阿里云
HybridDB for PostgreSQL

用户指南

文档版本：20190221

为了无法计算的价值 | 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
	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。	**禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *{stand | slave}*

目录

法律声明.....I
通用约定.....I
1 基础操作..... 1
2 管理实例..... 4
2.1 创建实例.....4
2.2 申请外网地址.....4
2.3 释放外网地址.....5
2.4 升级实例规格.....6
2.5 修改连接地址.....8
2.6 按量付费转包年包月.....8
2.7 重启实例.....9
2.8 开启SQL审计.....10
2.9 释放实例.....11
3 数据迁移.....12
4 用户权限管理.....13
5 插件管理.....15
6 JSON 数据类型操作..... 17
7 HyperLogLog 的使用..... 24
8 使用 Create Library 命令.....26
9 使用 PL ／ Java UDF..... 28
10 使用排序键（SortKey）..... 30

1 基础操作

对基于 Greenplum Database 的操作，HybridDB for PostgreSQL 与 Greenplum Database 开源数据库基本一致，包括 schema、类型支持、用户权限等。除 Greenplum Database 开源数据库一些特有的操作，如分布键、AO表等，其它操作皆可参见 PostgreSQL。

本文介绍了 HybridDB for PostgreSQL 的基础操作：创建数据库、创建分布键和查询。

创建数据库

HybridDB for PostgreSQL 中创建数据库的操作与 PostgreSQL 相同，可以通过 SQL 来执行，如在 `psql` 连接到 Greenplum 后执行如下命令：

```
=> create database mygpdb;
CREATE DATABASE
=> \c mygpdb
psql (9.4.4, server 8.3devel)
You are now connected to database "mygpdb" as user "mygpdb".
```

创建分布键

在 HybridDB for PostgreSQL 中，表分布在所有的 Segment 上，其分布规则是 HASH 或者随机。在建表时，指定分布键；当导入数据时，会根据分布键计算得到的 HASH 值分配到特定的 Segment 上。

```
=> create table vtbl(id serial, key integer, value text, shape cuboid
, location geometry, comment text) distributed by (key);
CREATE TABLE
```

当不指定分布键时（即不带后面的`distributed by (key)`时），Greenplum 会默认对 id 字段以 round-robin 的方式进行随机分配。

关于分布键

分布键对于查询性能至关重要。“均匀”为分布键选择的第一大原则，选取更有业务意义的字段，将有助于显著提高性能。

- 尽量选择分布均匀的列、或者多列，以防止数据倾斜。
- 尽量选择常用于连接运算的字段，对于并发较高的语句，该选择更为重要。
- 尽量选择高并发查询、过滤性高的条件列。
- 尽量不要轻易使用随机分布。

详细内容，请参见本文档参考中内容。

构造数据

1. 创建函数，用于生成随机字符串。

```
CREATE OR REPLACE FUNCTION random_string(integer) RETURNS text AS $
body$
SELECT array_to_string(array
                       (SELECT substring('0123456789ABCDEFGHIJ
KLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
                                         FROM (ceil(random()*62))::
int
                                         FOR 1)
                        FROM generate_series(1, $1)), '');
$body$
LANGUAGE SQL VOLATILE;
```

2. 创建分布键。

```
CREATE TABLE tbl(id serial, KEY integer, locate geometry, COMMENT
text) distributed by (key);
```

3. 构造数据。

```
INSERT INTO tbl(KEY, COMMENT, locate)
 SELECT
     KEY,
     COMMENT,
     ST_GeomFromText(locate) AS locate
 FROM
     (SELECT
         (a + 1) AS KEY,
         random_string(ceil(random() * 24)::integer) AS COMMENT,
         'POINT(' || ceil(random() * 36 + 99) || ' ' || ceil(random
() * 24 + 50) || ')' AS locate
     FROM
         generate_series(0, 99999) AS a)
     AS t;
```

查询示例

- 查询

```
  => select * from tbl where key = 751;
  | id  | key | value | shape        | locate
              | comment        |
  +-----+-----+-------+--------------
+-----------------------------------------------+-----------------+
  | 751 | 751 | red   | 0101000000000000000C05B400000000000004A40
 | B9hPhjeNWPqV |
  (1 row)
  Time: 513.101 ms
```

- 查询计划

```
  => explain select * from tbl where key = 751;
  Gather Motion 1:1  (slice1; segments: 1)  (cost=0.00..1519.28 rows
=1 width=53)
    ->  Seq Scan on tbl  (cost=0.00..1519.28 rows=1 width=53)
          Filter: key = 751
```

```
  Settings:  effective_cache_size=8GB; gp_statistics_use_fkeys=on
  Optimizer status: legacy query optimizer
```

参考

- *Pivotal Greenplum* **官方文档**
- *GP 4.3 Best Practice*
- *Greenplum* **数据分布黄金法则**

2 管理实例

2.1 创建实例

具体操作请参考创建实例。

2.2 申请外网地址

如果您的应用部署在与您的HybridDB for PostgreSQL实例在同一地域且网络类型相同的ECS上，则无需申请外网地址。如果您的应用部署在与您的HybridDB for PostgreSQL实例在不同地域或网络类型不同的ECS或者阿里云以外的系统上，需申请外网地址，用于对接应用。

说明:

只要在同一地域内（可用区可以不同）且网络类型相同的实例，就可以通过内网互通。

应用场景

内外网地址的使用场景如下所示：

- 单独使用内网地址：
  - 适用于应用部署在与您的HybridDB for PostgreSQL实例在同一地域的ECS上且实例与ECS的网络类型相同时。
- 单独使用外网地址：
  - 适用于应用部署在与您的HybridDB for PostgreSQL实例在不同地域的ECS上时。
  - 适用于应用部署在阿里云以外的系统上时。
- 同时使用内外网地址：
  - 适用于应用中有些模块部署在与您的HybridDB for PostgreSQL实例同一地域且网络类型相同的ECS上，有些模块又部署在与您的HybridDB for PostgreSQL实例在不同地域的ECS上时。
  - 适用于应用中有些模块部署在与您的HybridDB for PostgreSQL实例在同一地域且网络类型相同的ECS上，有些模块又部署在阿里云以外的系统上时。

注意事项

- 在访问数据库前，您需要将访问数据库的IP地址或者IP段加入白名单，操作请参见设置白名单。

- 使用外网地址会降低实例的安全性，请谨慎选择。为了获得更快的传输速率和更高的安全级别，建议您将应用迁移到与您的HybridDB for PostgreSQL在同一地域的ECS上。

操作步骤

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 在实例列表上方选择实例所在地域。
3. 找到目标实例，在操作栏中，单击管理。
4. 在基本信息页面，单击申请外网地址，跳转到数据库连接页面。也可直接单击左侧导航的数据库连接。
5. 在数据库连接页面，单击申请外网地址。
6. 在弹出的对话框中，单击确定，生成外网地址。

生成外网地址后，可在数据库连接页面，通过释放外网地址按键释放外网地址。

相关API

API	描述
#unique_11	申请外网地址

2.3 释放外网地址

申请外网地址之后，如果网络环境发生变化，用户不再需要用外网地址连接实例，可在HybridDB for PostgreSQL控制台释放外网地址。释放外网地址之后，请注意更改涉及该地址的应用设置。

在释放外网地址之前，请先阅读以下应用场景。

应用场景

内外网地址的使用场景如下所示：

- 单独使用内网地址：
  - 适用于应用部署在与您的HybridDB for PostgreSQL实例在同一地域的ECS上且实例与ECS的网络类型相同时。
- 单独使用外网地址：
  - 适用于应用部署在与您的HybridDB for PostgreSQL实例在不同地域的ECS上时。
  - 适用于应用部署在阿里云以外的系统上时。

- 同时使用内外网地址：
  - 适用于应用中有些模块部署在与您的HybridDB for PostgreSQL实例同一地域且网络类型相同的ECS上，有些模块又部署在与您的HybridDB for PostgreSQL实例在不同地域的ECS上时。
  - 适用于应用中有些模块部署在与您的HybridDB for PostgreSQL实例在同一地域且网络类型相同的ECS上，有些模块又部署在阿里云以外的系统上时。

操作步骤

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 在实例列表上方选择实例所在地域。
3. 找到目标实例，在操作栏中，单击管理。
4. 单击导航栏左侧的数据库连接。
5. 在数据库连接页面，单击释放外网地址。

   如未申请外网地址，数据库连接页面仅有申请外网地址按键。
6. 在弹出的对话框中，单击确定，释放外网地址。

相关API

API	描述
#unique_13	释放实例的外网连接串

2.4 升级实例规格

在使用云数据库 HybridDB for PostgreSQL 过程中，随着您的数据量和计算量的动态增长，一些计算资源如CPU、磁盘、内存以及数据处理节点数量将成为数据处理速度的瓶颈。为了支持实例的动态扩展，HybridDB for PostgreSQL 提供在线升级实例规格的功能，目前暂不支持对实例的降级操作。本文介绍了升级实例规格的相关操作。

查看当前实例规格

HybridDB for PostgreSQL 实例规格包括 计算组规格 和计算组数量，各种计算组规格详细说明见实例规格 。

您可以通过以下的步骤来查看当前的实例规格：

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 选择要操作实例所在的地域。例如，华东1。

3. 单击目标实例右侧操作栏中的管理按钮，进入基本信息页面。

   在基本信息页面的配置信息中显示了当前的实例规格，包括计算组规格、计算组详情、计算组数量、计算资源汇总。

HybridDB for PostgreSQL 目前提供两类规格：

- 高性能，规格名称以gpdb.group.segsdx开始，提供更好的I/O能力，带来更高的性能。
- 高容量，规格名称以gpdb.group.seghdx开始，提供更大、更实惠的空间，满足更高的存储需要。

升级实例规格

当确认需要升级实例规格后，您可以按照下面的步骤进行实例规格的升级：

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 选择要操作实例所在的地域。例如，华东1。
3. 单击目标实例右侧操作栏中的升级按钮，进入确认订单页面。
4. 选择合适的计算组规格和计算组数量，点击去开通按钮。

   目前，HybridDB for PostgreSQL 支持不同的计算组规格和计算组数量组合，详情参见规则组合速查表。在选择新的计算组规格和计算组数量组合时，要满足以下约束：

   - 新旧计算组规格必须为同一类计算组规格，而且新的计算组规格 >= 旧的计算组规格
   - 如果新的计算组规格 = 旧的计算组规格，新的计算组数量 > 旧的计算组数量

   在选择计算组规格和计算组数量的时候，除了满足以上约束，您还需要对自己的数据量和计算量进行评估，从而选择合适的计算组规格和计算组数量组合，详情可参考如何选择实例规格 。

   注意：
   根据您数据量的不同，实例规格升级的过程大约需要30分钟到3个小时的时间。在此过程中，为了保证数据的一致性，您实例只对外提供只读服务，并且会闪断两次，请您提前做出调整。当升级实例规格结束，对应实例恢复运行中状态，您可以正常访问数据库，而且实例的数据库内核版本自动升级为最新。

在完成了上述操作之后，您可以回到控制台，查看目标实例的运行状态。如果实例规格升级操作完成，则实例运行状态为运行中，否则为升降级中。

2.5 修改连接地址

HybridDB for PostgreSQL支持修改实例连接地址。例如，您改用其他HybridDB for PostgreSQL实例后，无需对应用程序进行修改，只需将新实例的连接地址修改为原实例的连接地址。

前提条件

- 实例正常运行。
- 实例已设置了白名单。

操作步骤

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 在实例列表上方选择实例所在地域。
3. 找到目标实例，在操作栏中，单击管理。
4. 单击左侧导航的数据库连接。
5. 在数据库连接页面，单击修改连接地址。
6. 在弹出的对话框中，单击连接类型下拉框，选择网络类型。

   可选的网络类型有内网地址和外网地址，其中只有在申请了外网地址后，才有外网地址这个选项。
7. 在随后出现的连接地址和端口输入框中，填入相关信息后，单击确定。

   设置成功后刷新页面，页面显示新的连接地址。

相关API

API	描述
#unique_19	修改实例连接地址和端口

2.6 按量付费转包年包月

购买按量付费的实例后，您可以根据需求将实例改变成包年包月的计费方式。

注意事项

- 包年包月的实例无法转成按量付费的实例，在您进行计费方式的转变前请务必考虑清楚，以免造成资源浪费。
- 在计费周期内，包年包月的实例只支持升级配置，不支持降级配置或者释放。
- 变更实例计费方式成功后，实例会即刻按照包年包月的实例计费。

- 按量付费实例变更为包年包月时会产生一个新购订单，您必须完成该订单的支付流程，计费方式的变更才能生效。若未支付或未成功支付，您的订单管理页面将会出现未完成订单，之后您将无法新购实例或变更实例计费类型。

变更条件

- 实例计费类型为按量付费，且实例状态为运行中。
- 实例没有未完成的变更计费方式（即新购）的订单。

操作步骤

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 找到目标实例，在操作栏中，单击按量转包年包月。
3. 在确认订单页面，选择购买时长，勾选《HybridDB for PostgreSQL服务协议》，单击去开通。
4. 在订单信息页面，确认订单信息，单击确认支付。

> 说明:
> 此时系统会生成一个转包年包月的订单。若该订单未支付或作废，将导致您无法进行新购实例或转包年包月的操作。您可以在订单管理页面支付或作废该订单。

2.7 重启实例

为了更好地满足用户的需求，HybridDB for PostgreSQL 会不断更新数据库内核版本。在实例创建时，默认使用的是最新版本的数据库内核。当新的版本发布之后，您可以通过重启实例来更新数据库内核版本，从而使用新版本中扩展的功能。本文介绍了重启实例的方法。

> 注意:
> 重启实例时，实例将不可用，请在业务低谷时执行该操作。

操作步骤

您可以按照下面的步骤来重启实例：

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 选择要操作实例所在的地域。例如，华东1。
3. 单击目标实例右侧操作栏中的管理 按钮，进入基本信息页面。

4. 单击页面右上角的重启实例，并在确认框中单击确定。如果您绑定了手机，还需要进行手机验证码验证。

> **注意:**
>
> 重启过程一般耗时3到30分钟，在此过程中该实例不能对外提供服务，请您提前做出调整。当实例重启结束，对应实例恢复运行中状态，您可以正常访问数据库。

完成了上述操作后，您可以回到控制台查看目标实例的运行状态。如果实例重启操作完成，则实例运行状态为运行中，否则为重启中。

相关API

API	描述
#unique_22	重启实例

2.8 开启SQL审计

HybridDB for PostgreSQL控制台提供SQL审计功能，您可通过该功能查看SQL明细、定期审计SQL。开通SQL审计功能后，实例性能不会受到影响。

背景信息

SQL审计会统计所有DML和DDL操作信息，这些信息是系统通过网络协议分析所得，在SQL查询量较大的时候可能会丢失少量记录。

注意事项

- SQL审计记录的保存时间为30天。
- 实例默认关闭SQL审计功能。开启该功能后，实例会产生额外费用，详细收费标准请参见云数据库*HybridDB for PostgreSQL* 详细价格信息。

操作步骤

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 在实例列表上方选择实例所在地域。
3. 找到目标实例，在操作栏中，单击管理。
4. 单击左侧导航的数据安全性。
5. 选择SQL审计页签，单击开启SQL审计。

6. 在弹出的对话框中单击确定，开启SQL审计功能。

   - 在开始SQL审计功能后，您可通过时间、DB、User、关键字等条件查询SQL信息。
   - 您还可在SQL审计页签，通过关闭SQL审计按键关闭已经开通的SQL审计。

相关API

API	描述
#unique_24	开启或关闭指定实例的SQL采集功能

2.9 释放实例

根据业务要求，您可以手动释放按量付费的实例。

前提条件

实例类型为按量付费实例。

> 说明:
> 您可以随时释放按量付费的实例；包年包月的实例不能主动删除或释放，到期后自动释放。

操作步骤

1. 登录云数据库*HybridDB for PostgreSQL*管理控制台。
2. 在实例列表上方选择实例所在地域。
3. 找到目标实例，在操作栏中，单击管理。
4. 在基本信息页面，单击运行状态右侧的释放按键。
5. 在弹出的对话框中勾选您确定释放实例？，然后单击确定，释放实例。

> 注意:
> 释放后的实例不可恢复，请谨慎操作。

相关API

API	描述
#unique_26	释放实例

3 数据迁移

数据迁移相关的操作请参见数据迁移方案概览。

4 用户权限管理

用户管理

实例创建过程中，会提示用户指定初始用户名和密码，这个初始用户为“根用户”。实例创建好后，用户可以使用该根用户连接数据库。使用*psql*（PostgreSQL 或 Greenplum 的客户端工具）连接数据库后，通过`\du+`命令可以查看所有用户的信息，示例如下：

注意:

除了根用户外，还有其他内部管理用户被创建。

```
postgres=> \du+
                                   List of roles
  Role name   |            Attributes             | Member of |
Description
--------------+-----------------------------------+-----------
+---------------
 root_user    |                                   |           |
rds_superuser
 ...
```

目前，HybridDB for PostgreSQL 没有开放 SUPERUSER 权限，对应的是 RDS_SUPERUSER，这一点与云数据库 RDS（PostgreSQL）中的权限体系一致。所以，根用户（如上面的示例中的 root_user）具有 RDS_SUPERUSER 权限，这个权限属性只能通过查看用户的描述（Description）来识别。根用户具有如下权限：

- 具有 CREATEROLE、CREATEDB 和 LOGIN 权限，即可以用来创建数据库和用户，但没有 SUPERUSER 权限。
- 查看和修改其它非超级用户的数据表，执行 SELECT、UPDATE、DELTE 或更改所有者（Owner）等操作。
- 查看其它非超级用户的连接信息、撤销（Cancel）其 SQL 或终止（Kill）其连接。
- 执行 CREATE EXTENSION 和 DROP EXTENSION 命令，创建和删除插件。
- 创建其他具有 RDS_SUPERUSER 权限的用户，示例如下：

```
CRATE ROLE root_user2 RDS_SUPERUSER LOGIN PASSWORD 'xyz';
```

权限管理

用户可以在数据库（Database）、模式（Schema）、表等多个层次管理权限。例如，赋予一个用户表上的读取权限，但收回修改权限，示例如下。

```
GRANT SELECT ON TABLE t1 TO normal_user1;
REVOKE UPDATE ON TABLE t1 FROM normal_user1;
```

```
REVOKE DELETE ON TABLE t1 FROM normal_user1;
```

参考文档

关于具体的用户与权限管理方法，请参见*Managing Roles and Privileges*。

5 插件管理

云数据库 HybridDB 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库，由多个计算组组成，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 HybridDB for PostgreSQL 支持如下插件：

- PostGIS：支持地理信息数据。
- MADlib：机器学习方面的函数库。
- fuzzystrmatch：字符串模糊匹配。
- orafunc：兼容 Oracle 的部分函数。
- oss_ext：支持从 OSS 读取数据。
- hll：支持用 HyperLogLog 算法进行统计。
- pljava：支持使用 PL/Java 语言编写用户自定义函数（UDF）。
- pgcrypto：支持加密函数。
- intarray：整数数组相关的函数、操作符和索引支持。

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;
```

6 JSON 数据类型操作

JSON 类型几乎已成为互联网及物联网（IoT）的基础数据类型，其重要性不言而喻，具体协议请参见 *JSON* 官网。

PostgreSQL 对 JSON 的支持已经比较完善，阿里云深度优化云数据库 HybridDB for PostgreSQL，基于 PostgreSQL 语法进行了 JSON 数据类型的支持。

检查现有版本是否支持 JSON

执行如下命令，检查是否已经支持 JSON：

```
=> SELECT '""'::json;
```

若系统出现如下信息，则说明已经支持 JSON 类型，可以使用实例了。若执行不成功，请重新启动实例。

```
 json
------
 ""
(1 row)
```

若系统出现如下信息，则说明尚未支持 JSON 类型。

```
ERROR:  type "json" does not exist
LINE 1: SELECT '""'::json;
                    ^
```

上述命令是一次从字符串到 JSON 格式的强制转换，这也基本上是其操作上的本质。

JSON 在数据库中的转换

数据库的操作主要分为读和写，JSON 的数据写入一般是字符串到 JSON。字符串中的内容必须符合 JSON 标准，包括字符串、数字、数组、对象等内容。如：

字符串

```
=> SELECT '"hijson"'::json;
 json
-------
 "hijson"
(1 row)
```

`::`在 PostgreSQL/Greenplum/HybridDB for PostgreSQL 中代表强制类型转换。在此转换的时候，会调用 JSON 类型的输入函数。因此，类型转换时同样会做 JSON 格式的检查，如下所示：

```
=> SELECT '{hijson:1024}'::json;
ERROR:  invalid input syntax for type json
LINE 1: SELECT '{hijson:1024}'::json;
               ^
DETAIL:  Token "hijson" is invalid.
```

```
CONTEXT:  JSON data, line 1: {hijson...
=>
```

上述"hijson"两边的"是必不可少的，因为在标准中，KEY 值对应的是一个字符串，所以这里的{hijson:1024}在语法上会报错。

除了类型上的强制转换，还有从数据库记录到 JSON 串的转换。

我们正常使用 JSON，不会只用一个 String 或一个 Number，而是一个包含一个或多个键值对的对象。所以，对 Greenplum 而言，支持到对象的转换，即支持了 JSON 的绝大多数场景，如：

```
=> select row_to_json(row('{"a":"a"}', 'b'));
           row_to_json
---------------------------------
 {"f1":"{\"a\":\"a\"}","f2":"b"}
(1 row)
=> select row_to_json(row('{"a":"a"}'::json, 'b'));
        row_to_json
---------------------------
 {"f1":{"a":"a"},"f2":"b"}
(1 row)
```

由此也可以看出字符串和 JSON 的区别，这样就可以很方便地将一整条记录转换成 JSON。

JSON 内部数据类型的定义

- 对象

  对象是 JSON 中最常用的，如：

  ```
  => select '{"key":"value"}'::json;
        json
  -----------------
   {"key":"value"}
  (1 row)
  ```

- 整数 & 浮点数

  JSON 的协议只有三种数字：整数、浮点数和常数表达式，当前 Greenplum 对这三种都有很好的支持。

  ```
  => SELECT '1024'::json;
   json
  ------
   1024
  (1 row)
  => SELECT '0.1'::json;
   json
  ------
   0.1
  (1 row)
  ```

  特殊情况下，需要如下信息：

  ```
  => SELECT '1e100'::json;
  ```

```
 json
-------
 1e100
(1 row)
=> SELECT '{"f":1e100}'::json;
    json
-------------
 {"f":1e100}
(1 row)
```

并且，包括下面这个长度超长的数字：

```
=> SELECT '9223372036854775808'::json;
        json
---------------------
 9223372036854775808
(1 row)
```

- **数组**

```
=> SELECT '[[1,2], [3,4,5]]'::json;
       json
------------------
 [[1,2], [3,4,5]]
(1 row)
```

操作符

JSON 支持的操作符类型

```
=> select oprname,oprcode from pg_operator where oprleft = 3114;
 oprname |          oprcode
---------+---------------------------
 ->      | json_object_field
 ->>     | json_object_field_text
 ->      | json_array_element
 ->>     | json_array_element_text
 #>      | json_extract_path_op
 #>>     | json_extract_path_text_op
(6 rows)
```

基本使用方法

```
=> SELECT '{"f":"1e100"}'::json -> 'f';
 ?column?
----------
 "1e100"
(1 row)
=> SELECT '{"f":"1e100"}'::json ->> 'f';
 ?column?
----------
 1e100
(1 row)
=> select '{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}'::json#>array
['f4','f6'];
 ?column?
-----------
 "stringy"
(1 row)
```

```
=> select '{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}'::json#>'{f4,
f6}';
 ?column?
-----------
 "stringy"
(1 row)
=> select '{"f2":["f3",1],"f4":{"f5":99,"f6":"stringy"}}'::json#>>'{f2
,0}';
 ?column?
----------
 f3
(1 row)
```

JSON 函数

支持的函数

```
postgres=# \df *json*
                                                        List of
functions
   Schema   |           Name            | Result data type |
         Argument data types                    |  Type
------------+---------------------------+------------------
+-------------------------------------------------------------+--------
 pg_catalog | array_to_json             | json             | anyarray
                                                 | normal
 pg_catalog | array_to_json             | json             | anyarray
, boolean                                         | normal
 pg_catalog | json_array_element        | json             | from_json
 json, element_index integer                     | normal
 pg_catalog | json_array_element_text   | text             | from_json
 json, element_index integer                     | normal
 pg_catalog | json_array_elements       | SETOF json       | from_json
 json, OUT value json                            | normal
 pg_catalog | json_array_length         | integer          | json
                                                 | normal
 pg_catalog | json_each                 | SETOF record     | from_json
 json, OUT key text, OUT value json              | normal
 pg_catalog | json_each_text            | SETOF record     | from_json
 json, OUT key text, OUT value text              | normal
 pg_catalog | json_extract_path         | json             | from_json
 json, VARIADIC path_elems text[]                | normal
 pg_catalog | json_extract_path_op      | json             | from_json
 json, path_elems text[]                         | normal
 pg_catalog | json_extract_path_text    | text             | from_json
 json, VARIADIC path_elems text[]                | normal
 pg_catalog | json_extract_path_text_op | text             | from_json
 json, path_elems text[]                         | normal
 pg_catalog | json_in                   | json             | cstring
                                                 | normal
 pg_catalog | json_object_field         | json             | from_json
 json, field_name text                           | normal
 pg_catalog | json_object_field_text    | text             | from_json
 json, field_name text                           | normal
 pg_catalog | json_object_keys          | SETOF text       | json
                                                 | normal
 pg_catalog | json_out                  | cstring          | json
                                                 | normal
 pg_catalog | json_populate_record      | anyelement       | base
anyelement, from_json json, use_json_as_text boolean | normal
 pg_catalog | json_populate_recordset   | SETOF anyelement | base
anyelement, from_json json, use_json_as_text boolean | normal
```

```
 pg_catalog | json_recv                       | json                 | internal
                                       | normal
 pg_catalog | json_send                       | bytea                | json
                                       | normal
 pg_catalog | row_to_json                     | json                 | record
                                       | normal
 pg_catalog | row_to_json                     | json                 | record,
boolean                                 | normal
 pg_catalog | to_json                         | json                 |
anyelement                                       | normal
(24 rows)
```

基本使用方法

```
=> SELECT array_to_json('{{1,5},{99,100}}'::int[]);
  array_to_json
------------------
 [[1,5],[99,100]]
(1 row)
=> SELECT row_to_json(row(1,'foo'));
     row_to_json
---------------------
 {"f1":1,"f2":"foo"}
(1 row)
=> SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]');
 json_array_length
-------------------
                 5
(1 row)
=> select * from json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null,"f5
":99,"f6":"stringy"}') q;
 key |   value
-----+-----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  | null
 f5  | 99
 f6  | "stringy"
(5 rows)
=> select json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null,"f5":"
null"}');
  json_each_text
-------------------
 (f1,"[1,2,3]")
 (f2,"{""f3"":1}")
 (f4,)
 (f5,null)
(4 rows)
=> select json_array_elements('[1,true,[1,[2,3]],null,{"f1":1,"f2":[7,
8,9]},false]');
  json_array_elements
-----------------------
 1
 true
 [1,[2,3]]
 null
 {"f1":1,"f2":[7,8,9]}
 false
(6 rows)
create type jpop as (a text, b int, c timestamp);
=> select * from json_populate_record(null::jpop,'{"a":"blurfl","x":43
.2}', false) q;
   a    | b | c
```

```
--------+---+---
 blurfl |   |
(1 row)
=>     select * from json_populate_recordset(null::jpop,'[{"a":"blurfl
","x":43.2},{"b":3,"c":"2012-01-20 10:42:53"}]',false) q;
   a    | b |            c
--------+---+--------------------------
 blurfl |   |
        | 3 | Fri Jan 20 10:42:53 2012
(2 rows)
```

完整操作事例

创建表

```
create table tj(id serial, ary int[], obj json, num integer);
=> insert into tj(ary, obj, num) values('{1,5}'::int[], '{"obj":1}', 5
);
INSERT 0 1
=> select row_to_json(q) from (select id, ary, obj, num from tj) as q;
                row_to_json
-------------------------------------------
 {"f1":1,"f2":[1,5],"f3":{"obj":1},"f4":5}
(1 row)
=> insert into tj(ary, obj, num) values('{2,5}'::int[], '{"obj":2}', 5
);
INSERT 0 1
=> select row_to_json(q) from (select id, ary, obj, num from tj) as q;
                row_to_json
-------------------------------------------
 {"f1":1,"f2":[1,5],"f3":{"obj":1},"f4":5}
 {"f1":2,"f2":[2,5],"f3":{"obj":2},"f4":5}
(2 rows)
```

多表 JOIN

```
create table tj2(id serial, ary int[], obj json, num integer);
=> insert into tj2(ary, obj, num) values('{2,5}'::int[], '{"obj":2}',
5);
INSERT 0 1
=> select * from tj, tj2 where tj.obj->>'obj' = tj2.obj->>'obj';
 id |  ary  |    obj    | num | id |  ary  |    obj    | num
----+-------+-----------+-----+----+-------+-----------+-----
  2 | {2,5} | {"obj":2} |   5 |  1 | {2,5} | {"obj":2} |   5
(1 row)
=> select * from tj, tj2 where json_object_field_text(tj.obj, 'obj')
 = json_object_field_text(tj2.obj, 'obj');
 id |  ary  |    obj    | num | id |  ary  |    obj    | num
----+-------+-----------+-----+----+-------+-----------+-----
  2 | {2,5} | {"obj":2} |   5 |  1 | {2,5} | {"obj":2} |   5
(1 row)
```

JSON 函数索引

```
CREATE TEMP TABLE test_json (
       json_type text,
       obj json
);
=> insert into test_json values('aa', '{"f2":{"f3":1},"f4":{"f5":99,"
f6":"foo"}}');
```

```
INSERT 0 1
=> insert into test_json values('cc', '{"f7":{"f3":1},"f8":{"f5":99,"
f6":"foo"}}');
INSERT 0 1
=> select obj->'f2' from test_json where json_type = 'aa';
 ?column?
----------
 {"f3":1}
(1 row)
=> create index i on test_json (json_extract_path_text(obj, '{f4}'));
CREATE INDEX
=> select * from test_json where json_extract_path_text(obj, '{f4}') =
 '{"f5":99,"f6":"foo"}';
 json_type |                  obj
-----------+-----------------------------------------
 aa        | {"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}
(1 row)
```

注意：JSON 类型暂时不能支持作为分布键来使用；也不支持 JSON 聚合函数。

下面是 Python 访问的一个例子：

```
#! /bin/env python
import time
import json
import psycopg2
def gpquery(sql):
    conn = None
    try:
        conn = psycopg2.connect("dbname=sanity1x2")
        conn.autocommit = True
        cur = conn.cursor()
        cur.execute(sql)
        return cur.fetchall()
    except Exception as e:
        if conn:
            try:
                conn.close()
            except:
                pass
            time.sleep(10)
        print e
    return None
def main():
    sql = "select obj from tj;"
    #rows = Connection(host, port, user, pwd, dbname).query(sql)
    rows = gpquery(sql)
    for row in rows:
        print json.loads(row[0])
if __name__ == "__main__":
    main()
```

7 HyperLogLog 的使用

阿里云深度优化云数据库 HybridDB 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_i
nteger(user_id)) from generate_series(1,10000) t(user_id);
insert into access_date select current_date-1, hll_add_agg(hll_hash_i
nteger(user_id)) from generate_series(5000,20000) t(user_id);
insert into access_date select current_date-2, hll_add_agg(hll_hash_i
nteger(user_id)) from generate_series(9000,40000) t(user_id);
postgres=# select #userids from access_date where acc_date=current_da
te;
     ?column?
------------------
 9725.85273370708
(1 row)
postgres=# select #userids from access_date where acc_date=current_da
te-1;
     ?column?
------------------
 14968.6596883279
(1 row)
postgres=# select #userids from access_date where acc_date=current_da
te-2;
     ?column?
------------------
 29361.5209149911
(1 row)
```

8 使用 Create Library 命令

为支持用户导入自定义软件包，HybridDB 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
  ```

- `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：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 3：删除一个库。**

```
drop library example;
```

- **示例 4：查看已经安装的库。**

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

9 使用 PL ／ Java UDF

HybridDB 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 HybridDB 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 HybridDB for PostgreSQL 中，执行 Create Library 命令，将文件导入到 HybridDB 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 HybridDB 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;
```

10 使用排序键（SortKey）

排序键（SortKey）是表的一种属性，可以将数据按照排序键顺序存储在磁盘文件中。使用排序键的优势主要有：

- 加速列存优化，收集的min、max元信息很少有重叠，过滤性好。
- 避免对含有order by和group by等需要排序的SQL数据再次排序，从磁盘中直接读取出来的数据就是满足条件的有序数据。

本文介绍了排序键在不同场景下的用法。

创建表时定义排序键

使用命令CREATE TABLE定义一个新的表。语法如下：

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (
[ { column_name data_type [ DEFAULT default_expr ]     [column_con
straint [ ... ]
[ ENCODING ( storage_directive [,...] ) ]
]
   | table_constraint
   | LIKE other_table [{INCLUDING | EXCLUDING}
                      {DEFAULTS | CONSTRAINTS}] ...}
   [, ... ] ]
   [column_reference_storage_directive [, ] ]
   )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH ( storage_parameter=value [, ... ] )
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
   [ SORTKEY (column, [ ... ] )]
   [ PARTITION BY partition_type (column)
       [ SUBPARTITION BY partition_type (column) ]
          [ SUBPARTITION TEMPLATE ( template_spec ) ]
       [...]
    ( partition_spec )
        | [ SUBPARTITION BY partition_type (column) ]
          [...]
    ( partition_spec
      [ ( subpartition_spec
           [(...)]
       ) ]
```

```
    )
```

样例

```
create table test(date text, time text, open float, high float, low
float, volume int) with(APPENDONLY=true,ORIENTATION=column) sortkey (
volume);
```

对表进行排序

命令如下：

```
VACUUM SORT ONLY [tablename]
```

修改排序键

命令如下：

```
ALTER [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name SET
SORTKEY (column, [ ... ] )
```

注意：

这个命令只会修改catalog，不会对数据立即排序，需要使用`VACUUM SORT ONLY`命令排序。

样例

```
alter table test set sortkey (high,low);
```

注意事项

一旦对表进行了更新（例如Insert、Update、Delete），表的数据将被视为未按排序键排序，查询中不会将直接从磁盘读出的数据视为有序数据。此时，需要重新执行`VACUUM SORT ONLY`命令重新整理表数据。