ALIBABA CLOUD

阿里云

云数据库RDS

AliPG 内核

文档版本：20201010

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.AliPG 优势

本文介绍AliPG的背景信息和优势。

背景信息

PostgreSQL（简称PG）是一款全球流行的企业级开源数据库，2017、2018连续两年蝉联DB-Engine年度数据库，2019年荣获OSCON（The O'Reilly Open Source Convention）开源软件终身成就奖。

阿里云支持一系列兼容PostgreSQL的云数据库服务产品，目前包括RDS PostgreSQL和专属集群MyBase for PostgreSQL，这些云数据库服务采用统一的数据库内核（简称AliPG），AliPG兼容PostgreSQL开源数据库，于2015年正式商用，目前支持9.4、10、11、12等PostgreSQL大版本，已稳定运行多年，支撑了大量阿里巴巴集团内部以及云上的客户业务。

采用AliPG的阿里云数据库产品

- RDS PostgreSQL
- 专属集群MyBase for PostgreSQL

支持的版本

- 12
- 11
- 10
- 9.4

优势

AliPG的研发理念：提前洞察行业需求，帮助客户拓展业务边界。

相比开源版本PostgreSQL，AliPG具有如下特点：

- 更快速度
  - 图像识别、向量相似搜索场景，相比通用解决方案提升上万倍性能。详情请参见图像识别、人脸识别、相似特征检索、相似人群圈选。
  - 实时营销、用户画像，相比通用解决方案提升上千倍性能。详情请参见实时精准营销（人群圈选）。
  - GIS MOD移动对象处理，相比开源PostGIS，性能提升50倍。详情请参见时空数据库简介。
- 更稳定性能

  针对性优化平台即服务（PaaS）的多租户（schema）场景，帮助传统软件实现从售卖License到售卖订阅服务的转型，支持大量元数据，优化连接、优化资源隔离，单个实例可支持上万租户。
- 更高安全性
  - 通过中国、国际安全标准认证，助力企业提升在融资、上市阶段的机构安全评分。
  - 安全加固：
    - 对动态视图、共享内存、dblink、历史命令、审计日志等包含密码的敏感信息进行加密。
    - 修补社区版本函数问题。
    - 支持全加密云数据库。
    - 新增半同步模式，支持自主配置最大保护、最高可用、最高性能等实例保护级别。
    - 支持逻辑订阅故障转移（Failover Slot），在使用逻辑复制功能时，主备切换不影响逻辑复制可靠性。

- 更灵活可控（专属集群MyBase for PostgreSQL）
  - 开放OS权限，用户对集群自主可控。
  - 自定义超配比，对于开发、测试、预发环境可以使用高超配比，例如64核的主机可以超配到128核，而在生产系统中使用独享资源，从而降低整体成本。

2.AliPG 小版本 Release Nostes

本文介绍AliPG的内核版本更新说明。

PostgreSQL 12

20200830

- 新特性
  - ganos插件升级到3.0版本。
  - 支持SQL防火墙插件，用于防止恶意SQL注入。
  - 支持bigm插件，用于模糊查询。
  - 支持TimescaleDB插件，用于时序数据处理。
- Bug修复
  - 修复一个 `rds_` 前缀后台参数不识别的错误。
  - 修复pg_cron插件无法正常创建使用的错误。
  - 打包了RDKit插件的若干依赖，修复缺少依赖无法载入的错误。

20200421

- 新特性
  - 内核升级到兼容社区12.2版本。
  - ganos插件升级到2.7版本。
  - 新增hll插件2.14版本。
  - 新增plproxy插件2.9.0版本。
  - 新增tsm_system_rows插件1.0版本。
  - 新增tsm_system_time插件1.0版本。
  - 新增smlar插件1.0版本。
  - 新增tds_fdw插件1.0版本。
- Bug修复

  修复逻辑订阅超时导致的实例重启问题。

20200221

- 新特性
  - 支持为rds_superuser角色预留指定连接数，以便连接数不足时可以连接实例排查故障。
  - 支持wal2json插件。
  - 升级ganos插件到2.6版本。
- Bug修复

  修复部分权限问题。

20191230

新特性

- 支持pg_roaringbitmap、rdkit、mysql_fdw、ganos插件。
- 支持用户高权限账号一次发布所有表和创建订阅。

PostgreSQL 11

20200830

- 新特性
  - ganos插件升级到3.0版本。
  - 支持SQL防火墙插件，用于防止恶意SQL注入。
  - 支持bigm插件，用于模糊查询。
  - 支持ZomboDB插件，用于文本搜索和分析。
- Bug修复
  - 修复一个 `rds_` 前缀后台参数不识别的错误。
  - 修复Failover Slot同名流复制槽导致备库崩溃的错误
  - 修复pg_cron插件无法正常创建使用的错误。
  - 修复PASE插件一个全局变量引起的正确性错误。

20200610

新特性

- 支持timescaledb插件1.7.1版本。
- 支持rds_superuser使用插件pageinspect。
- 支持rds_superuser赋予其他用户replication权限。

20200511

- 新特性

  ganos插件升级到2.8版本。
- Bug修复

  PASE插件解决HNSW索引执行insert操作慢的问题。

20200421

新特性

- 支持逻辑订阅故障转移（Failover Slot）功能。
- 新增plproxy插件2.9.0版本。
- 新增tsm_system_rows插件1.0版本。
- 新增tsm_system_time插件1.0版本。
- 新增smlar插件1.0版本。

20200402

- 新特性
  - 支持hll插件（2.14版本），增hll数据类型，毫秒级别响应，近似数据分析业务，例如实时PV、UV查询，或者特征标签contain判定，低成本、高效率，解决近似分析的问题。
  - 支持oss_fdw插件（1.1版本），可用于数据库OSS扩展存储，您可以将查询量少的历史数据存储在OSS，降低存储成本，同时满足低频的查询需求。
  - 支持tds_fdw插件（2.0.1版本），不需要etl程序，即可在PostgreSQL数据库中实时查询Sybase、SQL Server数据，满足多数据库种类跨库查询、迁移数据的需求。
- 插件升级

  - ganos升级到2.7版本。
  - wal2json升级2.2版本。
- 性能优化

  `shutdown -m fast` 命令优化。

20191218

新特性

- 支持PASE索引插件，提供图像识别索引。
- 支持用户高权限账号一次发布所有表和创建订阅。

PostgreSQL 10

20200830

- 新特性
  - ganos插件升级到3.0版本。
  - 支持SQL防火墙插件，用于防止恶意SQL注入。
  - 支持bigm插件，用于模糊查询。
- Bug修复

  修复pg_cron插件无法正常创建使用的错误。

20200212

- 新特性
  - 支持为rds_superuser角色预留指定连接数，以便连接数不足时可以连接实例排查故障。
  - 升级ganos插件到2.6版本。
- Bug修复

  修复流复制中等待时间长的问题。

20190703

- 新特性
  - 版本更新为10.9。
  - 支持在超时时将同步复制更改为异步复制。
- Bug修复
  - 修复make pg_hint_plan错误问题。
  - 修复外部rum失败问题。

PostgreSQL 9.4

20200623

- 新特性
  - wal2json升级2.2版本。
  - 支持xml2插件1.0版本。
- Bug修复

修复使用wal2json插件导致内存耗尽的问题。

20200210

支持为rds_superuser角色预留指定连接数，以便连接数不足时可以连接实例排查故障。

20190601

版本更新为9.4.19。

3.AliPG 功能模块

本文介绍AliPG特有的功能模块，包括高权限账号、时空数据库、读写外部数据、并发控制等。

功能模块介绍

类别	功能	描述
账号权限	rds_superuser	AliPG提供的rds_superuser是介于普通用户和superuser之间的一种用户，对应的账号称为高权限账号。由于云上环境的安全原因，AliPG不直接提供superuser，但是提供rds_superuser（主要裁剪了敏感安全权限）。rds_superuser用户可以创建和删除插件、创建和删除普通用户以及高权限账号、操作和访问所有普通用户的表、终止连接等。
时空数据库	Ganos时空引擎	阿里云自研Ganos时空引擎提供一系列的数据类型、函数和存储过程，用于对时间和空间数据进行高效存储、索引、查询和分析计算。
读写外部数据	oss_fdw	AliPG提供的oss_fdw插件可以将OSS中的数据加载到数据库中，也可以将数据库中的数据写入OSS中，为您提供数据迁移、冷热数据分离功能。
并发控制	pg_concurrency_control	AliPG提供的pg_concurrency_control插件可以控制事务执行、SQL查询、存储过程和DML操作的并发，您可以自定义大查询，pg_concurrency_control提供对大查询的并发控制功能，优化高并发下的执行性能，使得高并发业务性能更平滑。
逻辑订阅故障转移	Failover Slot	社区版PostgreSQL的Logical Slot在主备切换时会导致逻辑订阅断开，AliPG对此进行优化，可以将所有的Logical Slot从主实例同步到备实例，避免逻辑订阅断开。
位图功能扩展	varbitx	社区版PostgreSQL内置的varbit插件支持的bit类型操作函数比较简单，AliPG对其进行了扩展，支持更多的bit操作，可以覆盖更多的应用场景，例如实时用户画像推荐系统、门禁广告系统、购票系统等。
向量检索	PASE高效向量检索	PASE（PostgreSQL ANN search extension）是一款为AliPG数据库研发的高性能向量检索索引插件，使用业界中成熟稳定且高效的ANN（Approximate nearest neighbor）检索算法，包括IVFFlat和HNSW算法，通过这两种算法，可以在AliPG数据库中实现极高速向量查询。PASE暂时不支持特征向量的抽取与产出，您需要自行检索实体的特征向量，PASE负责的工作是根据已产出的海量级别的向量进行相似向量的检索。
日志查询	log_fdw	AliPG提供log_fdw插件，可以直接通过外部表查询到日志内容。
安全	安全加固	AliPG内置安全加固模块，完善自定义视图，增强函数安全，防止安全陷阱，规避社区安全漏洞。

相关文档

支持插件列表

4.支持插件列表

本文列出RDS PostgreSQL支持的插件及其版本。

> 说明 如果您对插件有需求或建议，请提交工单。

PostgreSQL 12

插件名称	插件版本
btree_gin	1.3
btree_gist	1.5
citext	1.6
cube	1.4
dblink	1.2
dict_int	1
earthdistance	1.1
fuzzystrmatch	1.1
hstore	1.6
intagg	1.1
intarray	1.2
isn	1.2
ltree	1.1
pg_buffercache	1.3
pg_prewarm	1.2
pg_stat_statements	1.7
pg_trgm	1.4
pgcrypto	1.3
pgrowlocks	1.2
pgstattuple	1.5
postgres_fdw	1

插件名称	插件版本
sslinfo	1.2
tablefunc	1
unaccent	1.1
plpgsql	1
plperl	1
pg_roaringbitmap	0.5.0
rdkit	3.8
mysql_fdw	1.1
ganos_geometry_sfcgal	3.0
ganos_geometry_topology	3.0
ganos_geometry	3.0
ganos_networking	3.0
ganos_pointcloud_geometry	3.0
ganos_pointcloud	3.0
ganos_raster	3.0
ganos_spatialref	3.0
ganos_trajectory	3.0
ganos_tiger_geocoder	3.0
ganos_address_standardizer	3.0
ganos_address_standardizer_data_us	3.0
wal2json	2.0
hll	2.14
plproxy	2.9.0
tsm_system_rows	1.0
tsm_system_time	1.0
smlar	1.0

插件名称	插件版本
tds_fdw	1.0
bigm	1.2
timescaledb	1.7.1

PostgreSQL 11

插件名称	插件版本
plpgsql	1
pg_stat_statements	1.6
btree_gin	1.3
btree_gist	1.5
citext	1.5
cube	1.4
rum	1.3
dblink	1.2
dict_int	1
earthdistance	1.1
hstore	1.5
intagg	1.1
intarray	1.2
isn	1.2
ltree	1.1
pgcrypto	1.3
pgrowlocks	1.2
pg_prewarm	1.2
pg_trgm	1.4
postgres_fdw	1
sslinfo	1.2

插件名称	插件版本
tablefunc	1
timescaledb	1.7.1
unaccent	1.1
fuzzystrmatch	1.1
pgstattuple	1.5
pg_buffercache	1.3
zhparser	1
pg_pathman	1.5
plperl	1
orafce	3.8
pg_concurrency_control	1
varbitx	1
postgis	2.5.1
pgrouting	2.6.2
postgis_sfcgal	2.5.1
postgis_topology	2.5.1
address_standardizer	2.5.1
address_standardizer_data_us	2.5.1
ogr_fdw	1
ganos_pointcloud	3.0
ganos_spatialref	3.0
log_fdw	1.0
wal2json	2.2
PL/v8	2.3.13
pg_cron	1.1
pase	0.0.1

插件名称	插件版本
hll	2.14
oss_fdw	1.1
tds_fdw	2.0.1
plproxy	2.9.0
tsm_system_rows	1.0
tsm_system_time	1.0
smlar	1.0
zombodb	4.0
bigm	1.2

PostgreSQL 10

插件名称	插件版本
pg_stat_statements	1.6
btree_gin	1.2
btree_gist	1.5
chkpass	1
citext	1.4
cube	1.2
dblink	1.2
dict_int	1
earthdistance	1.1
hstore	1.4
intagg	1.1
intarray	1.2
isn	1.1
ltree	1.1
pgcrypto	1.3

插件名称	插件版本
pgrowlocks	1.2
pg_prewarm	1.1
pg_trgm	1.3
postgres_fdw	1
sslinfo	1.2
tablefunc	1
unaccent	1.1
postgis_sfcgal	2.5.1
postgis_topology	2.5.1
fuzzystrmatch	1.1
postgis_tiger_geocoder	2.5.1
address_standardizer	2.5.1
address_standardizer_data_us	2.5.1
ogr_fdw	1
plperl	1
plv8	1.4.2
plls	1.4.2
plcoffee	1.4.2
uuid-ossp	1.1
zhparser	1
pgrouting	2.6.2
pg_hint_plan	1.3.0
pgstattuple	1.5
oss_fdw	1.1
ali_decoding	0.0.1
varbitx	1

插件名称	插件版本
pg_buffercache	1.3
q3c	1.5.0
pg_sphere	1
smlar	1
rum	1.3
pg_pathman	1.5
aggs_for_arrays	1.3.1
mysql_fdw	1
orafce	3.6
plproxy	2.8.0
pg_concurrency_control	1
postgis	2.5.1
ganos_geometry_sfcgal	3.0
ganos_geometry_topology	3.0
ganos_geometry	3.0
ganos_networking	3.0
ganos_pointcloud_geometry	3.0
ganos_pointcloud	3.0
ganos_raster	3.0
ganos_spatialref	3.0
ganos_trajectory	3.0
ganos_tiger_geocoder	3.0
ganos_address_standardizer	3.0
ganos_address_standardizer_data_us	3.0
bigm	1.2

PostgreSQL 9.4

插件名称	插件版本
plpgsql	1
pg_stat_statements	1.2
btree_gin	1
btree_gist	1
chkpass	1
citext	1
cube	1
dblink	1.1
dict_int	1
earthdistance	1
hstore	1.3
intagg	1
intarray	1
isn	1
ltree	1
pgcrypto	1.1
pgrowlocks	1.1
pg_prewarm	1
pg_trgm	1.1
postgres_fdw	1
sslinfo	1
tablefunc	1
tsearch2	1
unaccent	1
postgis	2.2.8
postgis_topology	2.2.8

插件名称	插件版本
fuzzystrmatch	1
postgis_tiger_geocoder	2.2.8
plperl	1
pltcl	1
plv8	1.4.2
plls	1.4.2
plcoffee	1.4.2
uuid-ossp	1
zhparser	1
pgrouting	2.0.0
rdkit	3.4
pg_hint_plan	1.1.3
pgstattuple	1.2
oss_fdw	1.1
jsonbx	1
ali_decoding	0.0.1
varbitx	1
pg_buffercache	1
smlar	1
pg_sphere	1
q3c	1.5.0
pg_awr	1
imgsmlr	1
orafce	3.6
pg_concurrency_control	1

5.垂直行业

5.2. 时序数据库（TimescaleDB）

RDS PostgreSQL实例新增TimescaleDB插件版本，支持时序数据的自动分片、高效写入、检索、准实时聚合等。

目前RDS PostgreSQL对外开放的是Open Source版本的TimescaleDB，由于License等问题，可能暂不支持一些高级特性，详情参见TimescaleDB。

前提条件

实例版本为PostgreSQL 11。

> 说明 部分存量用户可能已经创建过TimescaleDB插件，如果升级内核小版本后，出现如下类似提示：
>
> ```
> ERROR:  could not access file "$libdir/timescaledb-1.3.0": No such file or directory
> ```
>
> 请在对应数据库执行如下SQL更新插件：
>
> ```
> alter extension timescaledb update;
> ```

添加TimescaleDB插件

使用pgAdmin客户端连接实例，添加TimescaleDB，命令如下：

```
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

创建时序表

1. 创建标准表conditions，示例如下：

   ```
   CREATE TABLE conditions (
    time        TIMESTAMPTZ       NOT NULL,
    location    TEXT              NOT NULL,
    temperature DOUBLE PRECISION  NULL,
    humidity    DOUBLE PRECISION  NULL
   );
   ```

2. 创建时序表，示例如下：

   ```
   SELECT create_hypertable('conditions', 'time');
   ```

> 说明 详细命令说明请参见Create a Hypertable。

高效写入

您可以使用标准SQL命令将数据插入超表（Hypertables），示例如下：

```
INSERT INTO conditions(time, location, temperature, humidity)
  VALUES (NOW(), 'office', 70.0, 50.0);
```

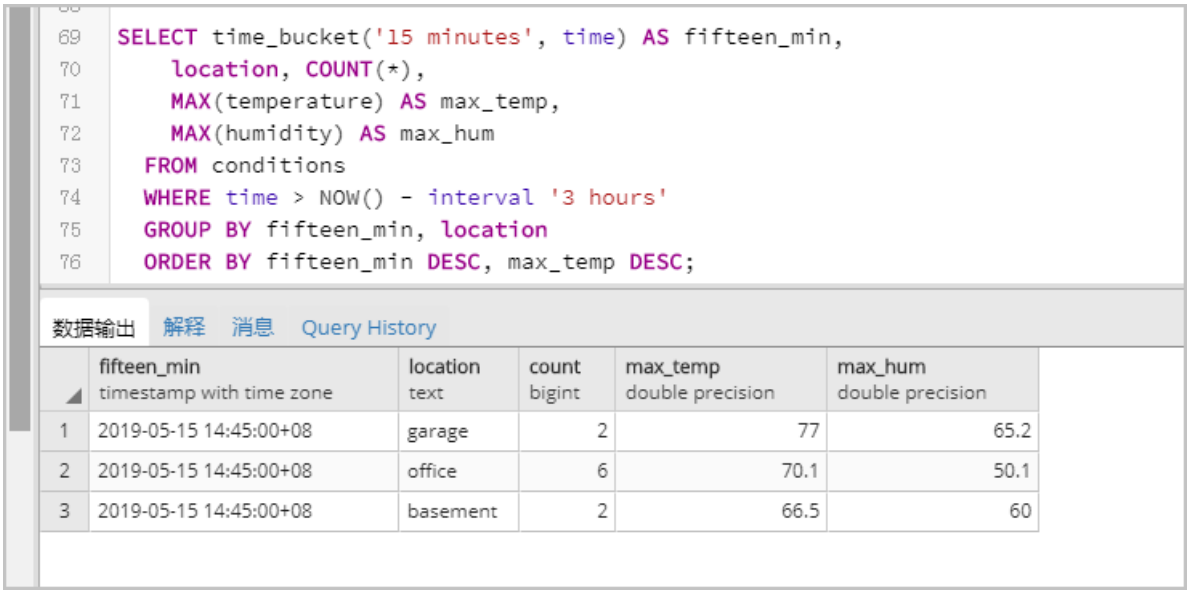
您还可以一次将多行数据插入到超表中，示例如下：

```
INSERT INTO conditions
  VALUES
    (NOW(), 'office', 70.0, 50.0),
    (NOW(), 'basement', 66.5, 60.0),
    (NOW(), 'garage', 77.0, 65.2);
```

检索

您可以使用高级SQL查询检索数据，示例如下：

```
--过去3小时内，每15分钟采集一次数据，按时间和温度排序。
SELECT time_bucket('15 minutes', time) AS fifteen_min,
    location, COUNT(*),
    MAX(temperature) AS max_temp,
    MAX(humidity) AS max_hum
  FROM conditions
  WHERE time > NOW() - interval '3 hours'
  GROUP BY fifteen_min, location
  ORDER BY fifteen_min DESC, max_temp DESC;
```



```
SELECT time_bucket('15 minutes', time) AS fifteen_min,
    location, COUNT(*),
    MAX(temperature) AS max_temp,
    MAX(humidity) AS max_hum
  FROM conditions
  WHERE time > NOW() - interval '3 hours'
  GROUP BY fifteen_min, location
  ORDER BY fifteen_min DESC, max_temp DESC;
```

数据输出 解释 消息 Query History

	fifteen_min timestamp with time zone	location text	count bigint	max_temp double precision	max_hum double precision
1	2019-05-15 14:45:00+08	garage	2	77	65.2
2	2019-05-15 14:45:00+08	office	6	70.1	50.1
3	2019-05-15 14:45:00+08	basement	2	66.5	60

您也可以使用固有的函数进行分析查询，示例如下：

```
--均值查询（Median）
SELECT percentile_cont(0.5)
 WITHIN GROUP (ORDER BY temperature)
 FROM conditions;
```

```
69  SELECT percentile_cont(0.5)
70    WITHIN GROUP (ORDER BY temperature)
71    FROM conditions;
```

数据输出 解释 消息 Query History

	percentile_cont double precision
1	70.05

```
--移动平均数（Moving Average）
SELECT time, AVG(temperature) OVER(ORDER BY time
    ROWS BETWEEN 9 PRECEDING AND CURRENT ROW)
  AS smooth_temp
 FROM conditions
 WHERE location = 'garage' and time > NOW() - interval '1 day'
 ORDER BY time DESC;
```

```
73  SELECT time, AVG(temperature) OVER(ORDER BY time
74        ROWS BETWEEN 9 PRECEDING AND CURRENT ROW)
75     AS smooth_temp
76   FROM conditions
77   WHERE location = 'garage' and time > NOW() - interval '1 day'
78   ORDER BY time DESC;
```

数据输出 解释 消息 Query History

	time timestamp with time zone	smooth_temp double precision
1	2019-05-15 14:52:30.203791+08	77
2	2019-05-15 14:52:27.426082+08	77

5.3. 数组相似度计算（smlar）

smlar插件可以用来计算两个相同类型数组的相似度。

前提条件

实例版本如下：

- PostgreSQL 12（内核小版本20200421及以上）
- PostgreSQL 11（内核小版本20200402及以上）

> 说明 您可以在基本信息页面的配置信息区域查看是否有升级内核小版本按钮。如果有按钮，您可以单击按钮查看当前版本；如果没有按钮，表示已经是最新版。详情请参见升级内核小版本。

背景信息

smlar插件提供多种函数计算两个相同类型数组的相似度，同时提供参数来控制相似度计算方法，目前支持所有内置的数据类型。

基本函数介绍

- float4 smlar(anyarray, anyarray)

  计算两个相同数据类型数组的相似度。

- float4 smlar(anyarray, anyarray, bool useIntersect)

  计算两个自定义复合类型（元素，权重）数组的相似度，复合类型如下：

  ```
  CREATE TYPE type_name AS (element_name anytype, weight_name FLOAT4);
  ```

  useIntersect为true时，计算过程只包含重叠元素的部分；为false时计算过程包含所有元素。

- float4 smlar(anyarray a, anyarray b, text formula)

  计算两个相同数据类型数组的相似度，数组通过formula指定。

  formula的预定义变量说明如下：

  - N.i：两个数组的共有元素的个数。
  - N.a：数组a中不重复元素的个数。
  - N.b：数组b中不重复元素的个数。

- float4 set_smlar_limit(float4)

  设置参数smlar.threshold的值。

- float4 show_smlar_limit()

  查看参数smlar.threshold的值。

- anyarray % anyarray

  如果两个数组的相似度大于参数smlar.threshold的值，返回true，否则返回false。

- text[] tsvector2textarray(tsvector)

  转化tsvector类型为text。

- anyarray array_unique(anyarray)

  对数组中的元素排序，排序结果不包含重复元素。

- float4 inarray(anyarray, anyelement)

如果anyelement存在于anyarray中，返回1，否则返回0。

- float4 inarray(anyarray, anyelement, float4, float4)

  如果anyelement存在于anyarray中，返回第3个参数，否则返回第4个参数。

相关参数说明和支持的数据类型请参见smlar。

使用插件

- 连接实例后创建smlar插件，命令如下：

```
testdb=> create extension smlar;
```

- 验证插件基本功能，示例如下：

```
testdb=> SELECT smlar('{1,4,6}'::int[], '{5,4,6}' );
 smlar
----------
 0.666667
(1 row)
testdb=> SELECT smlar('{1,4,6}'::int[], '{5,4,6}', 'N.i / sqrt(N.a * N.b)' );
 smlar
----------
 0.666667
(1 row)
```

- 卸载插件，命令如下：

```
testdb=> drop extension smlar;
```

5.4. 高效向量检索（PASE）

本文介绍RDS PostgreSQL如何通过PASE插件（基于IVFFlat或HNSW算法）实现高效向量检索。

前提条件

实例为RDS PostgreSQL 11。

背景信息

近年来，深度学习领域内的表示学习技术，作为人工智能的代表性技术，取得了长足性进展，在工业界中已经被大量应用，例如广告投放、人脸支付、图像识别、语音识别等场景。数据被嵌入至高维度向量，然后通过向量检索技术来查找相关的项目。

PASE（PostgreSQL ANN search extension）是一款为PostgreSQL数据库研发的高性能向量检索索引插件，使用业界中成熟稳定且高效的ANN（Approximate nearest neighbor）检索算法，包括IVFFlat和HNSW算法，通过这两种算法，可以在PG数据库中实现极高速向量查询。PASE暂时不支持特征向量的抽取与产出，您需要自己检索实体的特征向量，PASE负责的工作是根据已产出的海量级别的向量进行相似向量的检索。

目标读者

限于篇幅，本文不会对机器学习领域的相关名词做详细解释，所以阅读本文需要您有机器学习、搜索推荐相关知识基础。

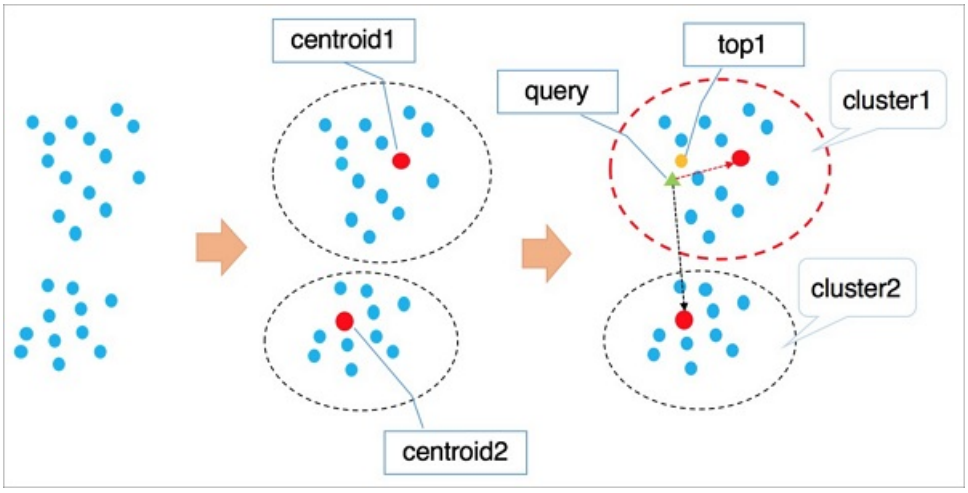
PASE算法简述

- IVFFlat算法

  IVFFlat是IVFADC的简化版本，适合于召回精度要求高，但对查询耗时要求不严格（100ms级别）的场景。相比其他算法，IVFFlat算法具有以下优点：

  - 如果查询向量是候选数据集中的一员，那么IVFFlat可以达到100%的召回率。
  - 算法简单，因此索引构建更快，存储空间更小。
  - 聚类中心点可以由使用者指定，通过简单的参数调节就可以控制召回精度。
  - 算法参数可解释性强，用户能够完全地控制算法的准确性。

  IVFFlat的算法原理参见下图。



  算法流程说明：

  i. 高维空间中的点基于隐形的聚类属性，按照kmeans等聚类算法对向量进行聚类处理，使得每个类簇有一个中心点。
  ii. 检索向量时首先遍历计算所有类簇的中心点，找到与目标向量最近的n个类簇中心。
  iii. 遍历计算n个类簇中心所在聚类中的所有元素，经过全局排序得到距离最近的k个向量。

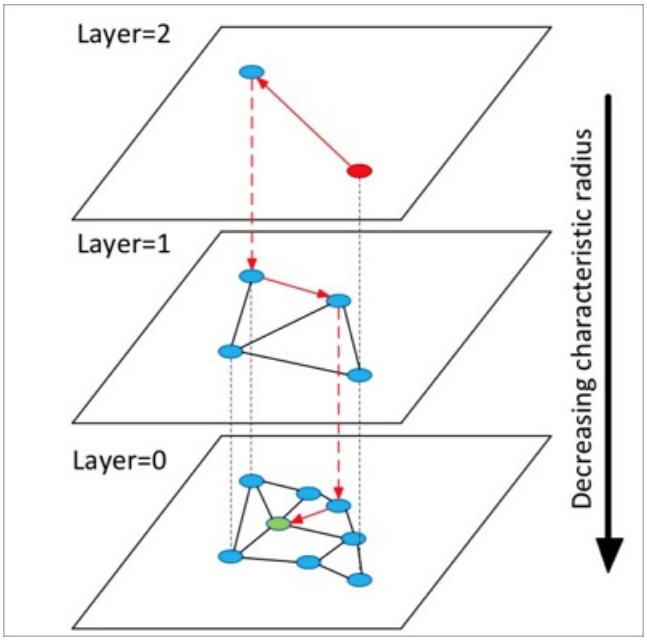
  > 说明
  > - 在查询类簇中心点时，会自动排除远离的类簇，加速查询过程，但是无法保证最优的前k个向量全部在这n个类簇中，因此会有精度损失。您可以通过类簇个数n来控制IVFFlat算法的准确性，n值越大，算法精度越高，但计算量会越大。
  > - IVFFlat和IVFADC的第一阶段完全一样，主要区别是第二阶段计算。IVFADC通过积量化来避免遍历计算，但是会导致精度损失，而IVFFlat是暴力计算，避免精度损失，并且计算量可控。

- HNSW算法

  HNSW（Hierarchical Navigable Small World）算法适合超大规模的向量数据集（千万级别以上），并且对查询延时有严格要求（10ms级别）的场景。

HNSW基于近邻图的算法，通过在近邻图快速迭代查找得到可能的相近点。在大数据量的情况下，使用HNSW算法的性能提升相比其他算法更加明显，但邻居点的存储会占用一部分存储空间，同时召回精度达到一定水平后难以通过简单的参数控制来提升。

HNSW的算法原理参见下图。



算法流程说明：

i. 构造多层图，每层图都是下层图的一个缩略，同时构成下层图的跳表，类似高速公路。
ii. 从顶层随机选中一个点开始查询。
iii. 第一次搜索其邻居点，把它们按距离目标的远近顺序存储在定长的动态列表中，以后每一次查找，依次取出动态列表中的点，搜索其邻居点，再把这些新探索的邻居点插入动态列表，每次插入动态列表需要重新排序，保留前k个。如果列表有变化则继续查找，不断迭代直至达到稳态，然后以动态列表中的第一个点作为下一层的入口点，进入下一层。
iv. 循环执行第3步，直到进入最底层。

> 说明 HNSW算法是在NSW算法的单层构图的基础上构造多层图，在图中进行最近邻查找，可以实现比聚类算法更高的查询加速比。

两种算法都有特定的适用业务场景，例如IVFFlat适合高精图像对比场景，HNSW适合搜索推荐的召回场景。后续会陆续集成业界领先的算法实现到PASE中。

使用PASE

1. 创建PASE插件。命令如下：

```
CREATE EXTENSION pase;
```

2. 计算向量相似度。您可以通过以下两种构造方式计算向量相似度：
   - 采用PASE数据类型构造方式计算

     示例

```
SELECT ARRAY[2, 1, 1]::float4[] <?> pase(ARRAY[3, 1, 1]::float4[]) AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> pase(ARRAY[3, 1, 1]::float4[], 0) AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> pase(ARRAY[3, 1, 1]::float4[], 0, 1) AS distance;
```

> 说明
>
> - <?>是PASE类型的操作符，表示计算左右两个向量的相似度。左边向量数据类型必须为float4[]，右边向量数据类型必须为pase。
> - pase类型为插件内定义的数据类型，最多包括三个构造函数。以示例第三条的 `float4[], 0, 1` 部分进行说明：第一个参数为float4[]，代表右向量数据类型；第二个参数在此处没有特殊作用，可以填入0；第三个参数表示相似度计算方式，0表示欧氏距离，1表示点积（内积）。
> - 左右向量的维度必须相等，否则计算会报错。

- 采用字符串构造方式计算

  示例

```
SELECT ARRAY[2, 1, 1]::float4[] <?> '3,1,1'::pase AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> '3,1,1:0'::pase AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> '3,1,1:0:1'::pase AS distance;
```

> 说明 采用字符串构造方式和采用PASE数据类型构造方式都是计算两个向量相似度，区别是字符串构造方式通过英文冒号（:）分隔。以示例第三条的 `3,1,1:0:1` 部分进行说明：第一个参数表示右向量；第二个参数在此处没有特殊作用，可以填入0；第三个参数表示相似度计算方式，0表示欧氏距离，1表示点积（内积）。

3. 创建索引。您可以使用两种算法创建索引：

> 说明 对于要使用PASE向量索引的用户，如果采用欧氏距离作为向量相似度计算公式，原始向量不需要做任何处理，但如果采用内积或余弦作为向量相似度计算公式，需要对向量进行归一化处理，如原始向量为 $x_1, x_2, x_3, \dots, x_n$，则需要满足：$x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2 = 1$，此时内积和余弦值相同。

- IVFFlat算法创建索引

  示例

```
CREATE INDEX ivfflat_idx ON vectors_table
USING
 pase_ivfflat(vector)
WITH
 (clustering_type = 1, distance_type = 0, dimension = 256, base64_encoded = 0, clustering_para
ms = "10,100");
```

参数说明如下。

参数	说明
clustering_type	IVFFlat算法对向量数据进行的聚类操作类型。必填项。取值： ■ 0：外部聚类，加载外部提供的中心点文件，由参数clustering_params控制。 ■ 1：内部聚类，即构建索引过程首先会在内部进行聚类操作，采用kmeans算法，由参数clustering_params控制。 对于初级用户，建议使用内部聚类方式。
distance_type	相似度计算方式。默认值为0。取值： ■ 0：欧式距离。 ■ 1：点积（内积）。使用此方式需要进行向量归一化，此时点积（内积）值的序和欧氏距离的序是反序关系。 当前仅支持欧式距离，点积（内积）需要向量归一化后，采用附录中提供的方法计算。
dimension	向量维度。必填项。
base64_encoded	数据是否采用base64编码。默认为0。取值： ■ 0：采用float4[]表示向量类型。 ■ 1：采用float[]的base64编码字符串表示向量类型。
clustering_params	对于外部聚类，该参数配置为中心点文件路径；对于内部聚类，该参数配置为聚类参数。格式为：`clustering_sample_ratio,k` 。必填项。 ■ clustering_sample_ratio：以1000为分母的聚类采样比例。取值范围为(0,1000]内的整数，例如值为1，表示对表中的数据按照千分之一的比例采样后进行kmeans聚类。值越大查询准确率越高，但创建索引的时间越长，建议采样的数据总量不要超过10万条。 ■ k：聚类中心数，值越大查询准确率越高，但创建索引时间越长，建议取值范围为[100,1000]。

- HNSW算法创建索引

  示例

```
CREATE INDEX hnsw_idx ON vectors_table
USING
 pase_hnsw(vector)
WITH
 (dim = 256, base_nb_num = 16, ef_build = 40, ef_search = 200, base64_encoded = 0);
```

  参数说明如下。

参数	说明
dim	向量维度。必填项。
base_nb_num	图中节点的邻居数。必填项。值越大查询准确率越高，但建索引时间越慢，同时索引量占空间越大，建议取值范围[16-128]。
ef_build	创建索引过程中的堆长度。必填项。越长效果越好，但创建索引越慢，建议取值范围[40,400]。
ef_search	查询过程中的堆长度。必填项。越长效果越好，但查询性能越差，可在查询时指定，该处为默认值：200。
base64_encoded	数据是否采用base64编码。默认值0。取值： ■ 0：采用float4[]表示向量类型。 ■ 1：采用float[]的base64编码字符串表示向量类型。

4. 查询。您可以使用两种索引查询：
   - 使用IVFFlat索引查询

     示例

     ```
     SELECT id, vector <#> '1,1,1'::pase as distance
     FROM vectors_ivfflat
     ORDER BY
     vector <#> '1,1,1:10:0'::pase
     ASC LIMIT 10;
     ```

     > 说明
     > - <#>为IVFFlat算法索引的操作符。
     > - 向量索引通过ORDER BY语句生效，支持ASC升序排序。
     > - pase数据类型为三段式，通过英文冒号（:）分隔。以示例的 `1,1,1:10:0` 进行说明：第一段为查询向量；第二段为IVFFlat的查询效果参数，取值范围为(0,1000]，值越大查询准确率越高，但查询性能越差，建议根据实际数据进行调试确定；第三段为查询时相似度计算方式，0表示欧式距离，1表示点积（内积）。使用点积（内积）方式需要进行向量归一化，此时点积（内积）值的序和欧氏距离的序是反序关系。

   - 使用HNSW索引查询

     示例

     ```
     SELECT id, vector <?> '1,1,1'::pase as distance
     FROM vectors_ivfflat
     ORDER BY
     vector <?> '1,1,1:100:0'::pase
     ASC LIMIT 10;
     ```

> 说明
>
> - <?>为HNSW算法索引的操作符。
> - 向量索引通过ORDER BY语句生效，支持ASC升序排序。
> - pase数据类型为三段式，通过英文冒号（:）分隔。以示例的 `1,1,1:10:0` 进行说明：第一段为查询向量；第二段为HNSW的查询效果参数，取值范围为(0,∞)，值越大查询准确率越高，但查询性能越差，建议根据实际数据进行调试确定，初始值建议从40开始；第三段为查询时相似度计算方式，0表示欧式距离，1表示点积（内积）。使用点积（内积）方式需要进行向量归一化，此时点积（内积）值的序和欧氏距离的序是反序关系。

附录

- 点积（内积）方式计算示例

  示例采用HNSW算法索引，创建function示例如下：

```
CREATE OR REPLACE FUNCTION inner_product_search(query_vector text, ef integer, k integer, table_
name text) RETURNS TABLE (id integer, uid text, distance float4) AS $$
BEGIN
   RETURN QUERY EXECUTE format('
   select a.id, a.vector <?> pase(ARRAY[%s], %s, 1) AS distance from
   (SELECT id, vector FROM %s ORDER BY vector <?> pase(ARRAY[%s], %s, 0) ASC LIMIT %s) a
   ORDER BY distance DESC;', query_vector, ef,  table_name,  query_vector, ef, k);
END
$$
LANGUAGE plpgsql;
```

> 说明　对于归一化的向量，内积=余弦，所以余弦值也可以用上述方式计算。

- IVFFlat索引自定义中心点文件

  此功能为高级功能，需要在服务器上指定路径上传中心点文件，并作为索引参数构建索引。详细参数请参见IVFFlat索引参数描述。文件格式如下：

```
向量维度|中心点个数|中心点向量集合
```

  示例

```
3|2|1,1,1,2,2,2
```

相关文档

- Product Quantization for Nearest Neighbor Search

  Herv´e Jegou, Matthijs Douze, Cordelia Schmid. Product quantization for nearest neighbor search.

- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs

Yu.A.Malkov, D.A.Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs.

5.5. 位图计算（roaringbitmap）

RDS PostgreSQL提供roaringbitmap插件，可以使用位图计算功能，提高查询性能。

前提条件

实例为RDS PostgreSQL 12。

背景信息

Roaring Bitmap算法是将32位的INT类型数据划分为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表现优秀。

操作步骤

1. 创建插件。示例如下：

```
CREATE EXTENSION roaringbitmap;
```

2. 创建带有RoaringBitmap数据类型的表。示例如下：

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

3. 使用rb_build函数插入roaringbitmap的数据。示例如下：

```
--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;
```

4. 进行Bitmap计算（OR、AND、XOR、ANDNOT）。示例如下：

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

5. 进行Bitmap聚合计算（OR、AND、XOR、BUILD），并生成新的roaringbitmap类型。示例如下：

```
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;
```

6. 统计基数（Cardinality），即统计roaringbitmap中包含多少个位置为1的BIT位。示例如下：

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

7. 从roaringbitmap中返回位置为1的BIT下标（即位置值）。示例如下：

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

Bitmap函数列表

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

函数名	输入	输出	描述	示例
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}'))
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}'))

函数名	输入	输出	描述	示例
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)
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}'))

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)
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}'))

5.6. 化学分子计算检索（RDKit）

RDS PostgreSQL提供RDKit插件，支持化学分子计算、化学分子检索等功能。

前提条件

实例版本为RDS PostgreSQL 12。

背景信息

RDKit插件支持mol数据类型（描述分子类型）和fp数据类型（描述分子指纹），在此基础上支持比较运算、相似度计算（Tanimoto、Dice）和GiST索引。

更多RDKit插件SQL操作请参见RDKit SQL。

注意事项

- mol数据类型的输入、输出函数遵循简化分子线性输入规范（SMILES）。
- fp数据类型的输入、输出功能遵循bytea（存储二进制数据的字段类型）格式。

创建插件

```
postgres=# create extension rdkit ;
CREATE EXTENSION
```

默认参数配置

```
postgres=# show rdkit.tanimoto_threshold ;
 rdkit.tanimoto_threshold
--------------------------
 0.5
(1 row)

postgres=# show rdkit.dice_threshold;
 rdkit.dice_threshold
----------------------
 0.5
(1 row)
```

索引支持

- mol和fp的比较运算操作支持Btree索引、Hash索引。例如：

  ```
  CREATE INDEX molidx ON pgmol (mol);
  CREATE INDEX molidx ON pgmol (fp);
  ```

- mol的 `%` 、 `#` 、 `@>` 、 `<@` 操作和fp结构的 `%` 、 `#` 操作支持Gist索引。例如：

  ```
  CREATE INDEX molidx ON pgmol USING gist (mol);
  ```

函数示例

- tanimoto_sml函数计算tanimoto相似度。

```
postgres=# \df tanimoto_sml
                          List of functions
 Schema |     Name     | Result data type | Argument data types | Type
--------+--------------+------------------+---------------------+------
 public | tanimoto_sml | double precision | bfp, bfp            | func
 public | tanimoto_sml | double precision | sfp, sfp            | func
(2 rows)
```

- dice_sml函数计算dice相似度。

```
postgres=# \df dice_sml
                          List of functions
 Schema |   Name    | Result data type | Argument data types | Type
--------+-----------+------------------+---------------------+------
 public | dice_sml | double precision | bfp, bfp            | func
 public | dice_sml | double precision | sfp, sfp            | func
(2 rows)
```

- substruct函数，当函数的第二个参数是第一个参数的子结构时，函数返回结果为TRUE。

```
postgres=# \df substruct
                          List of functions
 Schema |   Name    | Result data type | Argument data types | Type
--------+-----------+------------------+---------------------+------
 public | substruct | boolean          | mol, mol            | func
 public | substruct | boolean          | mol, qmol           | func
 public | substruct | boolean          | reaction, reaction  | func
(3 rows)
```

基础操作说明

- `mol % mol` 、 `fp % fp`

  当Tanimoto相似度计算结果小于GUC配置参数rdkit.tanimoto_threshold时，该操作返回结果为TRUE。

- `mol # mol` 、 `fp # fp`

  当Dice相似度计算结果小于GUC配置参数rdkit.dice_threshold时，该操作返回结果为TRUE。

- `mol @> mol`

  如果操作符 `@>` 左边对象包含右边对象，该操作返回结果为TRUE。

- `mol <@ mol`

如果操作符 `<@` 右边对象包含左边对象，该操作返回结果为TRUE。

6.异构数据库访问

6.1. 读写外部数据文本文件（oss_fdw）

阿里云支持通过oss_fdw插件将OSS中的数据加载到PostgreSQL和PPAS数据库中，也支持将PostgreSQL和PPAS数据库中的数据写入OSS中。

前提条件

实例版本如下：

- PostgreSQL 9.4
- PostgreSQL 10

oss_fdw用例

```
# PostgreSQL创建插件
create extension oss_fdw; ---对于PPAS，则执行select rds_manage_extension('create','oss_fdw');
# 创建server
CREATE SERVER ossserver FOREIGN DATA WRAPPER oss_fdw OPTIONS
    (host 'oss-cn-hangzhou.aliyuncs.com' ， id 'xxx'， key 'xxx'，bucket 'mybucket');
# 创建OSS外部表
CREATE FOREIGN TABLE ossexample
   (date text， time text， open float，
    high float， low float， volume int)
    SERVER ossserver
    OPTIONS ( filepath 'osstest/example.csv'， delimiter ', ' ，
      format 'csv'， encoding 'utf8'， PARSE_ERRORS '100');
# 创建表，数据就装载到这张表中。
create table example
     (date text， time text， open float，
      high float， low float， volume int);
# 数据从ossexample装载到example中。
insert into example select * from ossexample;

# oss_fdw能够正确估计OSS上的文件大小，正确的规划查询计划。
explain insert into example select * from ossexample;
                      QUERY PLAN
------------------------------------------------------------------------
 Insert on example  (cost=0.00..1.60 rows=6 width=92)
   ->  Foreign Scan on ossexample  (cost=0.00..1.60 rows=6 width=92)
       Foreign OssFile: osstest/example.csv.0
       Foreign OssFile Size: 728
(4 rows)
# 表example中的数据写出到OSS中。
insert into ossexample select * from example;
explain insert into ossexample select * from example;
                      QUERY PLAN
------------------------------------------------------------------
 Insert on ossexample  (cost=0.00..16.60 rows=660 width=92)
   ->  Seq Scan on example  (cost=0.00..16.60 rows=660 width=92)
(2 rows)
```

相关参数说明请参见下文。

oss_fdw参数

oss_fdw和其他fdw接口一样，对外部数据OSS中的数据进行封装。用户可以像使用数据表一样通过oss_fdw读取OSS中存放的数据。oss_fdw提供独有的参数用于连接和解析OSS上的文件数据。

> 说明
> - 目前oss_fdw支持读取和写入OSS中文件的格式为：text和csv，或者gzip格式的text、csv文件。
> - oss_fdw各参数的值需使用双引号（""）引起来，且不含无用空格。

CREATE SERVER参数

参数	说明
ossendpoint	是内网访问OSS的地址，也称为host。
id oss	账号ID。
key oss	账号Key。
bucket	存储空间，需要先创建OSS账号再设置该参数。

针对导入模式和导出模式，提供下列容错相关参数。网络条件较差时，可以调整以下参数，以保障导入和导出成功。

参数	说明
oss_connect_timeout	设置链接超时，单位秒，默认是10秒。
oss_dns_cache_timeout	设置DNS超时，单位秒，默认是60秒。
oss_speed_limit	设置能容忍的最小速率，默认是1024，即1K。
oss_speed_time	设置能容忍最小速率的最长时间，默认是15秒。

> 说明 如果使用了oss_speed_limit和oss_speed_time的默认值，表示如果连续15秒的传输速率小于1K，则超时。

CREATE FOREIGN TABLE参数

参数	说明
filepath	OSS中带路径的文件名。 • 文件名包含文件路径，但不包含bucket。 • 该参数匹配OSS对应路径上的多个文件，支持将多个文件加载到数据库。 • 文件命名为filepath和filepath.x 支持被导入到数据库，x要求从1开始，且连续。 例如，filepath、filepath.1、filepath.2、filepath.3、filepath.5 ，前4个文件会被匹配和导入，但是 filepath.5将无法导入。

参数	说明
dir	OSS中的虚拟文件目录。 • dir需要以（/）结尾。 • dir指定的虚拟文件目录中的所有文件（不包含子文件夹和子文件夹下的文件）都会被匹配和导入到数据库。
prefix	指定数据文件对应路径名的前缀，不支持正则表达式，且与 filepath、dir 互斥，三者只能设置其中一个。
format	指定文件的格式，目前只支持csv。
encoding	文件中数据的编码格式，支持常见的pg编码，如utf8。
parse_errors	容错模式解析，以行为单位，忽略文件分析过程中发生的错误。
delimiter	指定列的分割符。
quote	指定文件的引用字符。
escape	指定文件的逃逸字符。
null	指定匹配对应字符串的列为null，例如null 'test'，即列值为'test'的字符串为null。
force_not_null	指定某些列的值不为null。例如，`force_not_null 'id'` 表示：如果ID列的值为空，则该值为空字符串，而不是null。
compressiontype	设置读取和写入OSS上文件的格式： • none：默认的文件类型，即没有压缩的文本格式。 • gzip：读取文件的格式为gzip压缩格式。
compressionlevel	设置写入OSS的压缩格式的压缩等级，范围1到9，默认为6。

> **说明**
> - filepath和dir需要在OPTIONS参数中指定。
> - filepath和dir必须指定两个参数中的其中一个，且不能同时指定。
> - 导出模式目前只支持虚拟文件夹的匹配模式，即只支持dir，不支持filepath。

CREATE FOREIGN TABLE的导出模式参数

- oss_flush_block_size：单次刷出到OSS的buffer大小，默认32MB，可选范围1到128MB。
- oss_file_max_size：写入OSS的最大文件大小，超出之后会切换到另一个文件续写。默认1024MB，可选范围8到4000 MB。
- num_parallel_worker：写OSS数据的压缩模式中并行压缩线程的个数，范围1到8，默认并发数3。

辅助函数

FUNCTION oss_fdw_list_file (relname text, schema text DEFAULT 'public')

- 用于获得某个外部表所匹配的OSS上的文件名和文件的大小。
- 文件大小的单位是字节。

```
select * from oss_fdw_list_file('t_oss');
         name          |  size
-------------------------------+-----------
 oss_test/test.gz.1 | 739698350
 oss_test/test.gz.2 | 739413041
 oss_test/test.gz.3 | 739562048
(3 rows)
```

辅助功能

oss_fdw.rds_read_one_file：在读模式下，指定某个外表匹配的文件。设置后，该外部表在数据导入中只匹配这个被设置的文件。

例如，set oss_fdw.rds_read_one_file = ‘oss_test/example16.csv.1’;

```
set oss_fdw.rds_read_one_file = 'oss_test/test.gz.2';
select * from oss_fdw_list_file('t_oss');
         name          |  size
-------------------------------+-----------
 oss_test/test.gz.2 | 739413041
(1 rows)
```

oss_fdw注意事项

- oss_fdw是在PostgreSQL FOREIGN TABLE框架下开发的外部表插件。
- 数据导入的性能和PostgreSQL集群的资源（CPU、IO、MEM）相关，也和OSS相关。
- 为保证数据导入的性能，请确保云数据库PostgreSQL与OSS所在Region相同，相关信息请参见OSS endpoint 信息。
- 如果读取外表的SQL时触发 `ERROR: oss endpoint userendpoint not in aliyun white list` ，建议使用阿里云各可用区公共endpoint，详情请参见访问域名和数据中心。如果问题仍无法解决，请通过工单反馈。

错误处理

导入或导出出错时，日志中会出现下列错误提示信息：

- code：出错请求的HTTP状态码。
- error_code：OSS的错误码。
- error_msg：OSS的错误信息。
- req_id：标识该次请求的UUID。当您无法解决问题时，可以凭req_id来请求OSS开发工程师的帮助。

请参见以下链接中的文档了解和处理各类错误，超时相关的错误可以使用oss_ext相关参数处理。

- OSS help 页面
- PostgreSQL CREATE FOREIGN TABLE 手册

- OSS 错误处理
- OSS 错误响应

ID和Key隐藏

CREATE SERVER中的ID和Key信息如果不做任何处理，用户可以使用 `select * from pg_foreign_server` 看到明文信息，会暴露用户的ID和Key。我们通过对ID和Key进行对称加密实现对ID和Key的隐藏（不同的实例使用不同的密钥，最大限度保护用户信息），但无法使用类似GP一样的方法，增加一个数据类型，会导致老实例不兼容。

最终加密后的信息如下：

```
postgres=# select * from pg_foreign_server ;
 srvname  | srvowner | srvfdw | srvtype | srvversion | srvacl |
srvoptions
----------+----------+--------+---------+------------+--------+-----------------------------------------------------
---------------------------------------------------------------------------
---------------------------------
 ossserver |     10 |  16390 |       |        |      | {host=oss-cn-hangzhou-zmf.aliyuncs.com，id=MD5xxxxxx
xx，key=MD5xxxxxxxx，bucket=067862}
```

加密后的信息将会以MD5开头（总长度为len，len%8==3），这样导出之后再导入不会再次加密，但是用户不能创建MD5开头的Key和ID。

6.2. 读写MySQL数据（mysql_fdw）

RDS PostgreSQL提供mysql_fdw插件，可以读写RDS MySQL实例或自建MySQL数据库里的数据。

前提条件

- 实例为RDS PostgreSQL 10（云盘）或12（云盘）。
- PostgreSQL和MySQL数据库网络互通。通过设置白名单、设置本地防火墙等保证网络互通且使用的连接地址正确。

背景信息

PostgreSQL从9.6开始就支持并行计算，到11的时候并行计算性能得到巨大提升，10亿数据量的join查询可以实现秒级完成。所以很多用户会使用PostgreSQL作为小的数据仓库使用，同时又能提供高并发访问。未来推出的PostgreSQL 13还将支持列存储引擎，分析能力还会有巨大的提升。

使用mysql_fdw插件能够将PostgreSQL和MySQL连接，同步MySQL数据进行数据分析。

操作步骤

1. 新建mysql_fdw插件。

   ```
   postgres=> create extension mysql_fdw;
   CREATE EXTENSION
   ```

2. 创建MySQL服务器定义。

```
postgres=> CREATE SERVER <server名称>
postgres->     FOREIGN DATA WRAPPER mysql_fdw
postgres->     OPTIONS (host '<连接地址>', port '<连接端口>');
CREATE SERVER
```

示例

```
postgres=> CREATE SERVER mysql_server
postgres->     FOREIGN DATA WRAPPER mysql_fdw
postgres->     OPTIONS (host 'rm-xxx.mysql.rds.aliyuncs.com', port '3306');
CREATE SERVER
```

3. 创建用户映射，将MySQL服务器定义映射到PostgreSQL的某个用户上，将来使用这个用户访问MySQL的数据。

```
postgres=> CREATE USER MAPPING FOR <PostgreSQL用户名>
SERVER <server名称>
OPTIONS (username '<MySQL用户名>', password '<MySQL用户对应密码>');
CREATE USER MAPPING
```

示例

```
postgres=> CREATE USER MAPPING FOR pgtest
SERVER mysql_server
OPTIONS (username 'mysqltest', password 'Test1234!');
CREATE USER MAPPING
```

4. 使用上一步骤的PostgreSQL用户创建MySQL的外部表。

> 说明 外部表的字段名要与MySQL数据库中表的字段名相同，同时可以仅创建您想要查询的字段。例如MySQL数据库中的表有3个字段ID、NAME、AGE，您可以仅创建其中2个字段ID、NAME。

```
postgres=> CREATE FOREIGN TABLE <表名> (<字段名> <数据类型>,<字段名> <数据类型>...) server <server名称> options (dbname '<MySQL数据库名>', table_name '<MySQL表名>');
CREATE FOREIGN TABLE
```

示例

```
postgres=> CREATE FOREIGN TABLE ft_test (id1 int, name1 text) server mysql_server options (dbname 'test123', table_name 'test');
CREATE FOREIGN TABLE
```

测试读写

您可以通过外部表读写MySQL数据。

> 说明 MySQL对应的表必须有主键才可以写入数据，否则会报如下错误：
>
> ERROR: first column of remote table must be unique for INSERT/UPDATE/DELETE operation.

```
postgres=> select * from ft_test ;

postgres=> insert into ft_test values (2,'abc');
INSERT 0 1

postgres=> insert into ft_test select generate_series(3,100),'abc';
INSERT 0 98
postgres=> select count(*) from ft_test ;
 count
-------
    99
(1 row)
```

检查执行计划，即PostgreSQL查询MySQL数据的请求在MySQL中是如何执行的。

```
postgres=> explain verbose select count(*) from ft_test ;
                          QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=1027.50..1027.51 rows=1 width=8)
   Output: count(*)
   ->  Foreign Scan on public.ft_test  (cost=25.00..1025.00 rows=1000 width=0)
         Output: id, info
         Remote server startup cost: 25
         Remote query: SELECT NULL FROM `test123`.`test`
(6 rows)

postgres=> explain verbose select id from ft_test where id=2;
                        QUERY PLAN
-----------------------------------------------------------------------
 Foreign Scan on public.ft_test  (cost=25.00..1025.00 rows=1000 width=4)
   Output: id
   Remote server startup cost: 25
   Remote query: SELECT `id` FROM `test123`.`test` WHERE ((`id` = 2))
(4 rows)
```

6.4. 查询日志（log_fdw）

RDS PostgreSQL提供log_fdw插件，可以查询数据库日志。

前提条件

实例为RDS PostgreSQL 11。

背景信息

log_fdw插件提供如下两个函数，帮助您查询数据库日志。

- `list_postgres_log_files()`：列出所有的csvlog文件。
- `create_foreign_table_for_log_file(IN table_name text, IN log_server text, IN log_file text)`：创建一个与特定csvlog文件相对应的外部表。

操作步骤

1. 创建log_fdw插件。

   ```
   postgres=> create extension log_fdw;
   CREATE EXTENSION
   ```

2. 创建log服务器定义。

   ```
   postgres=> create server <server名称> foreign data wrapper log_fdw;
   ```

示例

```
postgres=> create server log_server foreign data wrapper log_fdw;
CREATE SERVER
```

3. 调用 `list_postgres_log_files()` 函数，列出所有的csvlog文件。

```
postgres=> select * from list_postgres_log_files() order by 1;
         file_name          | file_size_bytes
------------------------------------+-----------------
 postgresql-2020-01-10_095546.csv |         3794
 postgresql-2020-01-10_100336.csv |       318318
 postgresql-2020-01-11_000000.csv |       198437
 postgresql-2020-01-11_083546.csv |         4775
 postgresql-2020-01-13_030618.csv |         3347
```

4. 调用 `create_foreign_table_for_log_file(IN table_name text, IN log_server text, IN log_file text)` 函数，创建一个与特定csvlog文件相对应的外部表。

```
postgres=> select create_foreign_table_for_log_file('<外部表名称>', '<log服务器名称>', '<csvlog文件名称>');
```

示例

```
postgres=> select create_foreign_table_for_log_file('ft1', 'log_server', 'postgresql-2020-01-13_030618.csv');
 create_foreign_table_for_log_file
-----------------------------------
 t
(1 row)
```

5. 查询外部表即可查询到日志内容。

```
postgres=> select log_time, message from <外部表名称> order by log_time desc limit 2;
```

示例

```
postgres=> select log_time, message from ft1 order by log_time desc limit 2;
       log_time        |                      message
---------------------------+-------------------------------------------------------------------------
 2020-01-13 03:35:00.003+00 | cron job 1 completed: INSERT 0 1 1
 2020-01-13 03:35:00+00     | cron job 1 starting: INSERT INTO cron_test VALUES ('Hello World')
(2 rows)
```

外部表结构

```
postgres=> \d+ ft1
                                    Foreign table "public.ft1"
      Column       |         Type          | Collation | Nullable | Default | FDW options | Storage  | Stats targ
et | Description
-----------------------+---------------------------+-----------+----------+---------+-------------+----------+-----
---------+-------------
 log_time             | timestamp(3) with time zone |           |          |         |             | plain    |
 user_name            | text                        |           |          |         |             | extended |
 database_name        | text                        |           |          |         |             | extended |
 process_id           | integer                     |           |          |         |             | plain    |
 connection_from      | text                        |           |          |         |             | extended |
 session_id           | text                        |           |          |         |             | extended |
 session_line_num     | bigint                      |           |          |         |             | plain    |
 command_tag          | text                        |           |          |         |             | extended |
 session_start_time   | timestamp with time zone    |           |          |         |             | plain    |
 virtual_transaction_id | text                      |           |          |         |             | extended |
 transaction_id       | bigint                      |           |          |         |             | plain    |
 error_severity       | text                        |           |          |         |             | extended |
 sql_state_code       | text                        |           |          |         |             | extended |
 message              | text                        |           |          |         |             | extended |
 detail               | text                        |           |          |         |             | extended |
 hint                 | text                        |           |          |         |             | extended |
 internal_query       | text                        |           |          |         |             | extended |
 internal_query_pos   | integer                     |           |          |         |             | plain    |
 context              | text                        |           |          |         |             | extended |
 query                | text                        |           |          |         |             | extended |
 query_pos            | integer                     |           |          |         |             | plain    |
 location             | text                        |           |          |         |             | extended |
 application_name     | text                        |           |          |         |             | extended |
Server: log_server
FDW options: (filename 'postgresql-2020-01-13_030618.csv')
```

6.5. 查询其他类型数据库（tds_fdw）

您可以使用tds_fdw插件查询其他类型数据库的数据。

前提条件

实例版本如下：

- PostgreSQL 12（内核小版本20200421及以上）
- PostgreSQL 11（内核小版本20200402及以上）

> **说明** 您可以在基本信息页面的配置信息区域查看是否有升级内核小版本按钮。如果有按钮，您可以单击按钮查看当前版本；如果没有按钮，表示已经是最新版。详情请参见升级内核小版本。

背景信息

tds_fdw是PostgreSQL外部数据包装器，可以连接到使用Tabular Data Stream（TDS）协议的数据库，例如Sybase数据库和Microsoft SQL server。

详细说明请参见postgres_fdw。

创建插件

连接实例后创建tds_fdw插件，命令如下：

```
create extension tds_fdw;
```

使用插件

1. 创建服务器，示例如下：

   ```
   CREATE SERVER mssql_svr
    FOREIGN DATA WRAPPER tds_fdw
    OPTIONS (servername '127.0.0.1', port '1433', database 'tds_fdw_test', tds_version '7.1');
   ```

2. 创建外部表。您可以通过多种方式创建外部表：
   - 使用table_name定义创建外部表，示例如下：

     ```
     CREATE FOREIGN TABLE mssql_table (
      id integer,
      data varchar)
      SERVER mssql_svr
      OPTIONS (table_name 'dbo.mytable', row_estimate_method 'showplan_all');
     ```

   - 使用schema_name和table_name定义创建外部表，示例如下：

     ```
     CREATE FOREIGN TABLE mssql_table (
      id integer,
      data varchar)
      SERVER mssql_svr
      OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
     ```

   - 使用query定义创建外部表，示例如下：

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 data varchar)
 SERVER mssql_svr
 OPTIONS (query 'SELECT * FROM dbo.mytable', row_estimate_method 'showplan_all');
```

- 使用远程列名创建外部表，示例如下：

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 col2 varchar OPTIONS (column_name 'data'))
 SERVER mssql_svr
 OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
```

3. 创建用户映射，示例如下：

```
CREATE USER MAPPING FOR postgres
 SERVER mssql_svr
 OPTIONS (username 'sa', password '123456');
```

4. 导入外部模式，示例如下：

```
IMPORT FOREIGN SCHEMA dbo
 EXCEPT (mssql_table)
 FROM SERVER mssql_svr
 INTO public
 OPTIONS (import_default 'true');
```

6.6. 同步更新Oracle数据库（oracle_fdw）

RDS PostgreSQL提供oracle_fdw插件，可以连接到Oracle数据库，通过操作PostgreSQL表同步更新Oracle数据库中的表。

前提条件

- 实例为RDS PostgreSQL 12（内核版本20200421及以上）。

  > **说明** 您可以执行 `show rds_supported_extensions;` 查看是否支持oracle_fdw，不支持的话请升级内核版本。

- Oracle Client版本为11.2及以上。
- Oracle Server版本要求取决于Oracle Client版本。详情请参见Oracle官方文档。

背景信息

oracle_fdw是PostgreSQL外部表插件，可以读取Oracle数据库的数据，也非常方便地实现PostgreSQL与Oracle数据同步。

更多详细信息请参见oracle_fdw。

注意事项

- 如果需要执行UPDATE和DELETE操作，需要在创建外部表时为主键列设置key参数，详情请参见创建外部表。
- 外部表定义的列数据类型必须是oracle_fdw可以识别并可以转换的，oracle_fdw插件对于数据类型的转换规则请参见Data types。
- WHERE子句和ORDER BY子句支持计算下推，即oracle_fdw会将子句发送给Oracle进行计算。
- JOIN操作支持计算下推，但是有以下注意事项：
  - 两个表必须被定义在相同的映射中。
  - 三个及以上的JOIN操作不支持下推。
  - JOIN操作必须包含在SELECT操作中。
  - 没有JOIN条件的CROSS JOIN操作不支持下推。
  - 如果JOIN子句被下推，ORDER BY子句将不会被下推。
- oracle_fdw支持postgis插件，安装postgis插件后支持空间数据类型，包括：
  - POINT
  - LINE
  - POLYGON
  - MULTIPOINT
  - MULTILINE
  - MULTIPOLYGON

操作步骤

1. 新建oracle_fdw插件。命令如下：

   ```
   CREATE EXTENSION oracle_fdw;
   ```

2. 创建Oracle数据库映射。有如下两种命令：

   ```
   CREATE SERVER <SERVER名称>
   FOREIGN DATA WRAPPER oracle_fdw
   OPTIONS (dbserver '//<连接地址>:<连接端口>/<数据库名>');
   ```

   示例

   ```
   CREATE SERVER <SERVER名称>
   FOREIGN DATA WRAPPER oracle_fdw
   OPTIONS (dbserver '//127.0.0.1:5432/oradbname');
   ```

   ```
   CREATE SERVER oradb
   FOREIGN DATA WRAPPER oracle_fdw
   OPTIONS (host '<连接地址>', port '<连接端口>', dbname '<数据库名>');
   ```

   示例

```
CREATE SERVER oradb
FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (host '127.0.0.1', port '5432', dbname 'oradbname');
```

3. 创建用户映射。命令如下：

```
CREATE USER MAPPING
FOR <PostgreSQL用户名> SERVER <映射名>
OPTIONS (user '<Oracle数据库用户名>', password '<Oracle数据库用户密码>');
```

> 说明 如果不在PostgreSQL数据库中存储Oracle用户凭证，可以设置user为空字符串，然后提供必须的外部授权。

示例

```
CREATE USER MAPPING
FOR pguser SERVER oradb
OPTIONS (user 'orauser', password 'orapwd');
```

4. 创建Oracle的外部表。示例如下：

```
CREATE FOREIGN TABLE oratab (
        id       integer OPTIONS (key 'true')  NOT NULL,
        text     character varying(30),
        floating  double precision  NOT NULL
      ) SERVER oradb OPTIONS (table 'ORATAB',
                        schema 'ORAUSER',
                        max_long '32767',
                        readonly 'false',
                        sample_percent, '100',
                        prefetch, '200');
```

> 说明 外部表的结构需要和Oracle中的映射表结构保持一致。

OPTIONS内的参数说明如下。

参数	说明
key	是否设置对应的列为主键，取值为true或false，默认值为false。如果要执行UPDATE和DELETE操作，必须将所有主键列设置为true。
table	表名，一般是大写，必填参数。可以使用Oracle的SQL来定义table变量的值，例如：`OPTIONS (table '(SELECT col FROM tab WHERE val = "string")')`，此时不要使用schema参数。
schema	一般是Oracle用户名，保持大写，用来访问不属于当前连接用户的表。

参数	说明
max_long	限制Oracle表中LONG、LONG RAW、XMLTYPE类型列的最大长度，取值范围是1~1073741823，默认值是32767。
readonly	限制Oracle表为只读，不允许INSERT、UPDATE、DELETE操作。
sample_percent	设置随机选择Oracle表数据的比例，用于PostgreSQL表统计信息，取值范围是0.000001~100，默认值是100。
prefetch	外表扫描时，PostgreSQL和Oracle数据表之间一次性传输的行数，取值范围是0~1024，默认值是200，0代表取消prefetch功能。

完成以上步骤即可通过操作外表来实现对Oracle表的操作。支持DELETE、INSERT、UPDATE、SELECT等基本操作，支持导入外部表定义的操作，命令如下：

```
IMPORT FOREIGN SCHEMA <ora_schema_name>
FROM SERVER <server_name>
INTO <schema_name>
OPTIONS (case 'lower');
```

> **说明** case取值如下：
> - keep：表示保留Oracle上的对象名，通常是大写。
> - lower：表示转换所有的对象名为小写。
> - smart：表示只将对象名中都是大写字母的替换为小写。

卸载插件

卸载插件命令如下：

```
DROP EXTENSION oracle_fdw;
```

7.跨库操作（dblink、postgres_fdw）

使用PostgreSQL本身提供的扩展插件，例如dblink和postgres_fdw，可以跨库操作表。

背景信息

阿里云RDS for PostgreSQL 11云盘版实例开放dblink和postgres_fdw插件，支持相同VPC内实例（包括自建PostgreSQL数据库）间的跨库操作。如果要访问VPC外部的其他实例，可以通过相同VPC内ECS的端口跳转实现。

购买PostgreSQL 11云盘版实例。

注意事项

PostgreSQL 11云盘版的dblink和postgres_fdw插件进行跨库操作的注意事项如下：

- 相同VPC内的ECS/RDS PostgreSQL实例可以直接跨库操作。
- RDS PostgreSQL实例可以通过本VPC内的ECS实例进行端口跳转，实现跨库操作。
- 自建PostgreSQL实例可以通过oracle_fdw或mysql_fdw连接VPC外部的Oracle实例或MySQL实例。
- 连接自身跨库操作时，host请填写localhost，port请填写 `show port` 命令返回的本地端口。

使用dblink

1. 新建dblink插件。

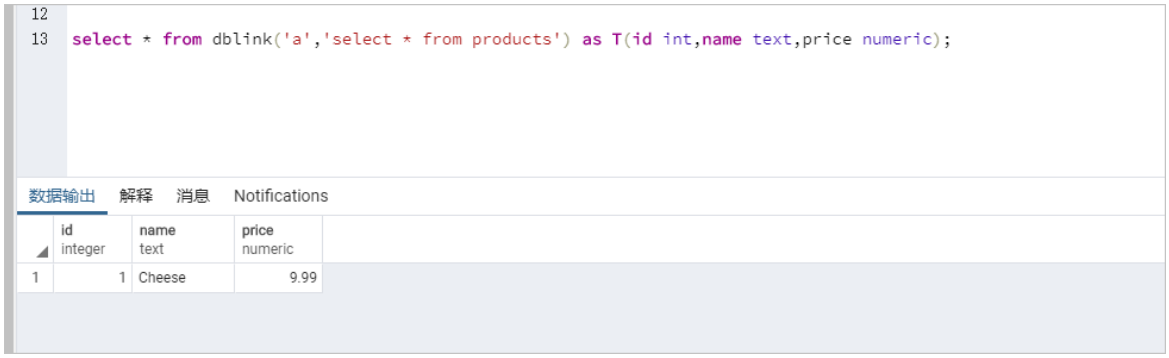
```
create extension dblink;
```

2. 创建dblink连接。

```
postgres=> select dblink_connect('<连接名称>', 'host=<同一VPC下的另一RDS的内网域名> port=<同一VPC下的另一RDS的内网监听端口> user=<远程数据库用户名> password=<密码> dbname=<库名>');

postgres=> SELECT * FROM dblink('<连接名称>', '<SQL命令>') as <表名>(<列名> <列类型>);
```

示例

```
postgres=> select dblink_connect('a', 'host=pgm-bpxxxxx.pg.rds.aliyuncs.com port=3433 user=testuser2 password=passwd1234 dbname=postgres');


postgres=> select * from dblink('a','select * from products') as T(id int,name text,price numeric); //查询远端表
```



更多详情请参见dblink。

使用postgres_fdw

1. 新建一个数据库。

```
postgres=> create database <数据库名>; //创建数据库


postgres=> \c <数据库名> //切换数据库
```

示例

```
postgres=> create database db1;
CREATE DATABASE


postgres=> \c db1
```

2. 新建postgres_fdw插件。

```
db1=> create extension postgres_fdw;
```

3. 新建远程数据库服务器。

```
db1=> CREATE SERVER <server名称>
      FOREIGN DATA WRAPPER postgres_fdw
      OPTIONS (host '<同一VPC下的另一RDS的内网域名>',port '<同一VPC下的另一RDS的内网监听端口>', db
name '<同一VPC下的另一RDS的库名>');


db1=> CREATE USER MAPPING FOR <本地数据库用户名>
      SERVER <server名称>
      OPTIONS (user '<远程数据库用户名>', password '<远程数据库密码>');
```

示例

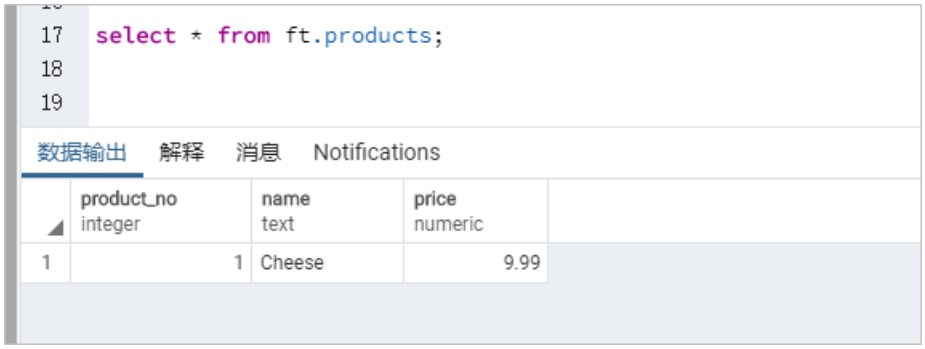
```
db1=> CREATE SERVER foreign_server1
      FOREIGN DATA WRAPPER postgres_fdw
      OPTIONS (host 'pgm-bpxxxxx.pg.rds.aliyuncs.com', port '3433', dbname 'postgres');
CREATE SERVER

db1=> CREATE USER MAPPING FOR testuser
      SERVER foreign_server1
      OPTIONS (user 'testuser2', password 'passwd1234');
CREATE USER MAPPING
```

4. 导入外部表。

```
db1=> import foreign schema public from server foreign_server1 into <SCHEMA名称>; //导入外部表

db1=> select * from <SCHEMA名称>.<表名>     //查询远端表
```

示例

```
db1=> import foreign schema public from server foreign_server1 into ft;
IMPORT FOREIGN SCHEMA

db1=> select * from ft.products;
```



更多详情请参见postgres_fdw。

8.基数统计（hll）

hll插件支持的数据类型HyperLogLog可以帮助您快速预估PV、UV等业务指标。

前提条件

实例版本如下：

- PostgreSQL 12（内核小版本20200421及以上）
- PostgreSQL 11（内核小版本20200402及以上）

> **说明** 您可以在基本信息页面的配置信息区域查看是否有升级内核小版本按钮。如果有按钮，您可以单击按钮查看当前版本；如果没有按钮，表示已经是最新版。详情请参见升级内核小版本。

背景信息

hll插件支持一种可变长、类似集合的数据类型HyperLogLog（hll），常用于在指定精度下返回近似的distinct值，例如1280字节的hll数据类型可以高精度地估计出近百亿的distinct值。hll适合互联网广告分析或其他有类似预估分析计算需求的行业，可以快速预估PV、UV等业务指标。

更多hll插件使用方法请参见postgresql-hll。

详细算法说明请参见论文HyperLogLog: the analysis of a near-optimalcardinality estimation algorithm。

创建hll插件

连接实例后创建hll插件，命令如下：

```
CREATE EXTENSION hll;
```

基础操作

- 创建一个含有hll字段的表，命令如下：

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

- 将int数据类型转换为hll_hashval，命令如下：

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

基本操作符

- hll类型支持如下操作符：
  - =
  - !=
  - <>

- ||
- #

示例如下：

```
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类型支持如下操作符：
  - =
  - !=
  - <>

  示例如下：

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

基本函数

- 支持hll_hash_boolean、hll_hash_smallint和hll_hash_bigint等hash函数，示例如下：

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

- 支持hll_add_agg函数，可以将int转换为hll格式，示例如下：

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

- 支持hll_union函数，可以将hll并集，示例如下：

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

- 支持hll_set_defaults函数，可以设置精度，示例如下：

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

- 支持hll_print函数，用于打印debug信息，示例如下：

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

示例

```
create table access_date (acc_date date unique, userids hll);
insert into access_date select current_date, hll_add_agg(hll_hash_integer(user_id)) from generate_ser
ies(1,10000) t(user_id);
insert into access_date select current_date-1, hll_add_agg(hll_hash_integer(user_id)) from generate_s
eries(5000,20000) t(user_id);
insert into access_date select current_date-2, hll_add_agg(hll_hash_integer(user_id)) from generate_s
eries(9000,40000) t(user_id);
postgres=# select #userids from access_date where acc_date=current_date;
   ?column?
------------------
 9725.85273370708
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-1;
   ?column?
------------------
 14968.6596883279
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-2;
   ?column?
------------------
 29361.5209149911
(1 row)
```

9.定时任务（pg_cron）

RDS PostgreSQL提供pg_cron插件，可以设置定时任务。

前提条件

实例为RDS PostgreSQL 11。

> 说明 仅新购实例支持，存量实例请提交工单开通。

背景信息

pg_cron是基于cron的作业调度插件，与常规cron保持相同的语法，但它可以直接从数据库安排PostgreSQL命令。

每一个定时任务分为两部分：

- 定时计划

  规定了用户使用插件的计划，例如每隔1分钟执行一次该任务。

- 定时任务

  用户具体的任务内容，例如 `select * from some_table` 。

语法

pg_cron使用标准的cron语法，其中*表示任意时间都运行，特定数字表示仅在这个数字时运行。

```
┌──────────────────────────── 分钟 (0 - 59)
│ ┌────────────────────────── 小时 (0 - 23)
│ │ ┌──────────────────────── 日期 (1 - 31)
│ │ │ ┌────────────────────── 月份 (1 - 12)
│ │ │ │ ┌──────────────────── 一周中的某一天 (0 - 6) (0表示周日)
│ │ │ │ │
│ │ │ │ │
│ │ │ │ │
* * * * *
```

示例

- 创建插件

```
CREATE EXTENSION pg_cron;
```

- 增加任务项

```
-- 周六3:30am (GMT) 删除过期数据。
SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval '1 wee
k'$$);

----------

-- 每天的10:00am (GMT) 执行磁盘清理。
SELECT cron.schedule('0 10 * * *', 'VACUUM');

----------

-- 每分钟执行指定脚本。
SELECT cron.schedule('* * * * *', 'select 1;');

----------

-- 每个小时的23分执行指定脚本。
SELECT cron.schedule('23 * * * *', 'select 1;');

----------

-- 每个月的4号执行指定脚本。
SELECT cron.schedule('* * 4 * *', 'select 1;');
```

- 查看当前任务

```
SELECT * FROM cron.job;

 jobid | schedule   |  command  | nodename  | nodeport | database | username | active
-------+------------+-----------+-----------+----------+----------+----------+--------
    43 | 0 10 * * * |   VACUUM; | localhost |     5433 | postgres | test     | t
```

- 删除任务项

```
SELECT cron.unschedule(43);
```

> 说明　43为jobid。

10.分片代理（PL/Proxy）

PL/Proxy插件包含CLUSTER模式和CONNECT模式，可以帮助您用不同方式访问数据库。

前提条件

实例版本如下：

- PostgreSQL 12（内核小版本20200421及以上）
- PostgreSQL 11（内核小版本20200402及以上）

> **说明** 您可以在基本信息页面的配置信息区域查看是否有升级内核小版本按钮。如果有按钮，您可以单击按钮查看当前版本；如果没有按钮，表示已经是最新版。详情请参见升级内核小版本。

背景信息

PL/Proxy插件包含如下两种模式：

- CLUSTER模式

  支持数据库水平拆分和SQL复制。

- CONNECT模式

  支持将SQL请求路由到指定的数据库。

更多PL/Proxy插件使用方法请参见PL/Proxy。

注意事项

- 相同VPC内的PostgreSQL实例可以直接跨库操作。
- 不同VPC内的PostgreSQL实例可以通过本VPC内的ECS实例进行端口跳转，实现跨库操作。
- 代理节点后端的数据节点数必须是2的N次方。

测试环境

选择一个数据库实例作为代理节点，另外两个数据库实例作为数据节点。详细信息如下。

IP	节点类型	数据库名	用户名
100.xx.xx.136	代理节点	postgres	postgres
100.xx.xx.72	数据节点	pl_db0	postgres
11.xx.xx.9	数据节点	pl_db1	postgres

创建PL/Proxy插件

创建PL/Proxy插件命令如下：

```
create extension plproxy
```

创建PL/Proxy集群

> 说明 CONNECT模式不需要进行本操作。

1. 创建PL/Proxy集群，指定连接的子节点的数据库名、IP地址和端口，示例如下：

```
postgres=# CREATE SERVER cluster_srv1 FOREIGN DATA WRAPPER plproxy
postgres-# OPTIONS (
postgres(#       connection_lifetime '1800',
postgres(#       disable_binary '1',
postgres(#       p0 'dbname=pl_db0 host=100.xxx.xxx.72 port=5678',
postgres(#       p1 'dbname=pl_db1 host=11.xxx.xxx.9 port=5678'
postgres(#       );
CREATE SERVER
```

2. 为postgres用户赋予权限，示例如下：

```
postgres=# grant usage on FOREIGN server cluster_srv1 to postgres;
GRANT
```

3. 创建用户映射，示例如下：

```
postgres=> create user mapping for postgres server cluster_srv1 options (user 'postgres');
CREATE USER MAPPING
```

创建测试表

在每个数据节点创建测试表（代理节点不需要创建），示例如下：

```
create table users(userid int, name text);
```

CLUSTER模式测试

数据水平拆分测试步骤如下：

1. 在每个数据节点创建插入函数，示例如下：

```
pl_db0=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
pl_db0-> RETURNS integer AS $$
pl_db0$>       INSERT INTO users (userid, name) VALUES ($1,$2);
pl_db0$>       SELECT 1;
pl_db0$> $$ LANGUAGE SQL;
CREATE FUNCTION

pl_db1=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
pl_db1-> RETURNS integer AS $$
pl_db1$>       INSERT INTO users (userid, name) VALUES ($1,$2);
pl_db1$>       SELECT 1;
pl_db1$> $$ LANGUAGE SQL;
CREATE FUNCTION
```

2. 在代理节点创建同名的插入函数，示例如下：

```
postgres=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
postgres-> RETURNS integer AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ANY;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

3. 在代理节点创建读取函数，示例如下：

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ALL ;
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

4. 在代理节点插入10条测试记录，示例如下：

```
SELECT insert_user(1001, 'Sven');
SELECT insert_user(1002, 'Marko');
SELECT insert_user(1003, 'Steve');
SELECT insert_user(1004, 'lottu');
SELECT insert_user(1005, 'rax');
SELECT insert_user(1006, 'ak');
SELECT insert_user(1007, 'jack');
SELECT insert_user(1008, 'molica');
SELECT insert_user(1009, 'pg');
SELECT insert_user(1010, 'oracle');
```

5. 由于插入函数执行的是RUN ON ANY，即插入数据时随机选取数据节点，查看每个数据节点的数据如下：

```
pl_db0=> select * from users;
 userid |  name
--------+--------
   1001 | Sven
   1003 | Steve
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
   1008 | molica
   1009 | pg
(8 rows)

pl_db1=> select * from users;
 userid |  name
--------+--------
   1002 | Marko
   1010 | oracle
(2 rows)
```

> **说明** 通过查询可以发现10条数据分布在不同数据节点，由于10条数据太少，导致分布不均匀。

6. 在代理节点执行读取函数，由于执行的是RUN ON ALL，即代理节点返回所有数据节点查询结果，示例如下：

```
postgres=> SELECT USERID,NAME FROM GET_USER_NAME();
 userid |  name
--------+--------
   1001 | Sven
   1003 | Steve
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
   1008 | molica
   1009 | pg
   1002 | Marko
   1010 | oracle
(10 rows)
```

SQL复制测试步骤如下：

1. 在各个节点创建清理函数用于清理表users数据，示例如下：

```
pl_db0=> CREATE OR REPLACE FUNCTION trunc_user()
pl_db0-> RETURNS integer AS $$
pl_db0$>       truncate table users;
pl_db0$>       SELECT 1;
pl_db0$> $$ LANGUAGE SQL;
CREATE FUNCTION

pl_db1=> CREATE OR REPLACE FUNCTION trunc_user()
pl_db1-> RETURNS integer AS $$
pl_db1$>       truncate table users;
pl_db1$>       SELECT 1;
pl_db1$> $$ LANGUAGE SQL;
CREATE FUNCTION

postgres=> CREATE OR REPLACE FUNCTION trunc_user()
postgres-> RETURNS SETOF integer AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ALL;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

2. 在代理节点执行清理函数，示例如下：

```
postgres=> SELECT TRUNC_USER();
 trunc_user
------------
          1
          1
(2 rows)
```

3. 在代理节点创建插入函数，示例如下：

```
postgres=> CREATE OR REPLACE FUNCTION insert_user_2(i_id int, i_name text)
postgres-> RETURNS SETOF integer AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ALL;
postgres$> TARGET insert_user;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

4. 在代理节点插入4条测试记录，示例如下：

```
SELECT insert_user_2(1004, 'lottu');
SELECT insert_user_2(1005, 'rax');
SELECT insert_user_2(1006, 'ak');
SELECT insert_user_2(1007, 'jack');
```

5. 查看每个数据节点的数据，示例如下：

```
pl_db0=> select * from users;
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)

pl_db1=> select * from users;
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
```

> 说明 每个数据节点的数据都一样，说明数据复制成功。

6. 在代理节点查询时，只需要执行RUN ON ANY，即在任意一个数据节点读取数据即可，示例如下：

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name_2()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ANY ;
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION

postgres=> SELECT USERID,NAME FROM GET_USER_NAME_2();
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
```

CONNECT模式测试

使用CONNECT模式时，代理节点可以直接跨实例访问，示例如下：

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name_3()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>     CONNECT 'dbname=pl_db0 host=100.81.137.72 port=56789';
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION

postgres=> SELECT USERID,NAME FROM GET_USER_NAME_3();
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
```

11.模糊查询（pg_bigm）

pg_bigm是阿里云产品RDS Postgresql的一款插件，该插件提供了全文本搜索能力，允许创建一个二元语法（2-gram）的GIN索引来加速搜索过程。

前提条件

RDS PostgreSQL实例需为以下版本之一：

- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

与pg_trgm异同

pg_trgm是RDS Postgresql的另一款插件，使用3-gram的模型来实现全文本搜索。pg_bigm插件是在pg_trgm基础上继续开发的，两者的区别如下。

功能和特性	pg_trgm	pg_bigm
全文搜索的短语匹配方法	3-gram	2-gram
支持的索引类型	GIN和GIST	GIN
支持的全文本搜索操作符号	LIKE 、ILIKE 、~ 、~*	LIKE
非字母语言的全文本搜索	不支持	支持
带有1~2个字符的关键字的全文本搜索	慢	快
相似性搜索	支持	支持
最大可以索引的列大小	238,609,291字节（约228 MB）	107,374,180字节（约102 MB）

注意事项

- 建立GIN索引的列的长度不可以超过107,374,180字节（约102 MB）。
- 如果数据库中存储的内容语言是非ASCII，则建议将数据库的编码方式改为UTF8。

> 说明 查询当前数据库的编码方式命令为 `select pg_encoding_to_char(encoding) from pg_database where datname = current_database();` 。

基本操作

- 创建插件

```
postgres=> create extension pg_bigm;
CREATE EXTENSION
```

- 创建索引

```
postgres=> CREATE TABLE pg_tools (tool text, description text);
CREATE TABLE
postgres=> INSERT INTO pg_tools VALUES ('pg_hint_plan', 'Tool that allows a user to specify an opti
mizer HINT to PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_dbms_stats', 'Tool that allows a user to stabilize plan
ner statistics in PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_bigm', 'Tool that provides 2-gram full text search cap
ability in PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_trgm', 'Tool that provides 3-gram full text search capa
bility in PostgreSQL');
INSERT 0 1
postgres=> CREATE INDEX pg_tools_idx ON pg_tools USING gin (description gin_bigm_ops);
CREATE INDEX
postgres=> CREATE INDEX pg_tools_multi_idx ON pg_tools USING gin (tool gin_bigm_ops, description
gin_bigm_ops) WITH (FASTUPDATE = off);
CREATE INDEX
```

- 执行全文本搜索

```
postgres=> SELECT * FROM pg_tools WHERE description LIKE '%search%';
 tool  |                    description
---------+-----------------------------------------------------------------------
 pg_bigm | Tool that provides 2-gram full text search capability in PostgreSQL
 pg_trgm | Tool that provides 3-gram full text search capability in PostgreSQL
(2 rows)
```

- 使用 `=%` 操作符执行相似性搜索

```
postgres=> SET pg_bigm.similarity_limit TO 0.2;
SET
postgres=> SELECT tool FROM pg_tools WHERE tool =% 'bigm';
 tool
---------
 pg_bigm
 pg_trgm
(2 rows)
```

- 卸载插件

```
postgres=> drop extension pg_bigm;
DROP EXTENSION
```

插件常用函数

- likequery函数
  - 作用：生成可以被LIKE关键字识别的字符串。
  - 参数：1个请求参数，类型为字符串。
  - 返回值：可以被LIKE关键字识别的搜索字符串。
  - 实现原理：
    - 在关键词前后添加 `%` 符号。
    - 使用 `\` 来自动转义符号 `%` 。
  - 示例如下：

```
postgres=> SELECT likequery('pg_bigm has improved the full text search performance by 200%');
                        likequery
-----------------------------------------------------------------
 %pg\_bigm has improved the full text search performance by 200\%%
(1 row)

postgres=> SELECT * FROM pg_tools WHERE description LIKE likequery('search');
 tool  |                    description
---------+-----------------------------------------------------------------------
 pg_bigm | Tool that provides 2-gram full text search capability in PostgreSQL
 pg_trgm | Tool that provides 3-gram full text search capability in PostgreSQL
(2 rows)
```

- show_bigm函数
  - 作用：返回给定字符串的所有2-gram元素的集合。
  - 参数：1个请求参数，类型为字符串。
  - 返回值：数组，包含所有的2-gram元素。
  - 实现原理：
    - 在字符串前后添加空格字符。
    - 计算所有的2-gram子串。
  - 示例如下：

```
postgres=> SELECT show_bigm('full text search');
                        show_bigm
-----------------------------------------------------------------
 {" f"," s"," t",ar,ch,ea,ex,fu,"h ","l ",ll,rc,se,"t ",te,ul,xt}
(1 row)
```

- bigm_similarity函数
  - 作用：计算两个字符串的相似度。
  - 参数：2个请求参数，类型为字符串。
  - 返回值：浮点数，表示相似度。
  - 实现原理：
    - 统计两个字符串共有的2-gram元素。
    - 相似度范围是[0, 1]，0代表两个字符串完全不一样，1代表两个字符串一样。

    > 说明
    > - 由于计算2-gram时，会在字符串前后添加空格，于是 `ABC` 和 `B` 的相似度为0， `ABC` 和 `A` 的相似度为0.25。
    > - bigm_similarity函数是大小写敏感的，例如 `ABC` 和 `abc` 的相似度为0。

  - 示例如下：

```
postgres=> SELECT bigm_similarity('full text search', 'text similarity search');
 bigm_similarity
-----------------
       0.5714286
(1 row)

postgres=> SELECT bigm_similarity('ABC', 'A');
 bigm_similarity
-----------------
            0.25
(1 row)

postgres=> SELECT bigm_similarity('ABC', 'B');
 bigm_similarity
-----------------
               0
(1 row)

postgres=> SELECT bigm_similarity('ABC', 'abc');
 bigm_similarity
-----------------
               0
(1 row)
```

- pg_gin_pending_stats函数
  - 作用：返回GIN索引的pending list中页面和元组的个数。

- 参数：1个，GIN索引的名字或者OID。
- 返回值：2个，pending list中页面的数量和元组的数量。

> 说明 如果GIN索引创建时，指定参数FASTUPDATE为False，则该GIN索引不存在pending list，即返回结果为0。

- 示例如下：

```
postgres=> SELECT * FROM pg_gin_pending_stats('pg_tools_idx');
 pages | tuples
-------+--------
     0 |      0
(1 row)
```

插件行为控制

- pg_bigm.last_update

  该插件的最后更新日期，只读参数，无法修改。

  示例如下：

```
SHOW pg_bigm.last_update;
```

- pg_bigm.enable_recheck

  决定是否进行recheck。

  > 说明 建议您保持默认值（ON）以保证结果正确性。

  示例如下：

```
postgres=> CREATE TABLE tbl (doc text);
CREATE TABLE
postgres=> INSERT INTO tbl VALUES('He is awaiting trial');
INSERT 0 1
postgres=> INSERT INTO tbl VALUES('It was a trivial mistake');
INSERT 0 1
postgres=> CREATE INDEX tbl_idx ON tbl USING gin (doc gin_bigm_ops);
CREATE INDEX
postgres=> SET enable_seqscan TO off;
SET
postgres=> EXPLAIN ANALYZE SELECT * FROM tbl WHERE doc LIKE likequery('trial');
                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on tbl  (cost=20.00..24.01 rows=1 width=32) (actual time=0.020..0.021 rows=1 loop
s=1)
   Recheck Cond: (doc ~~ '%trial%'::text)
   Rows Removed by Index Recheck: 1
   Heap Blocks: exact=1
   ->  Bitmap Index Scan on tbl_idx  (cost=0.00..20.00 rows=1 width=0) (actual time=0.013..0.013 rows=
2 loops=1)
         Index Cond: (doc ~~ '%trial%'::text)
 Planning Time: 0.117 ms
 Execution Time: 0.043 ms
(8 rows)

postgres=>
postgres=> SELECT * FROM tbl WHERE doc LIKE likequery('trial');
        doc
----------------------
 He is awaiting trial
(1 row)

postgres=> SET pg_bigm.enable_recheck = off;
SET
postgres=> SELECT * FROM tbl WHERE doc LIKE likequery('trial');
         doc
--------------------------
 He is awaiting trial
 It was a trivial mistake
(2 rows)
```

- pg_bigm.gin_key_limit

  限制用于全文本搜索的2-gram元素的最大个数，默认为0，0代表使用所有的2-gram元素。

  > **说明** 如果发现使用所有的2-gram元素导致性能下降，可以调整该参数值，限制2-gram元素的个数来提高性能。

- pg_bigm.similarity_limit

  设置相似度阈值，相似度超过这个阈值的元组会做为相似性搜索的结果。

12.逻辑解码（wal2json）

RDS PostgreSQL提供wal2json插件，可以将逻辑日志文件输出为JSON格式供您查看。

前提条件

- 实例为RDS PostgreSQL 11、12。
- 已设置实例参数wal_level = logical。详情请参见设置实例参数。

背景信息

wal2json是逻辑解码插件，使用该插件可以访问由INSERT和UPDATE生成的元组，解析WAL中的内容。

wal2json插件会在每个事务中生成一个JSON对象。JSON对象中提供了所有新/旧元组，额外选项还可以包括事务时间戳、限定架构、数据类型、事务ID等属性。详情请参见通过SQL获取JSON对象。

通过SQL获取JSON对象

1. 通过DMS登录RDS数据库。
2. 执行如下命令建表及初始化插件。

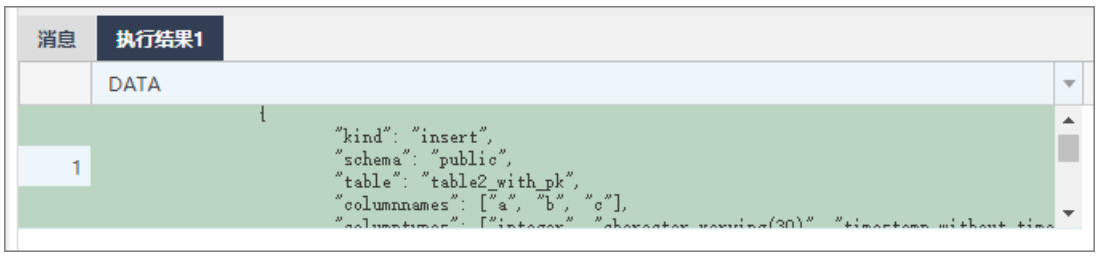
```
CREATE TABLE table2_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a, c));
CREATE TABLE table2_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);


SELECT 'init' FROM pg_create_logical_replication_slot('test_slot', 'wal2json');
```

3. 执行如下命令变更数据。

```
BEGIN;
INSERT INTO table2_with_pk (b, c) VALUES('Backup and Restore', now());
INSERT INTO table2_with_pk (b, c) VALUES('Tuning', now());
INSERT INTO table2_with_pk (b, c) VALUES('Replication', now());
DELETE FROM table2_with_pk WHERE a < 3;
INSERT INTO table2_without_pk (b, c) VALUES(2.34, 'Tapir');
UPDATE table2_without_pk SET c = 'Anta' WHERE c = 'Tapir';
COMMIT;
```

4. 执行如下命令输出JSON格式的日志信息。

```
SELECT data FROM pg_logical_slot_get_changes('test_slot', NULL, NULL, 'pretty-print', '1');
```



说明 如果需要停止输出并释放资源，请执行如下命令：

```
SELECT 'stop' FROM pg_drop_replication_slot('test_slot');
```

13.集成Elasticsearch（ZomboDB）

ZomboDB是一个PostgreSQL扩展插件，支持原生的访问方式，为PostgreSQL数据库带来了强大的文本索引和分析功能。

前提条件

RDS PostgreSQL实例版本为PostgreSQL 11。

背景信息

ZomboDB提供了一套全方位的查询语言，可以供您自由地查询关系型数据。此外，ZomboDB允许您创建ZomboDB类型的索引，此时ZomboDB完全接管远程的Elasticsearch，并负责文本搜索的事务正确性。

ZomboDB的优势在于允许您直接使用Elasticsearch的强大功能而不用处理同步、通信等问题。

插件的创建与删除

- 创建插件

```
CREATE EXTENSION zombodb;
```

- 删除插件

```
DROP EXTENSION zombodb;
```

示例

1. 创建一个表。

```
CREATE TABLE products (
    id SERIAL8 NOT NULL PRIMARY KEY,
    name text NOT NULL,
    keywords varchar(64)[],
    short_summary text,
    long_description zdb.fulltext,
    price bigint,
    inventory_count integer,
    discontinued boolean default false,
    availability_date date
);
```

2. 为表添加ZomboDB类型的索引。

```
CREATE INDEX idxproducts
          ON products
       USING zombodb ((products.*))
        WITH (url='localhost:9200/');
```

> 说明 WITH语句后跟随了Elasticsearch的地址，该地址指向了一个正在服务的Elasticsearch实例。

3. 使用ZomboDB格式的查询语句进行查询。

```
SELECT *
 FROM products
 WHERE products ==> '(keywords:(sports OR box) OR long_description:"wooden away"~5) AND pric
e:[1000 TO 20000]';
```

> 说明 ZomboDB格式的查询语句详细规则请参见ZomboDB官方文档。

14.SQL防火墙（sql_firewall）

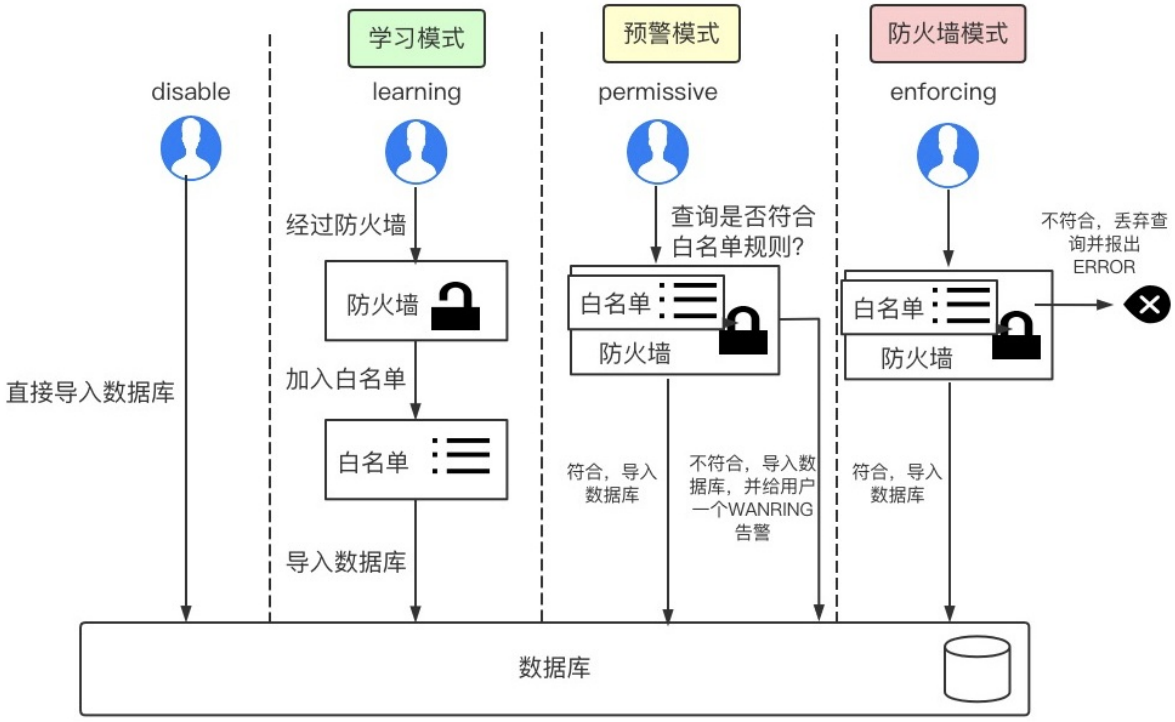
SQL防火墙是数据库层面的防火墙功能，可以防止恶意SQL注入。可以用来学习一些定义好的SQL规则，并将这些规则储存在数据库中作为白名单，学习完成后，可以限制用户执行这些定义规则之外的风险操作。

前提条件

RDS PostgreSQL实例需为以下版本之一：

- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

学习模式、预警模式与防火墙模式



SQL防火墙支持如下三种模式：

- 学习模式：防火墙记录用户执行的SQL，并添加到常用SQL白名单。
- 预警模式：防火墙对用户将执行的SQL进行判断，如果SQL不在白名单中，仍然会执行该SQL，但是会告警用户该SQL不在白名单中。
- 防火墙模式：防火墙对用户将执行的SQL进行判断，如果SQL不在白名单中，防火墙会拒绝执行该SQL并返回错误提示。

SQL防火墙的使用步骤

1. 打开防火墙的学习模式，让防火墙记录用户执行的SQL，并加入白名单。这个过程建议持续一段较长的时间，尽量让防火墙学习到所有可能执行的SQL。
2. 切换防火墙为预警模式，这个过程防火墙会对用户的一些不在白名单的SQL进行告警，用户结合自己的业务判断是否为风险SQL，如果这些SQL确实是用户需要的业务语句，则记录这些SQL，之后打开学习模式进行二次学习。

3. 经过前两步，用户常用SQL已经被记录完毕，打开防火墙模式，此时不在白名单的SQL均会被拒绝执行。

操作说明

- 创建插件

```
create extension sql_firewall;
```

- 删除插件

```
drop extension sql_firewall;
```

- 切换模式

  在控制台找到sql_firewall.firewall参数，修改参数值并重启实例。详情请参见设置实例参数。

  sql_firewall.firewall取值如下：

  - disable：关闭SQL防火墙
  - learning：学习模式
  - permissive：预警模式
  - enforcing：防火墙模式

- 功能函数
  - sql_firewall_reset()

    清空白名单。该函数只有在防火墙关闭模式下，rds_superuser权限的用户可执行该函数。
  - sql_firewall_stat_reset()

    清空统计信息。该函数只有在防火墙关闭模式下，rds_superuser权限的用户可执行该函数。

- 视图函数
  - sql_firewall.sql_firewall_statements

    展示目前数据库中所有的白名单及其被调用的次数。

```
postgres=# select * from sql_firewall.sql_firewall_statements;
   userid |  queryid   |           query           | calls
--------+------------+-------------------------------+-------
     10 | 3294787656 | select * from k1 where uid = ?; |     4
(1 row)
```

  - sql_firewall.sql_firewall_stat

    展示预警模式下的警告数量（sql_warning）和防火墙模式下的错误数量（sql_error）。

```
postgres=# select * from sql_firewall.sql_firewall_stat;
   sql_warning | sql_error
-------------+-----------
           2 |       1
(1 row)
```

示例

```
--预警模式

  postgres=# select * from sql_firewall.sql_firewall_statements;
  WARNING:  Prohibited SQL statement
   userid |  queryid   |           query            | calls
  --------+------------+--------------------------------+-------
      10 | 3294787656 | select * from k1 where uid = 1; |     1
  (1 row)

  postgres=# select * from k1 where uid = 1;
   uid |    uname
  -----+-------------
     1 | Park Gyu-ri
  (1 row)

  postgres=# select * from k1 where uid = 3;
   uid |   uname
  -----+-----------
     3 | Goo Ha-ra
  (1 row)

  postgres=# select * from k1 where uid = 3 or 1 = 1;
  WARNING:  Prohibited SQL statement
   uid |     uname
  -----+----------------
     1 | Park Gyu-ri
     2 | Nicole Jung
     3 | Goo Ha-ra
     4 | Han Seung-yeon
     5 | Kang Ji-young
  (5 rows)

--防火墙模式

  postgres=# select * from k1 where uid = 3;
   uid |   uname
  -----+-----------
     3 | Goo Ha-ra
  (1 row)
```

```
postgres=# select * from k1 where uid = 3 or 1 = 1;
ERROR:  Prohibited SQL statement
postgres=#
```

15.逻辑订阅故障转移（Failover Slot）

Failover Slot功能可以将所有的logical slot从主实例同步到备实例，从而实现logical slot的故障转移。

背景信息

使用逻辑订阅时如果不开启Failover Slot功能，当实例发生主备切换，逻辑订阅就会断开，因为slot不会随着主备切换转移到新主库上，除非手动重新创建slot，否则逻辑订阅无法重新连接。Failover Slot功能可以将所有的logical slot从主实例同步到备实例，避免逻辑订阅断开。

> **说明** 当前只支持logical slot的故障转移，physical slot暂不支持。

通过设置参数rds_failover_slot_mode（即时生效，无需重启实例）可以使用Failover Slot功能，取值如下：

- sync：同步模式
- async：异步模式
- off：关闭

如何设置参数请参见设置实例参数。

关于两种模式的详细说明请参见下文。

同步模式

同步模式可以尽量保证主备切换后逻辑订阅不丢失数据，但是极端情况会发生逻辑订阅的数据丢失。

如果备实例长时间断开连接，或者备实例重搭，则会导致这段时间内主备延迟较大，如果这时候发生主备切换，不能保证逻辑订阅不丢失数据。此时丢失的数据，除了主备切换本身造成的数据丢失外，还可能会丢失主备切换后新主实例上新增的部分数据。

想要避免这种极端情况，可以设置参数rds_priority_replication_force_wait为on（默认为off），修改该参数即时生效，无需重启实例。设置后，在备实例长时间断开连接，或者备实例重搭等情况时，主实例会一直等待备实例重新连接或者重搭完成，在此之前不会发送数据给逻辑订阅客户端，但是这样会降低逻辑订阅的可用性，不推荐这样操作。

异步模式

异步模式可以保证实例主备切换后，逻辑订阅不丢失数据，但是可能会给逻辑订阅客户端发送重复数据。

除非您可以接受异步模式时可能存在的数据不一致问题，否则推荐使用同步模式。

16.并发控制（pg_concurrency_control）

RDS PostgreSQL提供pg_concurrency_control插件，用于对SQL进行并发控制。

前提条件

RDS PostgreSQL实例版本为PostgreSQL 10。

参数说明

> 说明 暂不支持在控制台修改以下参数，您可以提交工单申请修改。

参数	默认值	说明
pg_concurrency_control.query_concurrency	0	设置Select类型SQL并发控制的排队个数限制。取值范围为0~1024，默认值为0，表示关闭Select类型SQL并发控制。
pg_concurrency_control.bigquery_concurrency	0	设置慢查询类型SQL并发控制的排队个数限制。取值范围为0~1024，默认值为0，表示关闭慢查询类型SQL并发控制。 您可以使用 `hint "/*+bigsql*/"` 的方式来指定一个请求为慢查询。例如： `/*+bigsql*/select * from test;` 此时 `select * from test;` 被认为是一个慢查询。
pg_concurrency_control.transaction_concurrency	0	设置事务块并发控制的排队个数限制。取值范围为0~1024，默认值为0，表示关闭事务块并发控制。
pg_concurrency_control.autocommit_concurrency	0	设置DML类型SQL并发控制的排队个数限制。取值范围为0~1024，默认值为0，表示关闭DML类型SQL并发控制。
pg_concurrency_control.control_timeout	1秒	设置Select类型SQL、DML类型SQL和事务块的并发控制排队等待时间。最小值为30毫秒（ms），最大值为3秒（s）。
pg_concurrency_control.bigsql_control_timeout	1秒	设置慢查询并发控制排队等待时间。最小值为30毫秒（ms），最大值为3秒（s）。
pg_concurrency_control.timeout_action	TCC_break	设置Select类型SQL、DML类型SQL和事务块的并发控制等待超时后的行为。取值： • TCC_break：跳过等待继续执行。 • TCC_rollback：超时后报错，事务回滚。 • TCC_wait：超时后重置时间戳继续等待。

参数	默认值	说明
pg_concurrency_control.bigsql_timeout_action	TCC_wait	设置慢查询并发控制等待超时后的行为。取值： • TCC_break：跳过等待继续执行。 • TCC_rollback：超时后报错，事务回滚。 • TCC_wait：超时后重置时间戳继续等待。

使用方法

1. 使用如下命令创建插件：

```
create extension pg_concurrency_control;
```

2. 设置并发控制排队个数限制大于0，即开启插件排队功能。

   例如设置pg_concurrency_control.query_concurrency=10，即开启Select类型SQL并发控制功能。其余功能开启方式类似。

   > 说明 暂不支持在控制台修改参数，您可以提交工单申请修改。

使用示例

对自定义SQL操作进行并发控制。

1. 使用如下命令查看排队视图：

```
select * from pg_concurrency_control_status();
```

   系统输出类似如下结果：

```
 autocommit_count | bigquery_count | query_count | transaction_count
------------------+----------------+-------------+-------------------
                0 |              0 |           0 |                 0
(1 row)
```

2. 设置pg_concurrency_control.query_concurrency大于0，例如10。
3. 执行慢查询语句：

```
/*+ bigsql */ select pg_sleep(10);
```

4. 再次查看排队视图：

```
select * from pg_concurrency_control_status();
```

   系统输出类似如下结果：

```
 autocommit_count | bigquery_count | query_count | transaction_count
------------------+----------------+-------------+-------------------
                0 |              1 |           0 |                 0
(1 row)
```

> **说明** 慢查询执行完毕后，排队信息会自动清空。