：

```
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;');
```

返回示例如下：

```
-[ 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       | {}
```

- Hint注册函数

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

参数	说明
`query`	包含Hint的SQL语句。
`application_name`	注册该Hint规则的应用标识字符串，此处默认留空（`''`）。

使用该函数可以为同一个SQL语句模板注册不同的Hint规则。当您重复插入SQL语句模板、Hint、应用标识字符串相同的Hint规则时，hint_plan.hints表中不会存在多组相同的Hint规则，只会将对应的Hint规则变为true，其它Hint规则变为false。

示例如下：

```
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;');
```

返回示例如下：

```
                                                                    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)
```

- Hint修改函数

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

参数	说明
`query`	包含Hint的SQL语句。
`application_name`	注册该Hint规则的应用标识字符串，此处默认留空（`''`）。

如果SQL语句对应的参数模板有可用的Hint，则将hint_plan.hints表中的原Hint替换为`query`自带的Hint；如果没有可用的Hint，则新注册的Hint规则。

示例如下：

i. 查询hint_plan.hints表中现有的Hint规则：

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

返回信息如下：

```
 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. 执行Hint修改函数：

```
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;');
```

返回信息如下：

```
                                                                    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. 查询修改Hint规则后的hint_plan.hints表：

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

可以看到同SQL语句模板的Hint从`Nestloop(t1 t2) Leading((t1 t2))`变成了`HashJoin(t1 t2) Leading((t1 t2))`，返回信息如下：

```
 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)
```

- Hint删除函数
  - 删除对应ID的Hint规则：

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

  - 删除对应SQL语句，Hint，应用标识字符串的Hint规则：

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

  - 删除所有对应SQL语句，应用标识字符串的Hint规则：

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

参数	说明
`id`	hint_plan.hints表中的标号（ID）。
`query`	SQL语句，可以不包含Hint。
`hint`	Hint。
`application_name`	注册该Hint规则的应用标识字符串，此处默认留空（`''`）。

示例如下：

查询原hint_plan.hints表信息：

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

返回信息如下：

```
 id |                              norm_query_string                              | applica
tion_name |                      hints                      |       query_hash       | enab
le | 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)
```

- 根据ID删除对应的Hint规则：

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

返回信息如下：

```
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)
```

查询删除后的hint_plan.hints表：

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

返回信息如下：

```
 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)
```

- 根据SQL语句、Hint和应用标识字符串删除对应的Hint规则：

```
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))');
```

返回信息如下：

```
                                                                    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)
```

查询删除后的hint_plan.hints表：

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

返回信息如下：

```
 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)
```

- 根据SQL语句、应用标识字符串删除对应的Hint规则：

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

返回信息如下：

```
                                                        delete_all_hint_table
-----------------------------------------------------------------------------------------
-------------------------------------------
 (3,"select * from t1 join t2 on t1.val = t2.val;","","set(optimizer on) set(rds_optimi
zer_options 0) ",-2169095602568752481,f,{})
 (4,"select * from t1 join t2 on t1.val = t2.val;","","set(optimizer off) ",-2169095602
568752481,t,{})
(2 rows)
```

查询删除后的hint_plan.hints表：

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

返回信息如下：

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

- Hint使能函数
  - 启用对应ID的Hint规则，启用后同SQL语句模板的其它Hint规则将不可用：

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

  - 启用对应SQL语句，Hint，应用标识字符串的Hint规则，启用后同SQL语句模板的其它Hint规则将不可用：

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

  - 禁止对应ID的Hint规则：

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

  - 禁止对应SQL语句，Hint，应用标识字符串的Hint规则：

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

  - 禁止对应SQL语句，应用标识字符串的Hint规则：

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

参数	说明
`id`	hint_plan.hints表中的标号（ID）。
`query`	SQL语句，可以不包含Hint。

参数	说明
`hint`	Hint规则。
`application_name`	注册该Hint规则的应用标识字符串，此处默认留空（`''`）。

示例如下：

查询原hint_plan.hints表信息：

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

返回信息如下：

```
 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)
```

- 禁止对应ID的Hint规则：

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

  返回信息如下：

  ```
                                                          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)
  ```

  查询变更状态后的hint_plan.hints表：

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

  返回信息如下：

  ```
   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)
  ```

- 启用对应ID的Hint规则：

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

返回信息如下：

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

查询变更状态后的hint_plan.hints表：

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

返回信息如下：

```
 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)
```

- 根据SQL语句、应用标识字符串启用对应的Hint规则：

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

返回信息如下：

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

查询变更状态后的hint_plan.hints表：

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

返回信息如下：

```
 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)
```

卸载pg_hint_plan插件

如果您不需要使用Hint功能，可以通过以下语句卸载插件：

```
DROP EXTENSION pg_hint_plan;
```

相关文档

通过Hint干预执行计划

2.PL/Java 使用

2.1. 使用PL/Java UDF

AnalyticDB PostgreSQL版支持用户使用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 PostgreSQL版中，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 PostgreSQL版中，执行Create Library命令，将文件导入到AnalyticDB PostgreSQL版中。

> **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 PostgreSQL版中，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 b
  ucket=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，参见使用STS临时访问凭证访问OSS。使用格式为：

  ```
  oss://oss_endpoint filepath=[folder/[folder/]...]/file_name id=userossid key=userosskey t
  oken=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=an
  alytics.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=an
  alytics.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版支持用户使用 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。

4.AnalyticDB PostgreSQL版错误码

AnalyticDB PostgreSQL版服务器发出的所有消息都赋予了五个字符的错误代码， 这些代码遵循SQL的"SQLSTATE" 代码的习惯。本文中列出了PostgreSQL 8.1定义的所有错误代码。

错误码	含义
00 类：操作成功	
00000	成功完成（SUCCESSFUL COMPLETION）
01 类：警告	
01000	警告（WARNING）
0100C	返回了动态结果（DYNAMIC RESULT SETS RETURNED）
01008	警告，隐含补齐了零比特位（IMPLICIT ZERO BIT PADDING）
01003	在集合函数里消除了空值（NULL VALUE ELIMINATED IN SET FUNCTION）
01007	无权限（PRIVILEGE NOT GRANTED）
01006	没有撤销权限（PRIVILEGE NOT REVOKED）
01004	字串数据在右端截断（STRING DATA RIGHT TRUNCATION）
01P01	废弃的特性（DEPRECATED FEATURE）
02 类：没有数据 — 按照 SQL 标准的要求，这也是警告类	
02000	没有数据（NO DATA）
02001	返回了没有附加动态结果集（NO ADDITIONAL DYNAMIC RESULT SETS RETURNED）
03 类：SQL语句尚未结束	

错误码	含义
03000	SQL语句尚未结束（SQL STATEMENT NOT YET COMPLETE）
08 类：连接异常	
08000	连接异常（CONNECTION EXCEPTION）
08003	连接不存在（CONNECTION DOES NOT EXIST）
08006	连接失败（CONNECTION FAILURE）
08001	SQL客户端不能建立SQL连接（SQLCLIENT UNABLE TO ESTABLISH SQLCONNECTION）
08004	SQL服务器拒绝建立SQL连接（SQLSERVER REJECTED ESTABLISHMENT OF SQLCONNECTION）
08007	未知的事务解析（TRANSACTION RESOLUTION UNKNOWN）
08P01	违反协议（PROTOCOL VIOLATION）
09 类：触发器动作异常	
09000	触发的动作异常（TRIGGERED ACTION EXCEPTION）
0A 类：不支持特性	
0A000	不支持此特性（FEATURE NOT SUPPORTED）
0B 类：非法事务初始化	
0B000	非法事务初始化（INVALID TRANSACTION INITIATION）
0F 类：指示器异常	

错误码	含义
0F000	指示器异常（LOCATOR EXCEPTION）
0F001	非法的定位器声明（INVALID LOCATOR SPECIFICATION）
0L 类：非法赋权	
0L000	非法赋权（INVALID GRANTOR）
0LP01	非法赋权操作（INVALID GRANT OPERATION）
0P 类：非法角色声明	
0P000	非法角色声明（INVALID ROLE SPECIFICATION）
21 类：势违反	
21000	势违反（CARDINALITY VIOLATION）
22 类：数据异常	
22000	数据异常（DATA EXCEPTION）
2202E	数组下标错误（ARRAY SUBSCRIPT ERROR）
22021	字符不在准备好的范围内（CHARACTER NOT IN REPERTOIRE）
22008	日期时间字段溢出（DATETIME FIELD OVERFLOW）
22012	被零除（DIVISION BY ZERO）
22005	赋值中出错（ERROR IN ASSIGNMENT）
2200B	逃逸字符冲突（ESCAPE CHARACTER CONFLICT）

错误码	含义
22022	指示器溢出（INDICATOR OVERFLOW）
22015	内部字段溢出（INTERVAL FIELD OVERFLOW）
2201E	对数运算的非法参数（INVALID ARGUMENT FOR LOGARITHM）
2201F	指数函数的非法参数（INVALID ARGUMENT FOR POWER FUNCTION）
2201G	宽桶函数的非法参数（INVALID ARGUMENT FOR WIDTH BUCKET FUNCTION）
22018	类型转换时非法的字符值（INVALID CHARACTER VALUE FOR CAST）
22007	非法日期时间格式（INVALID DATETIME FORMAT）
22019	非法的逃逸字符（INVALID ESCAPE CHARACTER）
2200D	非法的逃逸字节（INVALID ESCAPE OCTET）
22025	非法逃逸序列（INVALID ESCAPE SEQUENCE）
22P06	非标准使用逃逸字符（NONSTANDARD USE OF ESCAPE CHARACTER）
22010	非法指示器参数值（INVALID INDICATOR PARAMETER VALUE）
22020	非法限制值（INVALID LIMIT VALUE）
22023	非法参数值（INVALID PARAMETER VALUE）
2201B	非法正则表达式（INVALID REGULAR EXPRESSION）
22009	非法时区显示值（INVALID TIME ZONE DISPLACEMENT VALUE）
2200C	非法使用逃逸字符（INVALID USE OF ESCAPE CHARACTER）

错误码	含义
2200G	最相关类型不匹配（MOST SPECIFIC TYPE MISMATCH）
22004	不允许NULL值（NULL VALUE NOT ALLOWED）
22002	NULL值不能做指示器参数（NULL VALUE NO INDICATOR PARAMETER）
22003	数字值超出范围（NUMERIC VALUE OUT OF RANGE）
22026	字串数据长度不匹配（STRING DATA LENGTH MISMATCH）
22001	字串数据右边被截断（STRING DATA RIGHT TRUNCATION）
22011	抽取子字串错误（SUBSTRING ERROR）
22027	截断错误（TRIM ERROR）
22024	未结束的C字串（UNTERMINATED C STRING）
2200F	零长度的字符串（ZERO LENGTH CHARACTER STRING）
22P01	浮点异常（FLOATING POINT EXCEPTION）
22P02	非法文本表现形式（INVALID TEXT REPRESENTATION）
22P03	非法二进制表现形式（INVALID BINARY REPRESENTATION）
22P04	错误的COPY格式（BAD COPY FILE FORMAT）
22P05	不可翻译字符（UNTRANSLATABLE CHARACTER）
23 类：违反完整性约束	
23000	违反完整性约束（INTEGRITY CONSTRAINT VIOLATION）

错误码	含义
23001	违反限制（RESTRICT VIOLATION）
23502	违反非空（NOT NULL VIOLATION）
23503	违反外键约束（FOREIGN KEY VIOLATION）
23505	违反唯一约束（UNIQUE VIOLATION）
23514	违反检查（CHECK VIOLATION）
24 类：非法游标状态	
24000	非法游标状态（INVALID CURSOR STATE）
25 类：非法事务状态	
25000	非法事务状态（INVALID TRANSACTION STATE）
25001	活跃的SQL状态（ACTIVE SQL TRANSACTION）
25002	分支事务已经激活（BRANCH TRANSACTION ALREADY ACTIVE）
25008	持有的游标要求同样的隔离级别（HELD CURSOR REQUIRES SAME ISOLATION LEVEL）
25003	对分支事务的不恰当的访问方式（INAPPROPRIATE ACCESS MODE FOR BRANCH TRANSACTION）
25004	对分支事务的不恰当的隔离级别（INAPPROPRIATE ISOLATION LEVEL FOR BRANCH TRANSACTION）
25005	分支事务没有活跃的SQL事务（NO ACTIVE SQL TRANSACTION FOR BRANCH TRANSACTION）
25006	只读的SQL事务（READ ONLY SQL TRANSACTION）

错误码	含义
25007	不支持混和的模式和数据语句（SCHEMA AND DATA STATEMENT MIXING NOT SUPPORTED）
25P01	没有活跃的SQL事务（NO ACTIVE SQL TRANSACTION）
25P02	在失败的SQL事务中（IN FAILED SQL TRANSACTION）
26 类：非法SQL语句名	
26000	非法SQL语句名（INVALID SQL STATEMENT NAME）
27 类：触发的数据改变违规	
27000	触发的数据改变违规（TRIGGERED DATA CHANGE VIOLATION）
28 类：非法授权声明	
28000	非法授权声明（INVALID AUTHORIZATION SPECIFICATION）
2B 类：依然存在依赖的优先级描述符	
2B000	依然存在依赖的优先级描述符（DEPENDENT PRIVILEGE DESCRIPTORS STILL EXIST）
2BP01	依赖性对象仍然存在（DEPENDENT OBJECTS STILL EXIST）
2D 类：非法的事务终止	
2D000	非法的事务终止（INVALID TRANSACTION TERMINATION）
2F 类：SQL过程异常	
2F000	SQL过程异常（SQL ROUTINE EXCEPTION）

错误码	含义
2F005	执行的函数没有返回语句（FUNCTION EXECUTED NO RETURN STATEMENT）
2F002	不允许修改SQL数据（MODIFYING SQL DATA NOT PERMITTED）
2F003	企图使用禁止的SQL语句（PROHIBITED SQL STATEMENT ATTEMPTED）
2F004	不允许读取SQL数据（READING SQL DATA NOT PERMITTED）
34 类：非法游标名	
34000	非法游标名（INVALID CURSOR NAME）
38 类：外部过程异常	
38000	外部过程异常（EXTERNAL ROUTINE EXCEPTION）
38001	不允许包含的SQL（CONTAINING SQL NOT PERMITTED）
38002	不允许修改SQL数据（MODIFYING SQL DATA NOT PERMITTED）
38003	企图使用禁止的SQL语句（PROHIBITED SQL STATEMENT ATTEMPTED）
38004	不允许读取SQL数据（READING SQL DATA NOT PERMITTED）
39 类：外部过程调用异常	
39000	外部过程调用异常（EXTERNAL ROUTINE INVOCATION EXCEPTION）
39001	返回了非法的SQLSTATE（INVALID SQLSTATE RETURNED）
39004	不允许空值（NULL VALUE NOT ALLOWED）
39P01	违反触发器协议（TRIGGER PROTOCOL VIOLATED）

错误码	含义
39P02	违反SRF协议（SRF PROTOCOL VIOLATED）
3B 类：保存点异常	
3B000	保存点异常（SAVEPOINT EXCEPTION）
3B001	无效的保存点声明（INVALID SAVEPOINT SPECIFICATION）
3D 类：非法数据库名	
3D000	非法数据库名（INVALID CATALOG NAME）
3F 类：非法模式名	
3F000	非法模式名（INVALID SCHEMA NAME）
40 类：事务回滚	
40000	事务回滚（TRANSACTION ROLLBACK）
40002	违反事务完整性约束（TRANSACTION INTEGRITY CONSTRAINT VIOLATION）
40001	串行化失败（SERIALIZATION FAILURE）
40003	不知道语句是否结束（STATEMENT COMPLETION UNKNOWN）
40P01	侦测到死锁（DEADLOCK DETECTED）
42 类：语法错误或者违反访问规则	
42000	语法错误或者违反访问规则（SYNTAX ERROR OR ACCESS RULE VIOLATION）

错误码	含义
42601	语法错误（SYNTAX ERROR）
42501	权限不够（INSUFFICIENT PRIVILEGE）
42846	无法进行类型转换（CANNOT COERCE）
42803	分组错误（GROUPING ERROR）
42830	非法的外键（INVALID FOREIGN KEY）
42602	非法名字（INVALID NAME）
42622	名字太长（NAME TOO LONG）
42939	保留名字（RESERVED NAME）
42804	数据类型不匹配（DATATYPE MISMATCH）
42P18	未决的数据类型（INDETERMINATE DATATYPE）
42809	错误的对象类型（WRONG OBJECT TYPE）
42703	未定义的字段（UNDEFINED COLUMN）
42883	未定义的函数（UNDEFINED FUNCTION）
42P01	未定义的表（UNDEFINED TABLE）
42P02	未定义的参数（UNDEFINED PARAMETER）
42704	未定义对象（UNDEFINED OBJECT）
42701	重复的字段（DUPLICATE COLUMN）

错误码	含义
42P03	重复的游标（DUPLICATE CURSOR）
42P04	重复的数据库（DUPLICATE DATABASE））
42723	重复的函数（DUPLICATE FUNCTION）
42P05	重复的准备好语句（DUPLICATE PREPARED STATEMENT）
42P06	重复的模式（DUPLICATE SCHEMA）
42P07	重复的表（DUPLICATE TABLE）
42712	重复的别名（DUPLICATE ALIAS）
42710	重复的对象（DUPLICATE OBJECT）
42702	模糊的字段（AMBIGUOUS COLUMN）
42725	模糊的函数（AMBIGUOUS FUNCTION）
42P08	模糊的参数（AMBIGUOUS PARAMETER）
42P09	模糊的别名（AMBIGUOUS ALIAS）
42P10	非法字段引用（INVALID COLUMN REFERENCE）
42611	非法字段定义（INVALID COLUMN DEFINITION）
42P11	非法游标定义（INVALID CURSOR DEFINITION）
42P12	非法的数据库定义（INVALID DATABASE DEFINITION）
42P13	非法函数定义（INVALID FUNCTION DEFINITION）

错误码	含义
42P14	非法准备好语句定义（INVALID PREPARED STATEMENT DEFINITION）
42P15	非法模式定义（INVALID SCHEMA DEFINITION）
42P16	非法表定义（INVALID TABLE DEFINITION）
42P17	非法对象定义（INVALID OBJECT DEFINITION）
44 类：违反 WITH CHECK 选项	
44000	违反 WITH CHECK 选项（WITH CHECK OPTION VIOLATION）
53 类：资源不够	
53000	资源不够（INSUFFICIENT RESOURCES）
53100	磁盘满（DISK FULL）
53200	内存耗尽（OUT OF MEMORY）
53300	太多连接（TOO MANY CONNECTIONS）
54 类：超过程序限制	
54000	超过程序限制（PROGRAM LIMIT EXCEEDED）
54001	语句太复杂（STATEMENT TOO COMPLEX）
54011	太多字段（TOO MANY COLUMNS）
54023	参数太多（TOO MANY ARGUMENTS）
55 类：对象不在预先要求的状态	

错误码	含义
55000	对象不在预先要求的状态（OBJECT NOT IN PREREQUISITE STATE）
55006	对象在使用中（OBJECT IN USE）
55P02	无法修改运行时参数（CANT CHANGE RUNTIME PARAM）
55P03	锁不可获得（LOCK NOT AVAILABLE）
57 类：操作者干涉	
57000	操作者干涉（OPERATOR INTERVENTION）
57014	查询被取消（QUERY CANCELED）
57P01	管理员关机（ADMIN SHUTDOWN）
57P02	崩溃关机（CRASH SHUTDOWN）
57P03	现在无法连接（CANNOT CONNECT NOW）
58 类：系统错误（PostgreSQL 自己内部的错误）	
58030	IO错误（IO ERROR）
58P01	未定义的文件（UNDEFINED FILE）
58P02	重复的文件（DUPLICATE FILE）
F0 类：配置文件错误	
F0000	配置文件错误（CONFIG FILE ERROR）
F0001	锁文件存在（LOCK FILE EXISTS）

错误码	含义
P0 类：PL/PGSQL 错误	
P0000	PL/PGSQL 错误（PLPGSQL ERROR）
P0001	抛出异常（RAISE EXCEPTION）
XX 类：内部错误	
XX000	内部错误（INTERNAL ERROR）
XX001	数据损坏（DATA CORRUPTED）
XX002	索引损坏（INDEX CORRUPTED）

5.数据写入方式概述

AnalyticDB for PostgreSQL 提供多种数据写入/加载方式，本文将介绍 INSERT 语句、COPY命令、OSS外表的并行写入、使用 Client SDK 写入四种写入方式。

其中 INSERT 语句和 COPY 命令的写入数据均通过 Master 节点，INSERT 语句每秒支持3MB的数据写入量，COPY 命令的写入性能快于 INSERT，可以达到每秒 30MB。OSS 外表为各个数据分区 `(Segment)` 并行读取，单分区从 OSS 加载的能力即可达到 30MB，整体加载速率随分区数线性扩展。

方式一：INSERT 语句

通过 INSERT 语句写入数据，为提升写入速度，建议单条 INSERT 语句拼装多个值 VALUE 执行，可实现每秒 3MB 数据量写入。

```
INSERT INTO performers (name, specialty) VALUES ('Sinatra', 'Singer'), …;
```

> **注意**
>
> 执行高吞吐的写入时，建议关闭 ORCA SQL 优化器。ORCA 适合复杂查询优化，但对简单语句，会带来不必要的解析和优化CPU计算代价。Session 会话级别关闭ORCA优化器命令：“set optimizer = off;”；或提工单在实例级别关闭。

方式二：COPY命令

使用\COPY命令，可以将本地的文本文件数据导入 AnalyticDB for PostgreSQL，本地的文本文件要求是格式化的，如通过逗号、分号或特有符号作为分割符号的文件。同时也支持用户使用 JDBC 执行 COPY 语句，JDBC 中封装了 CopyIn 方法。采用 COPY 方式，支持每秒 30MB 的数据写入或导入速度。

COPY命令的操作细节可以参见COPY命令的使用

方式三：OSS 外表导入和导出

通过 OSS 外表，数据可以从各个节点（Segment ）并行的写入或导出。整体写入或导出速度，随节点数线性扩展，单节点每秒支持 30MB 的数据写入。

建立OSS外表及数据写入和导出操作，可以参见使用OSS外表高速导入OSS数据

方式四：通过Client SDK 开发高效写入程序

AnalyticDB PostgreSQL版Client SDK通过 API 形式旨在为用户提供高性能写入数据到AnalyticDB PostgreSQL版的方式，支持用户定制化开发或对接写入程序。通过 SDK 开发写入程序，可简化在AnalyticDB PostgreSQL版中写入数据的流程，无需担心连接池、缓存等问题，相比较直接COPY/INSERT写入，通过并行化等内部机制有几倍性能提升。

Client SDK客户端API，请参照Client SDK的使用