ALIBABA CLOUD

阿里云

云原生数据仓库 AnalyticDB PostgreSQL 版

数据管理

文档版本：20200909

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.数据库管理

数据库(Database)是表、索引、视图、存储过程、操作符的集合。用户可以在一个AnalyticDB for PostgreSQL实例中创建多个数据库，但是客户端程序一次只能连接上并且访问一个数据库，无法跨数据库进行查询。

创建数据库

使用 `CREATE DATABASE` 命令创建一个新的数据库，命令如下：

```
CREATE DATABASE <dbname> [ [WITH] [OWNER [=] <dbowner>]
                [ENCODING [=] <encoding>]
```

说明：

- <dbname>：待创建的数据库名称。
- <dbowner>：拥有新数据库的数据库用户名，默认为执行该命令的用户。
- <encoding>：在新数据库中使用的字符集编码。指定一个字符串常量（例如 'SQL_ASCII'），一个整数编码号，默认为utf-8。

示例：

```
CREATE DATABASE mygpdb;
```

删除数据库

使用 `DROP DATABASE` 命令删除一个数据库。它会移除该数据库的元数据并且删除该数据库在磁盘上的目录及其中包含的数据，命令如下：

```
DROP DATABASE <dbname>
```

说明：

<dbname>：待删除的数据库名称。

示例：

```
DROP DATABASE mygpdb;
```

更多信息

详情请参考Pivotal Greenplum 官方文档。

2.Schema管理

Schema是数据库的命名空间，它是一个数据库内部的对象（表、索引、视图、存储过程、操作符）的集合。Schema在每个数据库中是唯一的。每个数据库都一个名为public的默认Schema。

如果用户没有创建任何Schema，对象会被创建在public schema中。所有的该数据库角色（用户）都在默认的public schema中拥有CREATE和USAGE特权。

创建Schema

使用 `CREATE SCHEMA` 命令来创建一个新的Schema，命令如下：

```
CREATE SCHEMA <schema_name> [AUTHORIZATION <username>]
```

> 说明
>
> - <schema_name>：schema名称。
> - <username>：如指定authorization username，则创建的schema属于该用户。否则，属于执行该命令的用户。

示例：

```
CREATE SCHEMA myschema;
```

设置Schema搜索路径

数据库的search_path用于配置参数设置Schema的搜索顺序。

使用 `ALTER DATABASE` 命令可以设置搜索路径。例如：

```
ALTER DATABASE mydatabase SET search_path TO myschema, public, pg_catalog;
```

您也可以使用 `ALTERROLE` 命令为特定的角色（用户）设置search_path。例如：

```
ALTER ROLE sally SET search_path TO myschema, public, pg_catalog;
```

查看当前Schema

使用 `current_schema()` 函数可以查看当前的Schema。例如：

```
SELECT current_schema();
```

您也可以使用 `SHOW` 命令查看当前的搜索路径。例如：

```
SHOW search_path;
```

删除Schema

使用 `DROP SCHEMA` 命令删除一个空的Schema。例如：

```
DROP SCHEMA myschema;
```

> **说明** 默认情况下，Schema它必须为空才可以删除。

删除一个Schema连同其中的所有对象（表、数据、函数等等），可以使用：

```
DROP SCHEMA myschema CASCADE;
```

更多信息

详情请参考Pivotal Greenplum 官方文档。

3.数据表管理

AnalyticDB forPostgreSQL数据库中的表与任何一种关系型数据库中的表类似，不同的是表中的行被分布在不同Segment上，表的分布策略决定了在不同Segment上面的分布情况。

创建普通表

CREATE TABLE命令用于创建一个表，创建表时可以定义以下内容：

- 表的列以及数据类型
- 表约束的定义
- 表分布定义
- 表存储格式
- 分区表定义

使用 `CREATE TABLE` 命令创建表，格式如下：

```
CREATE TABLE table_name (
[ { column_name data_type [ DEFAULT default_expr ]   -- 表的列定义
  [column_constraint [ ... ]                  -- 列的约束定义
]
  | table_constraint                         -- 表级别的约束定义
  ])
  [ WITH ( storage_parameter=value [, ... ] )      -- 表存储格式定义
  [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]  -- 表的分布键定义
  [ partition clause]                         -- 表的分区定义
```

示例：

示例中的建表语句创建了一个表，使用trans_id作为分布键，并基于 date 设置了RANGE分区功能。

```
CREATE TABLE sales (
 trans_id int,
 date date,
 amount decimal(9,2),
 region text)
 DISTRIBUTED BY (trans_id)
 PARTITION BY RANGE(date)
 (start (date '2018-01-01') inclusive
  end (date '2019-01-01') exclusive every (interval '1 month'),
  default partition outlying_dates);
```

创建临时表

临时表（Temporary Table）会在会话结束时自动删除，或选择性地在当前事务结束的时候删除，用于存储 临时中间结果。创建临时表的命令如下：

```
CREATE TEMPORARY TABLE table_name(...)
  [ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP}]
```

> 说明 临时表的行为在事务块结束时的行为可以通过上述语句中的ON COMMIT来控制。
>
> - PRESERVE ROWS：在事务结束时候保留数据，这是默认的行为。
> - DELETE ROWS：在每个事务块结束时，临时表的所有行都将被删除。
> - DROP：在当前事务结束时，会删除临时表。

示例：

创建一个临时表，事务结束时候删除该临时表。

```
CREATE TEMPORARY TABLE temp_foo (a int, b text) ON COMMIT DROP;
```

表约束的定义

您可以在列和表上定义约束来限制表中的数据，但是有以下一些限制：

- CHECK约束引用的列只能在其所在的表中。
- UNIQUE和PRIMARY KEY约束必须包含分布键列，UNIQUE和PRIMARY KEY约束不支持追加优化表和列存表。
- 允许FOREIGN KEY 约束，但实际上并不会做外键约束检查。
- 分区表上的约束必须应用到所有的分区表上，不能只应用于部分分区表。

约束命令格式如下：

```
UNIQUE ( column_name [, ... ] )
  | PRIMARY KEY ( column_name [, ... ] )
  | CHECK ( expression )
  | FOREIGN KEY ( column_name [, ... ] )
        REFERENCES table_name [ ( column_name [, ... ] ) ]
        [ key_match_type ]
        [ key_action ]
        [ key_checking_mode ]
```

检查约束（Check Constraints)

检查约束（Check Constraints）指定列中的值必须满足一个布尔表达式，例如：

```
CREATE TABLE products
        ( product_no integer,
          name text,
          price numeric CHECK (price > 0) );
```

非空约束（Not-Null Constraints)

非空约束（Not-Null Constraints）指定列不能有空值，例如：

```
CREATE TABLE products
    ( product_no integer NOT NULL,
      name text NOT NULL,
      price numeric );
```

唯一约束（Unique Constraints)

唯一约束（Unique Constraints）确保一列或者一组列中包含的数据对于表中所有的行都是唯一的。包含唯一约束的表必须是哈希分布，并且约束列需要包含分布键列，例如：

```
CREATE TABLE products
    ( product_no integer UNIQUE,
      name text,
      price numeric)
    DISTRIBUTED BY (product_no);
```

主键（Primary Keys)

主键约束（Primary Keys Constraints）是一个UNIQUE约束和一个 NOT NULL约束的组合。包含主键约束的表必须是哈希分布，并且约束列需要包含分布键列。如果一个表具有主键，这个列（或者这一组列）会被默认选中为该表的分布键，例如：

```
CREATE TABLE products
    ( product_no integer PRIMARY KEY,
      name text,
      price numeric)
    DISTRIBUTED BY (product_no);
```

更多信息

详情请参考Pivotal Greenplum 官方文档。

4.表分布定义

选择表分布策略

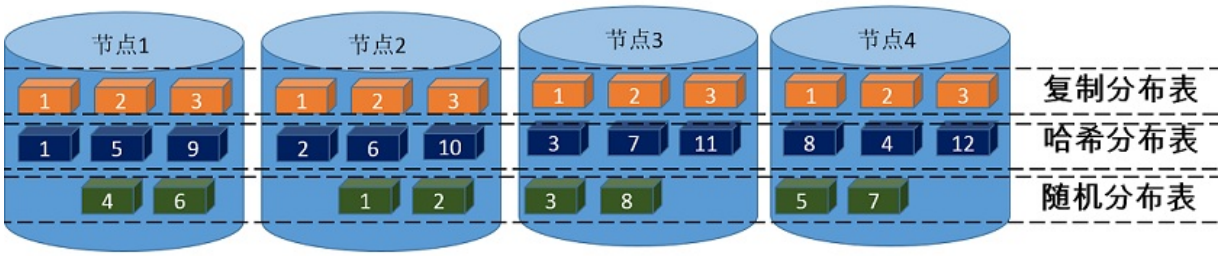
AnalyticDB for PostgreSQL 支持三种数据在节点间的分布方式，按指定列的哈希（HASH）分布、随机（RANDOMLY）分布、复制（REPLICATED）分布。

```
CREATE TABLE table_name (...) [ DISTRIBUTED BY (column [,..] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
```

> 说明 ADB PG 4.3版本只支持哈希(HASH)分布和随机(RANDOMLY)分布，复制(REPLICATED)分布为 6.0 版本新增加特性。

建表语句 `CREATE TABLE` 支持如下三个分布策略的子句：

- `DISTRIBUTED BY (column, [ ... ])` 指定数据按分布列的哈希值在节点（Segment）间分布，根据分布列哈希值将每一行分配给特定节点（Segment）。相同的值将始终散列到同一个节点。选择唯一的分布键（例如Primary Key）将确保较均匀的数据分布。哈希分布是表的默认分布策略， 如果创建表时未提供DISTRIBUTED子句，则将PRIMARY KEY或表的第一个合格列用作分布键。如果表中没有合格的列，则退化为随机分布策略。
- `DISTRIBUTED RANDOMLY` 指定数据按循环的方式均匀分配在各节点 （Segment） 间，与哈希分布策略不同，具有相同值的数据行不一定位于同一个segment上。虽然随机分布确保了数据的平均分布，但只建议当表没有合适的离散分布的数据列作为哈希分布列时采用随机分布策略。
- `DISTRIBUTED REPLICATED` 指定数据为复制分布，即每个节点（Segment）上有该表的全量数据，这种分布策略下表数据将均匀分布，因为每个segment都存储着同样的数据行，当有大表与小表join，把足够小的表指定为replicated也可能提升性能。



示例：

示例中的建表语句创建了一个哈希（Hash）分布的表，数据将按分布键的哈希值被分配到对应的节点 Segment 数据节点。

```
CREATE TABLE products (name varchar(40),
                prod_id integer,
                supplier_id integer)
                DISTRIBUTED BY (prod_id);
```

示例中的建表语句创建了一个随机（Randomly）分布的表，数据被循环着放置到各个 Segment 数据节点。 当表没有合适的离散分布的数据列作为哈希分布列时，可以采用随机分布策略。

```
CREATE TABLE random_stuff (things text,
                doodads text,
                etc text)
                DISTRIBUTED RANDOMLY;
```

示例中的建表语句创建了一个复制（Replicated）分布的表，每个 Segment 数据节点都存储有一个全量的表数据。

```
CREATE TABLE replicated_stuff (things text,
                doodads text,
                etc text)
                DISTRIBUTED REPLICATED;
```

对于按分布键的简单查询，包括 UPDATE/DELETE 等语句，AnalyticDB for PostgreSQL 具有按节点的分布键进行数据节点裁剪的功能，例如products表使用prod_id作为分布键，以下查询只会被发送到满足prod_id=101的segment上执行，从而极大提升该 SQL 执行性能：

```
select * from products where prod_id = 101;
```

表分布键选择原则

合理规划分布键，对表查询的性能至关重要，有以下原则需要关注：

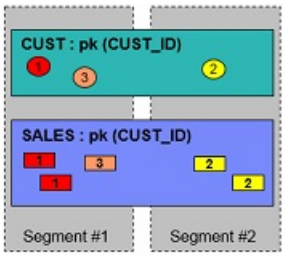
- 选择数据分布均匀的列或者多个列：若选择的分布列数值分布不均匀，则可能导致数据倾斜。某些Segment分区节点存储数据多(查询负载高)。根据木桶原理，时间消耗会卡在数据多的节点上。故不应选择bool类型，时间日期类型数据作为分布键。
- 选择经常需要 JOIN 的列作为分布键，可以实现图一所示**本地关联（Collocated JOIN）**计算，即当JOIN键和分布键一致时，可以在 Segment分区节点内部完成JOIN。否则需要将一个表进行重分布（Redistribute motion）来实现图二所示**重分布关联（Redistributed Join）**或者广播其中小表(Broadcast motion)来实现图三所示**广播关联（Broadcast Join），后两种方式都会有较大的网络开销**。
- 尽量选择高频率出现的查询条件列作为分布键，从而可能实现按分布键做节点 segment 的裁剪。
- 若未指定分布键，默认表的主键为分布键，若表没有主键，则默认将第一列当做分布键。
- 分布键可以被定义为一个或多个列。例如：

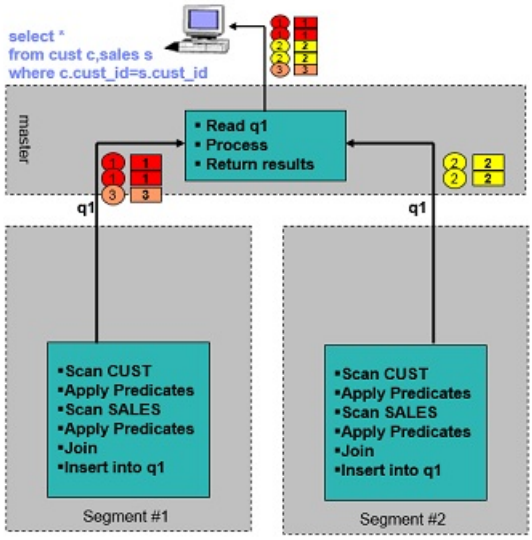
  ```
  create table t1(c1 int, c2 int) distributed by (c1,c2);
  ```

- 谨慎选择随机分布DISTRIBUTED RANDOMLY，这将使得上述本地关联，或者节点裁剪不可能实现。

图一：本地关联 (Collocated JOIN)

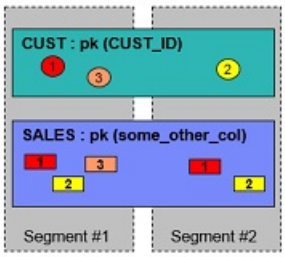
- CUST 表 和 SALES 表都采用 CUST_ID 为分布列
- 关联（JOIN）分别在各自 Segment内完成，没有数据网络传输（Motion）

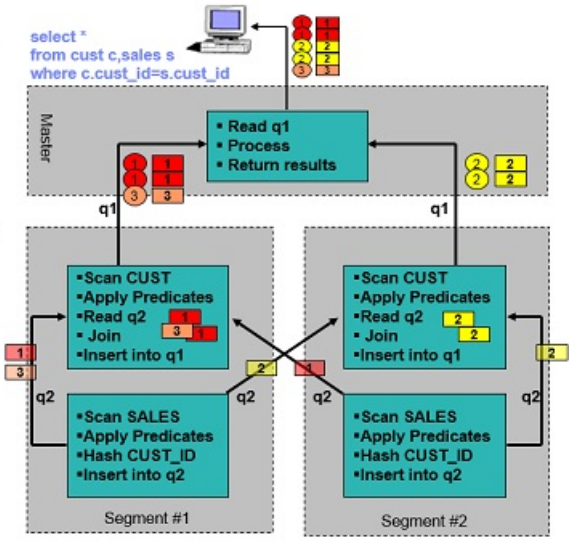


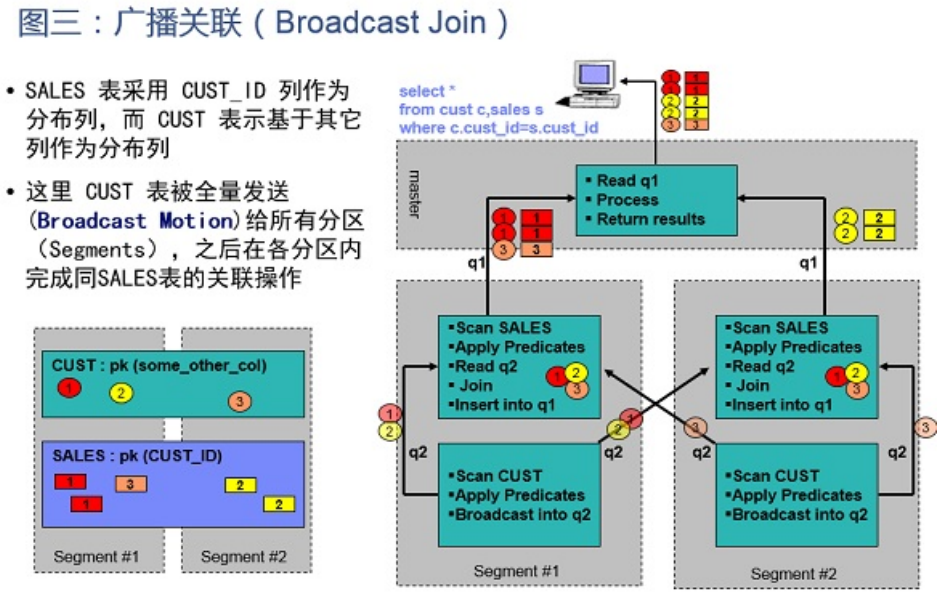


图二：重分布关联（Redistuted Join）

- CUST 表采用 CUST_ID 列作为分布列，而 SALES 表示基于其它列作为分布列
- 这里 SALES 表按 CUST_ID列的HASH值被发送（Redistribute Motion）到对应的分区（Segments），之后在各分区内完成同CUST表的关联操作







表分布键的约束

- 分布键的列不能被更新（UPDATE）。
- 主键和唯一键必须包含分布键。例如：

```
create table t1(c1 int, c2 int, primary key (c1)) distributed by (c2);
```

> 说明 由于主键c1不包含分布键c2，所以建表语句返回失败。
>
> ```
> ERROR: PRIMARY KEY and DISTRIBUTED BY definitions incompatible
> ```

- Geometry类型和用户自定义数据类型不能作为分布键。

数据倾斜检查和处理

当某些表上的查询性能差时，可以查看是否是分区键设置不合理造成了数据倾斜，例如：

```
create table t1(c1 int, c2 int) distributed by (c1);
```

您可以通过下述语句来查看表的数据倾斜情况。

```
select gp_segment_id,count(1) from  t1 group by 1 order by 2 desc;
 gp_segment_id | count
---------------+--------
             2 | 131191
             0 |     72
             1 |     68
(3 rows)
```

如果发现某些 Segment上存储的数据明显多于其他 Segment，该表存在数据倾斜。建议选取数据分布平均的列作为分布列，比如通过 `ALTER TABLE` 命令更改C2为分布键。

```
alter table t1 set distributed by (c2);
```

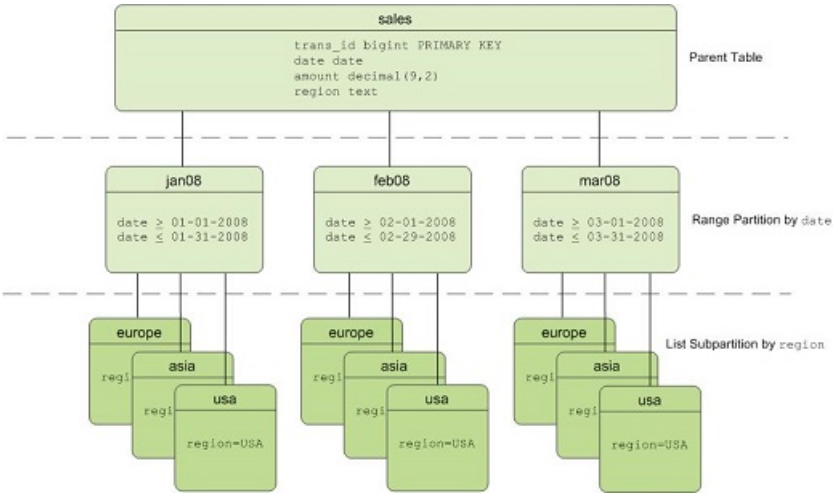
表t1的分布键被改为c2，该表的数据按照c2被重新分布，数据不再倾斜。

5.表分区定义

将大表定义为分区表，从而将其分成较小的存储单元，根据查询条件，会只扫描满足条件的分区而避免全表扫描，从而显著提升查询性能。

支持的表分区类型

- **范围（RANGE）分区**：基于一个数值型范围划分数据，例如按着日期区间定义。
- **值（LIST）分区**：基于一个值列表划分数据，例如按着 城市属性定义。
- **多级分区表**：上述两种类型的多级组合。



上图为一个多级分区表设计实例，一级分区采用按月的区间（Range）分区，二级分区采用按地区的值（List）分区设计。

创建范围(RANGE)分区表

用户可以通过给出一个START值、一个END值以及一个定义分区增量值的子句让数据库自动产生分区。默认情况下，START值总是被包括在内而END值总是被排除在外，例如：

```
CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( START (date '2016-01-01') INCLUSIVE
  END (date '2017-01-01') EXCLUSIVE
  EVERY (INTERVAL '1 day') );
```

也可以创建一个按数字范围分区的表，使用单个数字数据类型列作为分区键列，例如：

```
CREATE TABLE rank (id int, rank int, year int, gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
( START (2006) END (2016) EVERY (1),
  DEFAULT PARTITION extra );
```

创建值（LIST)分区表

一个按列表分区的表可以使用任意允许等值比较的数据类型列作为它的分区键列。对于列表分区，您必须为每一个用户想要创建的分区（列表值）声明一个分区说明，例如：

```
CREATE TABLE rank (id int, rank int, year int, gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
  PARTITION boys VALUES ('M'),
  DEFAULT PARTITION other );
```

创建多级分区表

支持创建多级的分区表。下述建表语句创建了具有三级表分区的表。一级分区在month字段上做RANGE分区，二级分区在region上做了LIST分区。

```
CREATE TABLE sales
  (id int, year int, month int, day int, region text)
DISTRIBUTED BY (id)
PARTITION BY RANGE (month)
  SUBPARTITION BY LIST (region)
    SUBPARTITION TEMPLATE (
    SUBPARTITION usa VALUES ('usa'),
    SUBPARTITION europe VALUES ('europe'),
    SUBPARTITION asia VALUES ('asia'),
    DEFAULT SUBPARTITION other_regions)
(START (1) END (13) EVERY (1),
DEFAULT PARTITION other_months );
```

分区定义的粒度

通常分区表的定义都涉及到粒度问题，比如按时间分区，究竟是按天，按周，按月等。粒度越细，每张表的数据就越少，但是分区的数量就越多，反之亦然。关于分区的数量，没有绝对的标准，一般分区的数量在200左右已经算是比较多了。分区表数目过多，会有多方面的影响，比如查询优化器生成执行计划较慢，同时很多维护工作也会变慢，比如VACUUM等。

> **注意**
>
> 请对多级分区格外谨慎，因为分区文件的数量可能会增长得非常快。例如，如果一个表被按照月和城市划分并且有24个月以及1,00个城市，那么表分区的总数就是2400。特别对于列存表，会把每一列存在一个物理表中，因此如果这个表有100个列，系统就需要为该表管理十多万个文件。

分区表查询优化

AnalyticDB for PostgreSQL 支持分区表的分区裁剪功能，根据查询条件会只扫描所需的数据分区而避免扫描整个表的全部内容，提升查询性能。例如对于如下查询：

```
explain
 select * from sales
 where year = 2008
  and  month = 1
  and  day = 3
  and region = 'usa';
```

由于查询条件落在一级分区2008的二级子分区1的三级子分区 'usa' 上，查询只会扫描读取这一个三级子分区数据。如下其查询计划显示，总计468个三级子分区中，只有一个分区被读取。

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..431.00 rows=1 width=24)
 ->  Sequence  (cost=0.00..431.00 rows=1 width=24)
     ->  Partition Selector for sales (dynamic scan id: 1)  (cost=10.00..100.00 rows=25 width=4)
         Filter: year = 2008 AND month = 1 AND region = 'usa'::text
         Partitions selected:  1 (out of 468)
     ->  Dynamic Table Scan on sales (dynamic scan id: 1)  (cost=0.00..431.00 rows=1 width=24)
         Filter: year = 2008 AND month = 1 AND day = 3 AND region = 'usa'::text
```

查询分区表定义

可以通过如下 SQL 语句查询一个分区表的所有分区定义信息：

```
SELECT
 partitionboundary,
 partitiontablename,
 partitionname,
 partitionlevel,
 partitionrank
FROM pg_partitions
WHERE tablename='sales';
```

分区表维护

分区表支持多种分区管理操作，包括新增分区，删除分区，重命名分区，清空截断分区，交换分区，分裂分区等，详情请参考Greenplum 官方文档。

6.表存储格式定义

AnalyticDB for PostgreSQL支持多种存储格式。当您创建一个表时，可以选择表的存储格式为行存表或者列存表。

行存表

默认情况下，AnalyticDB for PostgreSQL 创建的是行存表（Heap Table），使用的 PostgreSQL 堆存储模型。行存表适合数据更新较频繁的场景，或者采用INSERT方式的实时写入的场景，同时当行存表建有B-Tree索引时，具备更好的点查询数据检索性能。

示例：

下述语句创建了一个默认堆存储类型的行存表。

```
CREATE TABLE foo (a int, b text) DISTRIBUTED BY (a);
```

> **说明** 当采用数据传输服务 DTS 写入 ADB PG数据时，ADB PG的目标表应设计为行存表，而不要采用列存表。DTS 为准实时数据同步方式，除INSERT 外，即支持 UPDATE 和 DELETE 等较多更新操作的同步。

列存表

列存表（Column-Oriented Table）的按列存储格式，数据访问只会读取涉及的列，适合少量列的数据查询、聚集等数据仓库应用场景，在此类场景中，列存表能够提供更高效的I/O。但列存表不适合频繁的更新操作或者大批量的INSERT写入场景，这时其效率较低。列存表的数据写入建议采用COPY等批量加载方式。列存表可以提供平均 3-5倍的较高数据压缩率。

示例：

列存表必须是追加优化表。例如，要创建一个列存表，必须指定为 "appendonly=true" 。

```
CREATE TABLE bar (a int, b text)
   WITH (appendonly=true, orientation=column)
   DISTRIBUTED BY (a);
```

压缩

压缩主要用于列存表或者追加写（"appendonly=true"）的行存表，有以下两种类型的压缩可用。

- 应用于整个表的表级压缩。
- 应用到指定列的列级压缩。用户可以为不同的列应用不同的列级压缩算法。

目前AnalyticDB for PostgreSQL支持的压缩算法如下：

- 4.3 版本支持zlib、rle_type
- 6.0 版本支持zstd、zlib、rle_type、lz4

> **说明** 也可以指定QuickLZ压缩算法，但内部会使用zlib算法替换，另外rle_type算法只适用于列存表。

示例：

创建一个使用zlib压缩且压缩级别为5的列存表。

```
CREATE TABLE foo (a int, b text)
  WITH (appendonly=true, orientation=column, compresstype=zlib, compresslevel=5);
```

创建一个使用zstd压缩且压缩级别为9的列存表。

```
CREATE TABLE foo (a int, b text)
  WITH (appendonly=true, orientation=column, compresstype=zstd, compresslevel=9);
```

7.索引管理

索引类型

AnalyticDB for PostgreSQL支持B-tree索引，位图索引，GIN索引（6.0版本支持）和GiST索引（6.0版本支持），不支持Hash索引。

> 说明 B-tree索引是默认的索引类型。位图索引（Bitmap Index）为每一个键值都存储一个位图，位图索引提供了和常规索引相同的功能但索引空间大大减少。对于拥有100至100,000个可区分值的列并且当被索引列常常与其他被索引列联合查询时，位图索引表现最好。

创建索引

使用 `CREATE INDEX` 命令在表上创建一个B-tree索引。

示例：

在employee表的gender列上创建一个B-tree索引。

```
CREATE INDEX gender_idx ON employee (gender);
```

在films表中的title列上创建一个位图索引。

```
CREATE INDEX title_bmp_idx ON films USING bitmap (title);
```

在lineitem表的l_comment列上创建一个GIN索引支持全文搜索（6.0版本支持）。

```
CREATE INDEX lineitem_idx ON lineitem USING gin(to_tsvector('english', l_comment));
```

在arrayt表的intarray数组类型列上创建一个GIN索引（6.0版本支持）。

```
CREATE INDEX arrayt_idx ON arrayt USING gin(intarray);
```

在customer表的c_comment列上创建一个GiST索引支持全文搜索（6.0版本支持）。

```
CREATE INDEX customer_idx ON customer USING gist(to_tsvector('english', c_comment));
```

重建索引

使用 `REINDEX INDEX` 命令重建索引。

示例：

重建索引my_index。

```
REINDEX INDEX my_index;
```

重建my_table表上的所有索引。

```
REINDEX TABLE my_table;
```

删除索引

使用 `DROP INDEX` 命令删除一个索引。

示例：

```
DROP INDEX title_idx;
```

> **说明** 在加载大量数据时，先删除所有索引并载入数据，然后重建索引会更快。

索引的选择原则

- 用户的查询负载。

  索引能改进查询返回单一记录或者非常小的数据集的性能，例如OLTP类型查询。

- 压缩表。

  在被压缩过的追加优化表上，索引也可以提高返回一个目标行集合的查询的性能，只有必要的行才会被解压。

- 避免在频繁更新的列上建立索引。

  在一个被频繁更新的列上建立索引会增加该列被更新时所要求的写操作数据量。

- 创建选择性高的B-tree索引。

  例如，如果一个表有1000行并且一个列中有800个不同的值，则该索引的选择度为0.8，索引的选择性会比较高。唯一索引的选择度总是1.0。

- 为低选择度的列使用位图索引。对于区分值区间在100至100,000之间的列位图索引表现最好。
- 索引在连接中用到的列。

  在被用于频繁连接的一个列（例如一个外键列）上的索引能够提升连接性能，因为这让查询优化器有更多的连接方法可以使用。

- 索引在谓词中频繁使用的列。

  频繁地在WHERE子句中被引用的列是索引的首选。

- 避免创建重叠的索引。

  例如在多列索引中，具有相同前导列的索引是冗余的。

- 批量载入前删掉索引。

  对于载入大量数据到一个表中，请考虑先删掉索引并且在数据装载完成后重建它们，这常常比更新索引更快。

- 测试并且比较使用索引和不使用索引的查询性能。

  只有被索引列的查询性能有提升时才增加索引。

- 创建完索引，建议对标执行analyze。

更多信息

详情请参考Pivotal Greenplum 官方文档。

8.视图管理

视图允许用户保存常用的或者复杂的查询。视图没有物理存储，当用户访问时，视图会作为一个子查询运行。

创建视图

使用 `CREATE VIEW` 命令创建一个查询的视图。

示例：

```
CREATE VIEW myview AS SELECT * FROM products WHERE kind = 'food';
```

> **说明** 视图会忽略存储在视图中的ORDERBY以及SORT操作。

删除视图

使用 `DROP VIEW` 命令删除一个视图。

示例：

```
DROP VIEW myview;
```

更多信息

详情请参考Pivotal Greenplum 官方文档。

9.物化视图管理

物化视图类似于视图，允许用户保存经常使用的或复杂的查询。物化视图有实际的物理储存，但不支持直接写入更新数据。在查询中引用物化视图时，数据直接从物化视图返回。物化视图的数据不会自动刷新，因此可能不是最新的，但是访问物化视图中的数据要比直接或通过视图访问底层表中的数据快得多。物化视图在用户能够容忍定期更新数据的情况下拥有巨大的性能优势。

创建物化视图

使用命令 `CREATE MATERIALIZED VIEW` 创建一个查询的物化视图。

```
CREATE MATERIALIZED VIEW my_materialized_view as
   SELECT * FROM people WHERE age > 40
   DISTRIBUTED BY (id);

SELECT * from my_materialized_view ORDER BY age;
   id    |   name   |   city   | age
------------+------------+------------+-----
 004      | zhaoyi    | zhenzhou  |  44
 005      | xuliui    | jiaxing   |  54
 006      | maodi     | shanghai  |  55
(3 rows)
```

物化视图定义中的查询仅用于填充物化视图。物化视图定义和普通表的定义一样（除了不能指定OID）。DISTRIBUTED BY在创建时可选，若没有该语句会选择默认的第一个可分区列进行分区。

> **说明** 物化视图会忽略存储在物化视图中的ORDER BY以及SORT操作。

刷新或禁用物化视图

使用 `REFRESH MATERIALIZED VIEW` 命令刷新物化视图数据。

```
INSERT INTO people VALUES('007','sunshen','shenzhen',60);

SELECT * from my_materialized_view ORDER BY age;
   id   |   name   |   city   | age
------------+------------+------------+-----
 004      | zhaoyi    | zhenzhou  | 44
 005      | xuliui    | jiaxing   | 54
 006      | maodi     | shanghai  | 55
(3 rows)

REFRESH MATERIALIZED VIEW my_materialized_view;

SELECT * from my_materialized_view ORDER BY age;
   id   |   name   |   city   | age
------------+------------+------------+-----
 004      | zhaoyi    | zhenzhou  | 44
 005      | xuliui    | jiaxing   | 54
 006      | maodi     | shanghai  | 55
 007      | sunshen   | shenzhen  | 60
(4 rows)
```

使用 `With NO DATA` 子句会删除当前数据且不生成新数据，还会使物化视图处于无法扫描状态。如果试图查询无法扫描状态的物化视图，将返回一个错误。

```
REFRESH MATERIALIZED VIEW my_materialized_view With NO DATA;

SELECT * from my_materialized_view ORDER BY age;
ERROR:  materialized view "my_materialized_view" has not been populated
HINT:  Use the REFRESH MATERIALIZED VIEW command.

REFRESH MATERIALIZED VIEW my_materialized_view;

SELECT * from my_materialized_view ORDER BY age;
   id   |   name   |   city   | age
------------+------------+------------+-----
 004      | zhaoyi    | zhenzhou  | 44
 005      | xuliui    | jiaxing   | 54
 006      | maodi     | shanghai  | 55
 007      | sunshen   | shenzhen  | 60
(4 rows)
```

删除物化视图

使用 `DROP MATERIALIZED VIEW` 命令删除一个物化视图。

```
CREATE MATERIALIZED VIEW depend_materialized_view as
   SELECT * FROM my_materialized_view WHERE age > 50
   DISTRIBUTED BY (id);

DROP MATERIALIZED VIEW depend_materialized_view;
```

使用 `DROP MATERIALIZED VIEW ... CASCADE` 命令还会删除所有依赖该物化视图的对象。例如，如果另一个物化视图依赖于即将被删除的物化视图，那么另一个物化视图也将被删除。

> 注意 如果没有CASCADE选项，DROP MATERIALIZED VIEW命令就会失败。

```
CREATE MATERIALIZED VIEW depend_materialized_view as
   SELECT * FROM my_materialized_view WHERE age > 50
   DISTRIBUTED BY (id);

DROP MATERIALIZED VIEW my_materialized_view;
ERROR:  cannot drop materialized view my_materialized_view because other objects depend on it
DETAIL:  materialized view depend_materialized_view depends on materialized view my_materialized_
view
HINT:  Use DROP ... CASCADE to drop the dependent objects too.

DROP MATERIALIZED VIEW my_materialized_view CASCADE;
```

适用场景

- 可以容忍数据不是最新或者定期手动更新的场景。
- 经常使用的或者特别复杂的查询。
- 物化视图结合外部数据源（如OSS外表、ODPS外表）可以实现更快的查询分析。外部数据通过物化视图实现本地存储，同时也可以对物化视图创建索引。

更多信息

详情请参考Pivotal Greenplum 官方文档

10.空间回收

背景信息

表中的数据被删除或更新后（UPDATE/DELETE），物理存储层面并不会直接删除数据，而是标记这些数据不可见，所以会在数据页中留下很多“空洞”，在读取数据时，这些“空洞”会随数据页一起加载，拖慢数据扫描速度，需要定期回收删除的空间。

空间回收方法

使用 `VACUUM` 命令，可以对表进行重新整理，回收空间，以便获取更好的数据读取性能。VACUUM命令如下：

```
VACUUM [FULL] [FREEZE] [VERBOSE] [table];
```

VACUUM 会在页内进行整理，VACUUM FULL会跨数据页移动数据。 VACUUM执行速度更快， VACUUM FULL执行地更彻底，但会请求排他锁。建议定期对系统表进行VACUUM（每周一次）。

使用建议

什么情况下做VACUUM?

- 不锁表回收空间，只能回收部分空间。
- 频率：对于有较多实时更新的表，每天做一次。
- 如果更新是每天一次批量进行的，可以在每天批量更新后做一次。
- 对系统影响：不会锁表，表可以正常读写。会导致CPU、I/O使用率增加，可能影响查询的性能。

什么情况下做VACUUM FULL?

- 锁表，通过重建表,可回收所有空洞空间。对做了大量更新后的表，建议尽快执行VACUUM FULL。
- 频率：至少每周执行一次。如果每天会更新几乎所有数据，需要每天做一次。
- 对系统影响：会对正在进行vacuum full的表锁定，无法读写。会导致CPU、I/O使用率增加。建议在维护窗口进行操作。

查询需要执行VACUUM的表

AnalyticDB for PostgreSQL提供了一个gp_bloat_diag视图，统计当前页数和实际需要页数的比例。通过analyze table来收集统计信息之后，查看该视图。

```
gpadmin=# SELECT * FROM gp_toolkit.gp_bloat_diag;
bdirelid | bdinspname | bdirelname | bdirelpages | bdiexppages |              bdidiag
----------+------------+------------+-------------+-------------+---------------------------------------
   21488 | public     | t1         |          97 |           1 | significant amount of bloat suspected
(1 row)
```

其结果只包括发生了中度或者显著膨胀的表。当实际页面数和预期页面的比率超过4但小于10时，就会报告为中度膨胀。当该比率超过10时就会报告显著膨胀。对于这些表，可以考虑进行VACUUM FULL来回收空间。

VACUUM FREEZE的使用

AnalyticDB for PostgreSQL执行的所有事务都有唯一的事务ID(XID)，XID是单调递增的，上限是20亿。

随着数据库执行事务的增多，为防止XID超过极限，在XID接近xid_stop_limit-xid_warn_limit(默认500000000)时， ADB for PG会对执行事务的sql返回warning信息，提醒用户：

```
WARNING: database "database_name" must be vacuumed within number_of_transactions transactions
```

用户可通过手动执行VACUUM FREEZE当前database来缩小XID。

如果忽略这个warning信息，在XID继续增长到超过xid_stop_limit(默认1000000000)时， ADB for PG会拒绝新的事务执行，并返回报错信息：

```
FATAL: database is not accepting commands to avoid wraparound data loss in database "database_name"
```

此时，您需要通过提交工单联系阿里云工程师来解决此问题。

11.磁盘配额管理

Diskquota 可以对云原生数据仓库 AnalyticDB PostgreSQL (简称 ADB PG) 的磁盘配额进行管理，ADB PG 数据库可以允许超级用户为 schema 和 role 设置磁盘使用配额。本文将介绍 Diskquota 如何在云原生数据仓库 AnalyticDB PostgreSQL版中创建和使用。

启用和关闭Diskquota

1. 创建Diskquota数据库，Diskquota模块使用此数据库存储启用该模块的数据库列表。

```
$ createdb diskquota;
```

2. 请提交工单联系ADB PG后台技术人员，将Diskquota加入shared_preload_libraries并重启ADB PG实例。
3. 启用Diskquota。

```
=# CREATE EXTENSION diskquota;
CREATE EXTENSION
```

4. 关闭Diskquota。

```
=# drop extension diskquota;
DROP EXTENSION
```

说明 如果在已经包含数据的数据库中使用Diskquota，则必须初始化Diskquota 表。文件比较多时，需要耗费一定时间。

```
SELECT diskquota.init_table_size_table();
```

设置 schema 或 role 的磁盘配额大小

- 设置 schema 的磁盘配额。

```
=# SELECT diskquota.set_schema_quota('adbpg1', '1MB');
 set_schema_quota
------------------


(1 row)
```

- 设置 role 的磁盘配额。

```
=# select diskquota.set_role_quota('u1', '250 MB');
 set_role_quota
----------------


(1 row)
```

> **说明** 磁盘配额以 MB、GB、TB、PB 为单位。 当设置为-1时，代表取消磁盘配额限制。
> Diskquota 在查询前与磁盘配额和黑名单比较，当超过配额时自动加入黑名单或本身在黑名单中便取消执行，属于软限制。

示例 对 schema 进行磁盘配额管理

1. 创建数据库与 schema 。

```
$createdb myadbpg
$psql myadbpg

=# CREATE EXTENSION diskquota;       #启动diskquota
CREATE EXTENSION

=# CREATE SCHEMA adbpg1;
CREATE SCHEMA
```

2. 设置 schema 磁盘配额。

```
=# SELECT diskquota.set_schema_quota('adbpg1', '1MB');
 set_schema_quota
------------------

(1 row)
```

3. 创建表并插入数据。

```
=# SET search_path TO adbpg1;
SET

=# CREATE TABLE a(i int);

=# INSERT INTO a SELECT generate_series(1,100);
INSERT 0 100

=# INSERT INTO a SELECT generate_series(1,10000000);
INSERT 0 10000000
```

4. 超出配额时发生错误，并禁止插入。

```
=# INSERT INTO a SELECT generate_series(1,100);
ERROR:  schema's disk space quota exceeded with name:adbpg1
```

5. 通过将配额设置为 *-1* 取消 adbpg1 的磁盘配额限制，然后再次插入少量数据。

   INSERT命令之前的5秒钟睡眠可确保在运行命令之前更新磁盘配额表大小数据。

```
=# SELECT diskquota.set_schema_quota('adbpg1', '-1');
 set_schema_quota
------------------

(1 row)
```

```
=#  SELECT pg_sleep(5);
 pg_sleep
----------

(1 row)

=# INSERT INTO a SELECT generate_series(1,100);
INSERT 0 100
```

查看磁盘使用情况

- 查看 schema 的磁盘使用情况。

```
 =#  SELECT * FROM diskquota.show_fast_schema_quota_view;
 schema_name | schema_oid | quota_in_mb | nspsize_in_bytes
-------------+------------+-------------+------------------
 adbpg1      |      16806 |        2000 |        721321984
(1 row)
```

- 查看 role 的磁盘使用情况。

```
=# SELECT * FROM diskquota.show_fast_role_quota_view;
 role_name | role_oid | quota_in_mb | rolsize_in_bytes
-----------+----------+-------------+------------------
 u1        |    16810 |         250 |                0
(1 row)
```

> **说明** 启用 Diskquota 后经 ADB PG官方测试，有低于 2%-3%的性能损失。