阿里云
云数据库 POLARDB

POLARDB PostgreSQL数据库

文档版本：20200109

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. **阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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。
	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。
	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。	执行`cd /d C:/window`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。	`switch {active|stand}`

目录

1 概述

POLARDB是阿里云自研的下一代关系型云数据库，兼容MySQL、PostgreSQL、Oracle引擎，存储容量最高可达100TB，单库最多可扩展到16个节点，适用于企业多样化的数据库应用场景。

POLARDB采用存储和计算分离的架构，所有计算节点共享一份数据，提供分钟级的配置升降级、秒级的故障恢复、全局数据一致性和免费的数据备份容灾服务。POLARDB既融合了商业数据库稳定可靠、高性能、可扩展的特征，又具有开源云数据库简单开放、自我迭代的优势。

基本概念

- **集群：一个集群包含一个主节点以及最多15个只读节点（最少一个，用于提供Active-Active高可用）。集群ID以pc开头（代表POLARDB cluster）。**
- **节点：一个独立占用物理内存的数据库服务进程。节点ID以pi开头（代表POLARDB instance）。**
- **数据库：在节点下创建的逻辑单元，一个节点可以创建多个数据库，数据库在节点内的命名唯一。**
- **地域和可用区：地域是指物理的数据中心。可用区是指在同一地域内，拥有独立电力和网络的物理区域。更多信息请参考阿里云全球基础设施。**

控制台

阿里云提供了简单易用的Web控制台，方便您操作阿里云的各种产品和服务，包括云数据库POLARDB。在控制台上，您可以创建、连接和配置POLARDB数据库。

关于控制台的界面介绍，请参考阿里云管理控制台。

POLARDB控制台地址： *https://polardb.console.aliyun.com*

2 POLARDB PostgreSQL快速入门

快速入门旨在介绍如何创建POLARDB PostgreSQL集群、进行基本设置以及连接数据库集群，使您能够了解从购买POLARDB到开始使用的流程。

使用流程

通常，从购买POLARDB（创建新集群）到可以开始使用，您需要完成如下操作。

1. **创建***POLARDB PostgreSQL***数据库集群**
2. *#unique_6*
3. **创建数据库账号**
4. **查看连接地址**
5. **连接数据库集群**

3 数据迁移

3.1 数据迁移方案概览

云数据库POLARDB提供了多种数据迁移方案，可满足不同上云、迁云的业务需求，使您可以在不影响业务的情况下平滑将数据库迁移至阿里云云数据库POLARDB上面。

通过使用阿里云数据传输服务（DTS），您可以实现POLARDB的结构迁移、全量迁移等。

数据迁移

使用场景	文档链接
从RDS迁移至POLARDB	从RDS for PostgreSQL迁移至POLARDB for PostgreSQL
从自建数据库迁移至POLARDB	从自建PostgreSQL迁移至POLARDB for PostgreSQL

3.2 从自建PostgreSQL迁移至POLARDB for PostgreSQL

本文介绍通过`pg_dumpall`、`pg_dump`和`pg_restore`命令将自建PostgreSQL数据库迁移至POLARDB for PostgreSQL中。

迁移的源库为RDS for PostgreSQL实例时，请参考从RDS for PostgreSQL迁移至POLARDB for PostgreSQL。

前提条件

POLARDB for PostgreSQL实例的存储空间应大于自建PostgreSQL数据库的存储空间。

注意事项

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

准备工作

1. 创建一个Linux操作系统的ECS实例，本案例使用的ECS为Ubuntu 16.04 64位操作系统。详情请参考创建ECS实例。

   说明:

   - 要求ECS实例和迁移的目标POLARDB for PostgreSQL实例处于同一个专有网络。
   - 可创建一个按量付费的ECS实例，迁移完成后释放实例。

2. 在ECS实例中安装PostgreSQL，以便执行数据恢复的命令。详情请参考*PostgreSQL*官方文档。

> **说明：**
> 请确保安装的PostgreSQL数据库版本与自建PostgreSQL数据库版本一致。

操作步骤一 备份自建数据库

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

1. 在自建PostgreSQL数据库服务器上执行以下命令，备份数据库中的所有角色信息。

```
pg_dumpall -U <username> -h <hostname> -p <port> -r -f <filename>
```

参数说明：

- <username>：登录自建PostgreSQL数据库的账号。
- <hostname>：自建PostgreSQL数据库的连接地址，本机可使用localhost。
- <port>：数据库服务的端口号。
- <filename>：生成的备份文件名称。

示例：

```
pg_dumpall -U postgres -h localhost -p 5432 -r -f roleinfo.sql
```

2. 命令行提示Password:时，输入数据库账号对应的密码，开始备份数据库中的所有角色信息。
3. 使用vim命令将角色信息备份文件中的SUPERUSER替换为polar_superuser。

> **说明：**
> 如果角色信息备份文件中没有SUPERUSER信息，可跳过本步骤。

```
-- PostgreSQL database cluster dump
--

SET default_transaction_read_only = off;

SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;

--
-- Roles
--

CREATE ROLE data1;
ALTER ROLE data1 WITH NOSUPERUSER INHERIT CREATEROLE CREATEDB LOGIN NOREPLICATION NOBYPASSRLS PASSWORD 'md[redacted]';
CREATE ROLE manisha;
ALTER ROLE manisha WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN NOREPLICATION NOBYPASSRLS PASSWORD 'md[redacted]';
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYPASSRLS PASSWORD 'md5[redacted]';
CREATE ROLE testuser;
ALTER ROLE testuser WITH NOSUPERUSER INHERIT NOCREATEROLE CREATEDB LOGIN NOREPLICATION NOBYPASSRLS;



--
-- PostgreSQL database cluster dump complete
--
```

4. 在自建PostgreSQL数据库服务器上执行以下命令，备份数据库中的数据。

```
pg_dump -U <username> -h <hostname> -p <port> <dbname> -Fd -j <njobs
> -f <dumpdir>
```

参数说明：

- <username>：登录自建PostgreSQL数据库的账号。
- <hostname>：自建PostgreSQL数据库的连接地址，本机可使用localhost。
- <port>：数据库服务的端口号。
- <dbname>：要备份的数据库名。
- <njobs>：同时执行备份作业的并发数。

  说明：

  - 参数<njobs>可减少转储的时间，但也会增加数据库服务器的负载。
  - 如果您的自建PostgreSQL数据库是9.2以前的版本，您还需要指定--no-synchronized-snapshots参数。
- <dumpdir>：生成的备份文件所属目录。

示例：

```
pg_dump -U postgres -h localhost -p 5432 mytestdata -Fd -j 5 -f
postgresdump
```

5. 命令行提示Password:时，输入数据库账号对应的密码，数据库开始备份。
6. 等待备份完成，PostgreSQL数据库数据将备份至指定的目录中，本案例为*postgresdump*。

操作步骤二 数据迁移至POLARDB for PostgreSQL

1. 将备份文件所属的目录上传至ECS实例中。

   说明：

   包含角色信息备份文件和数据库备份文件。

2. 在ECS上执行以下命令，将角色信息备份文件中的角色信息迁移至POLARDB for PostgreSQL实例中。

```
psql -U <username> -h <hostname>  -p <port> -d <dbname>  -f <
filename>
```

参数说明：

- <username>：登录POLARDB for PostgreSQL数据库的账号。
- <hostname>：POLARDB for PostgreSQL实例的主地址（私网）。
- <port>：数据库服务的端口号，默认为1921。
- <dbname>：连接的数据库的名称，默认为postgres。
- <filename>：角色信息备份文件名。

```
psql -U gctest -h pc-xxxxxxxx.pg.polardb.cn-qd-pldb1.rds.aliyuncs.
com -d postgres -p 1921 -f roleinfo.sql
```

3. 命令行提示Password:时，输入数据库账号对应的密码，角色信息开始导入。
4. 在ECS上执行以下命令，将数据库数据恢复至POLARDB for PostgreSQL实例中。

```
pg_restore -U <username> -h <hostname> -p <port> -d <dbname> -j <
njobs> <dumpdir>
```

参数说明：

- <username>：登录POLARDB for PostgreSQL数据库的账号。
- <hostname>：POLARDB for PostgreSQL实例的主地址（私网），详情请参考#unique_14。
- <port>：数据库服务的端口号，默认为1921。
- <dbname>：连接并直接恢复到的目标数据库名。

  说明：
  目标数据库需已存在，如不存在请在目标实例中创建该数据库。
- <njobs>：同时执行数据恢复作业的并发数。

  说明：

此选项可减少数据恢复的时间，但也会增加数据库服务器的负载。

- <dumpdir>：备份文件所在目录。

示例：

```
pg_restore -U gctest -h pc-mxxxxxxxx.pg.polardb.cn-qd-pldb1.rds.
aliyuncs.com -p 1921 -d mytestdata -j 6 postgresdump
```

5. 命令行提示 `Password:` 时，输入数据库账号对应的密码，数据开始迁移。

说明：

如果忘记密码，请参考*#unique_15/unique_15_Connect_42_section_ckb_hpq_tdb*。

等待数据迁移完成即可。

3.3 从RDS for PostgreSQL迁移至POLARDB for PostgreSQL

本文介绍通过`pg_dump`和`pg_restore`命令将自建PostgreSQL数据库迁移至POLARDB for PostgreSQL中。

迁移的源库为自建PostgreSQL数据库时，请参考**从自建***PostgreSQL***迁移至***POLARDB for PostgreSQL*。

前提条件

POLARDB for PostgreSQL实例的存储空间应大于RDS for PostgreSQL实例的存储空间。

注意事项

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

准备工作

1. 创建一个Linux操作系统的ECS实例，本案例使用的ECS为Ubuntu 16.04 64位操作系统。详情请参考**创建***ECS***实例**。

说明：

- 要求ECS实例和迁移的目标POLARDB for PostgreSQL实例处于同一个专有网络。
- 可创建一个按量付费的ECS实例，迁移完成后释放实例。

2. 在ECS实例中安装PostgreSQL，以便执行数据恢复的命令。详情请参考*PostgreSQL***官方文档**。

说明：

请确保安装的PostgreSQL数据库版本与自建PostgreSQL数据库版本一致。

操作步骤一 备份RDS for PostgreSQL数据库

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

1. **在ECS上执行以下命令，备份数据库中的数据。**

```
pg_dump -U <username> -h <hostname> -p <port> <dbname> -Fd -j <njobs
> -f <dumpdir>
```

参数说明：

- **<username>：登录自建PostgreSQL数据库的账号。**
- **<hostname>：自建PostgreSQL数据库的连接地址，本机可使用**`localhost`**。**
- **<port>：数据库服务的端口号。**
- **<dbname>：指定要连接的数据库的名称，默认为**`postgres`**。**
- **<njobs>：同时执行备份作业的并发数。**

  说明：

  - **参数<njobs>可减少转储的时间，但也会增加数据库服务器的负载。**
  - **如果您的自建PostgreSQL数据库是9.2以前的版本，您还需要指定**`--no-synchronized-snapshots`**参数。**
- **<dumpdir>：生成的备份文件所属目录。**

示例：

```
pg_dump -U postgres -h localhost -p 5432 postgres -Fd -j 5 -f
postgresdump
```

2. **命令行提示**`Password:`**时，输入数据库账号对应的密码，数据库开始备份。**
3. **等待备份完成，PostgreSQL数据库数据将备份至指定的目录中，本案例为***postgresdump***。**

操作步骤二 数据迁移至POLARDB for PostgreSQL

1. **在ECS上连接POLARDB for PostgreSQL数据库。**

```
psql -U <username> -h <hostname> -p <port> -d <dbname>
```

参数说明：

- **<username>：登录POLARDB for PostgreSQL数据库的账号。**
- **<hostname>：POLARDB for PostgreSQL实例的主地址（私网），详情请参考**#unique_14。
- **<port>：数据库服务的端口号，默认为1921。**
- **<dbname>：要连接的数据库名称。**

示例：

```
psql -h pc-mxxxxxxxx.pg.polardb.cn-qd-pldb1.rds.aliyuncs.com -p 3433
 -d postgres -U gctest
```

2. **根据源RDS for PostgreSQL实例数据库中的角色信息，在目标POLARDB for PostgreSQL实例中创建角色信息，并对数据恢复的目标数据库进行授权，详情请参考官方文档** *CREATE ROLE*和*GRANT*。
3. **在ECS上执行以下命令，将数据库数据迁移至POLARDB for PostgreSQL实例中。**

```
pg_restore -U <username> -h <hostname> -p <port> -d <dbname> -j <
njobs> <dumpdir>
```

参数说明：

- **<username>：登录POLARDB for PostgreSQL数据库的账号。**
- **<hostname>：POLARDB for PostgreSQL实例的主地址（私网）。**
- **<port>：数据库服务的端口号，默认为1921。**
- **<dbname>：连接并直接恢复到的目标数据库名。**

> **说明：**
> **目标数据库需已存在，如不存在请在目标实例中创建该数据库。**

- **<njobs>：同时执行数据恢复作业的并发数。**

> **说明：**

此选项可减少数据恢复的时间，但也会增加数据库服务器的负载。

- <dumpdir>：备份文件所在目录。

示例：

```
pg_restore -U gctest -h pc-mxxxxxxxx.pg.polardb.cn-qd-pldb1.rds.
aliyuncs.com -p 1921 -d postgres -j 6 postgresdump
```

4. 命令行提示`Password:`时，输入数据库账号对应的密码，数据开始迁移。

说明：

如果忘记密码，请参考*#unique_16/unique_16_Connect_42_section_ckb_hpq_tdb*。

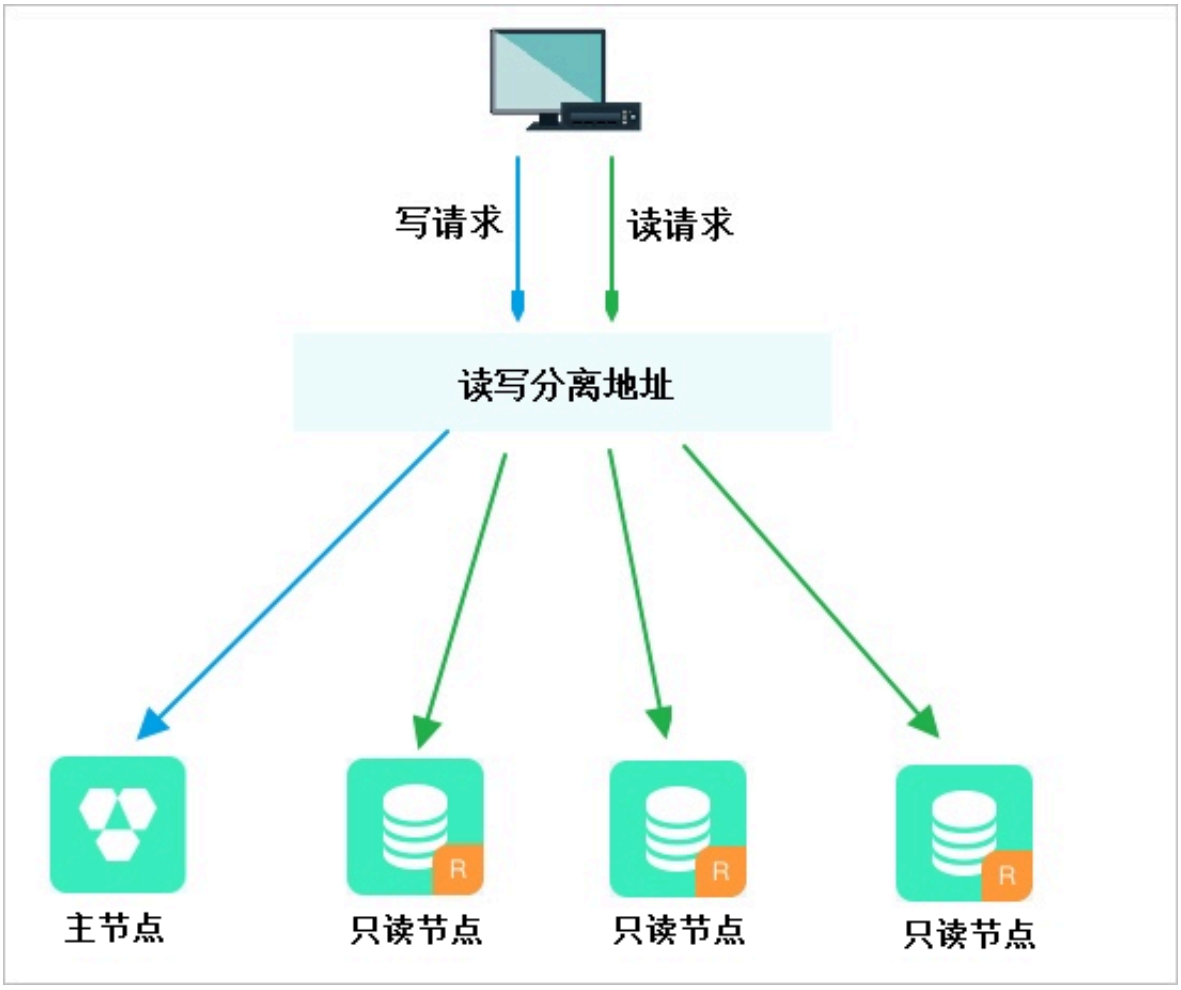
等待数据迁移完成即可。

4 读写分离

POLARDB for PostgreSQL集群自带读写分离功能，通过一个集群地址（读写分离地址）实现读写请求的自动转发。

背景信息

在对数据库有少量写请求，但有大量读请求的应用场景下，单个节点可能无法承受读取压力，甚至对业务产生影响。使用集群地址（读写分离地址），就可以使写请求自动转发到主节点，读请求自动转发到各个只读节点。从而实现读取能力的弹性扩展，满足大量的数据库读取需求。



功能优势

- **统一读写分离地址，方便维护。**

  您未使用集群地址（读写分离地址）时，需要在应用程序中分别配置主节点和每个只读节点的连接地址，才能实现将写请求发往主节点而将读请求发往只读节点。POLARDB提供一个集群地址（读写分离地址），您连接该地址后即可对主节点和只读节点进行读写操作，读写请求被自动

转发到对应节点，可降低维护成本。同时，您只需添加只读节点的个数，即可不断扩展系统的处理能力，应用程序无需做任何修改。

- Session级别的读一致性。

  当客户端通过读写分离建立与后端的连接后，读写分离中间件会自动与主节点和各个只读节点建立连接。在同一个连接内（同一个session 内），读写分离中间件会根据各个数据库节点的数据同步程度，来选择合适的节点，在保证数据正确的基础上（写操作之后的读有正确的结果），实现读写请求的负载均衡。

- 扩展查询（prepare语句/命令）的负载均衡。

  读写分离中间件会自动根据excute语句/命令中的信息来找到之前执行prepare语句/命令的数据库节点，从而做到扩展查询的负载均衡。

- 支持原生高安全链路，提升性能。

  如果您在云上自行搭建代理层实现读写分离，数据在到达数据库之前需要经历多个组件的语句解析和转发，对响应延迟有较大的影响。而 POLARDB读写分离中间件隶属于集群组件，相比外部组件而言，能够有效降低延迟，提升处理速度。

- 节点健康检查，提升数据库系统的可用性。

  读写分离模块将自动对主节点和只读节点进行健康检查，当发现某个节点出现宕机或者延迟超过阈值时，将不再分配读请求给该节点，读写请求在剩余的健康节点间进行分配。以此确保单个只读节点发生故障时，不会影响应用的正常访问。当节点被修复后，该节点会自动被加入请求分配体系内。

功能限制

- 暂不支持如下命令或功能：
  - 不支持Replication-mode方式进行建连，即不支持通过读写分离地址自行搭建主备复制集群。如需自行搭建主备复制集群，请使用主节点的连接地址。
  - 不支持临时表的ROWTYPE。

    ```
    create temp table fullname (first text, last text);
    select '(Joe,von Blow)'::fullname, '(Joe,d''Blow)'::fullname;
    ```

  - 不支持函数中创建临时资源。
    - 函数中创建临时表，然后再执行对临时表的查询SQL可能会收到表不存在的错误信息。
    - 函数中带有prepare语句，后续execute时可能会收到statement name不存在的错误信息。

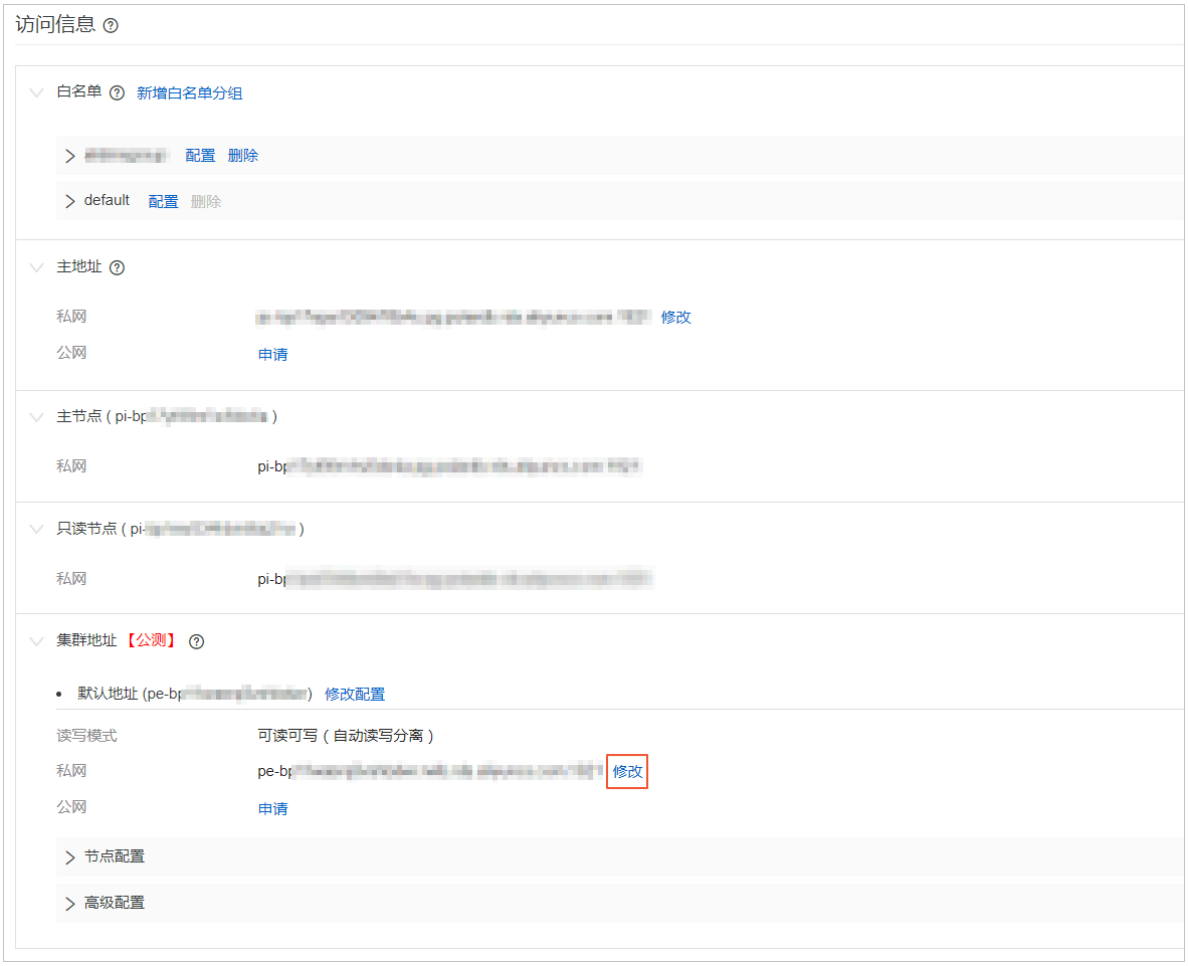
- **路由相关限制：**
  - **多语句（multi-statement）会路由到主节点，且之后本连接内的所有请求均路由到主节点。**
  - **大的请求报文（大于等于16MB）会路由到主节点，且之后本连接内的所有请求均路由到主节点。**
  - **事务中的请求都路由到主节点，事务退出后，恢复负载均衡。**
  - **所有使用函数（除聚合函数，例如，count、sum）的语句，会路由到主节点。**

申请或修改集群地址

1. **登录*POLARDB*控制台。**
2. **在控制台左上角，选择地域。**
3. **单击目标集群ID。**
4. **在基本信息页面的访问信息区域，找到集群地址。**
5. **单击申请，在弹出的对话框中单击确认，刷新后即可看到集群私网地址。**

> **说明:**
> **存量集群若未申请集群读写分离连接地址的需要手动申请集群私网地址，新购集群自动开通集群私网地址。若已有集群地址，请直接查看第6步。**

6. 单击修改，设置新的集群私网地址，并单击提交。



下一步

连接数据库集群

> **说明:**
> 申请集群地址后，您只需在应用中配置该集群地址，即可实现自动读写分离。

相关API

API	描述
#unique_18	创建POLARDB集群的公网地址。
#unique_19	查询POLARDB集群的地址信息。
#unique_20	修改POLARDB集群地址属性。
#unique_21	修改POLARDB集群默认访问地址前缀。
#unique_22	释放POLARDB集群地址（除了自定义集群地址的私网地址）。

5 待处理事件

当POLARDB出现待处理事件时，会在控制台提醒您及时处理。

POLARDB运维事件（数据库软件升级、硬件维护与升级）除了在短信、语音、邮件或站内信通知之外，还会在控制台进行通知。您可以查看具体的事件类型、任务ID、集群名称、切换时间等，也可以手动修改切换时间。

前提条件

有未处理的运维事件。

说明:

您可以在控制台左侧导航栏的待处理事件看到提醒。

修改切换时间

1. **登录*POLARDB*控制台。**
2. **在左侧导航栏单击待处理事件。**

说明:

强制要求预约时间的运维事件会弹窗提醒，请尽快完成预约。

3. **在待处理事件页面选择相应的事件类型。**

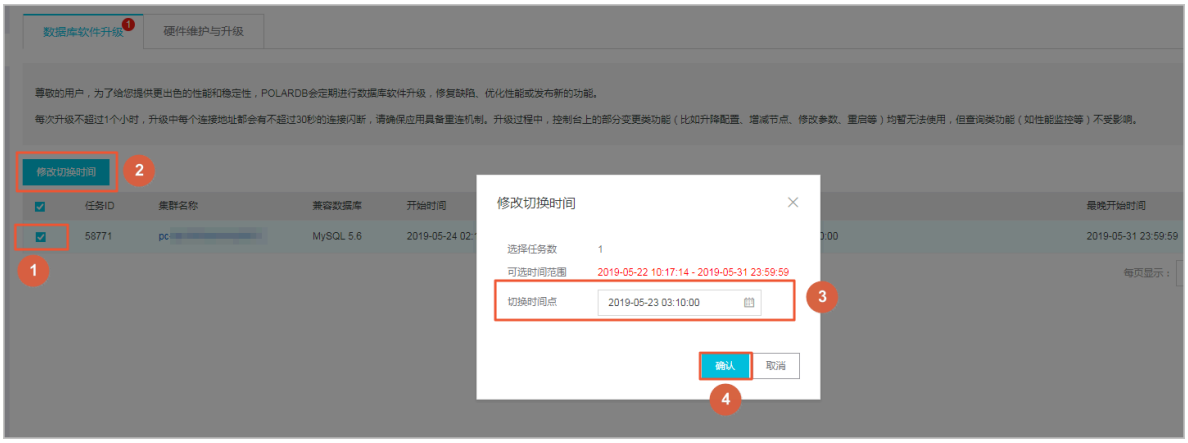
说明:

不同的事件类型页面会有不同的通知信息。

4. **在下方事件列表查看事件的详细信息，如需修改切换时间，请在左侧勾选对应的实例，然后单击修改切换时间，在弹出的对话框中设置时间并单击确认。**

说明:

切换时间不能晚于最晚操作时间。



历史事件

您可以在左侧导航栏的历史事件里查看已完成的事件。

6 设置集群白名单

创建POLARDB PostgreSQL数据库集群后，您需要设置POLARDB集群的IP白名单，并创建集群的初始账号，才能连接和使用该集群。

注意事项

- **默认情况下，IP白名单只包含IP地址127.0.0.1，表示任何IP地址均无法访问该数据库集群。**
- **若将IP白名单设置为%或者0.0.0.0/0，表示允许任何IP地址访问数据库集群。该设置将极大降低数据库的安全性，如非必要请勿使用。**
- **POLARDB暂不支持自动获取VPC中的ECS内网IP以供您选择，请手动填写需要访问POLARDB的ECS内网IP。**

设置白名单

1. **登录*POLARDB*控制台。**
2. **在页面左上角，选择实例所在地域。**
3. **单击目标集群ID，进入页面。**
4. **单击。**
5. **在集群白名单页面，可以配置已有白名单或新增IP白名单。**

   - **单击栏中的，配置该IP白名单。**
   - **单击，可以新增IP白名单。**
6. **填写白名单，单击。**
   - **如果您的ECS服务器需要访问POLARDB，可在ECS实例详情页面配置信息区域，查看ECS服务器的IP地址，然后填写到白名单中。**

> **说明:**

如果ECS与POLARDB位于同一地域（例如，华东1），填写ECS的私网IP地址；如果ECS与POLARDB位于不同的地域，填写ECS的公网IP地址，或者将ECS迁移到POLARDB所在地域后填写ECS私网IP地址。

- 如果您本地的服务器、电脑或其它云服务器需要访问POLARDB，请将其IP地址添加到白名单中。

下一步

设置集群白名单以及创建数据库账号后，您就可以连接数据库集群，对数据库进行操作。

- 创建数据库账号
- 连接数据库集群

常见问题

1. 已添加ECS的IP地址到IP白名单中，但是还是无法访问。

   答：

   a. 确认IP白名单是否正确。如果是通过内网地址访问，需添加ECS的私网IP地址。如果是通过公网地址进行访问，需添加ECS的公网IP地址。

   b. 确认网络类型是否一致。如果ECS实例的网络类型是经典网络，可参考经典网络迁移到专有网络方案将ECS实例迁移至POLARDB所在的专有网络。

   说明：

   如果该ECS 还要访问其他经典网络内网资源，请勿操作，因为迁移后会无法访问经典网络。

   或者通过*Classlink*打通经典网络到专有网络的网络。

   c. 确认是否位于同一个VPC。如果不是，需要重新购买一个POLARDB，或者通过云企业网来打通两个VPC网络实现访问。

2. 什么原因导致公网连接失败?

   答：

   a. 如果是ECS通过公网地址进行访问，请确认添加的是ECS的公网IP地址，而不是私网IP地址。

   b. IP白名单设置为0.0.0.0/0，然后尝试访问，如果能成功访问，表示IP白名单中之前填写的公网地址错误。请参考查看连接地址确定真正的公网地址。

3. 如何实现内网连接?

答：ECS和POLARDB内网访问需要满足以下条件：

- **相同地域。**
- **相同网络类型，如果是VPC，需要在相同VPC下。**
- **ECS内网IP在POLARDB集群IP白名单中。**

4. 如何限制某个用户只能从特定的IP地址访问POLARDB?

答：可以创建高权限账号，然后使用高权限账号对普通账号限定访问IP。

```
CREATE USER 'alitest'@'192.168.1.101' ;

select * from mysql.user where user='alitest';
```

相关API

API	描述
#unique_25	**查看允许访问数据库集群的IP名单。**
#unique_26	**修改允许访问数据库集群的IP名单。**

7 计费

7.1 按小时付费转包年包月

您可以根据需求将后付费（按小时付费）的集群转变为预付费（包年包月）的计费方式。本操作对集群的运行不会有任何影响。

> **说明:**
> **历史规格不支持直接转包年包月，请先*#unique_29*，然后再转包年包月。**

注意事项

包年包月的集群无法转成按小时付费的集群。在您将集群转为包年包月前请务必考虑清楚，以免造成资源浪费。

前提条件

- **集群状态为运行中。**
- **集群没有未完成的按小时付费转包年包月的订单。如果有，需要先在订单管理页面支付或作废该订单。**

操作步骤

1. **登录*POLARDB*控制台。**
2. **选择集群所在的地域。**
3. **找到目标集群，在操作列中选择... > 转包年包月。**

4. **选择购买时长，勾选云数据库POLARDB服务协议，单击去开通，根据提示完成支付。**

> **说明:**
> - **本操作会产生一个新购订单，只有完成了订单的支付，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，导致您无法新购集群或再次执行转包年包月。此时需支付或作废该订单。**

7.2 手动续费集群

您可以通过POLARDB控制台或续费管理控制台进行手动续费。通过续费管理控制台还可以为多个实例进行批量续费。

> **说明:**
> **按小时付费集群没有到期时间，不涉及续费操作。**

方法一：POLARDB控制台续费

1. **进入*POLARDB*控制台。**
2. **在左上角选择地域，即可显示您账号在该地域的所有集群。**
3. **找到目标集群，在右侧选择... > 续费。**

4. **选择续费时长并勾选服务协议，单击去支付完成支付即可。**

方法二：续费管理控制台续费

1. **进入*POLARDB*控制台。**
2. **在控制台右上方，选择费用 > 续费管理。**

3. **在控制台左上角单击体验新版，切换到新版控制台。**

4. 通过搜索过滤功能在手动续费页签中找到目标集群，您可以单个续费或批量续费：

- 单个续费

a. 在目标集群右侧单击续费。

说明：

- 示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到云数据库POLARDB，然后进行续费操作。

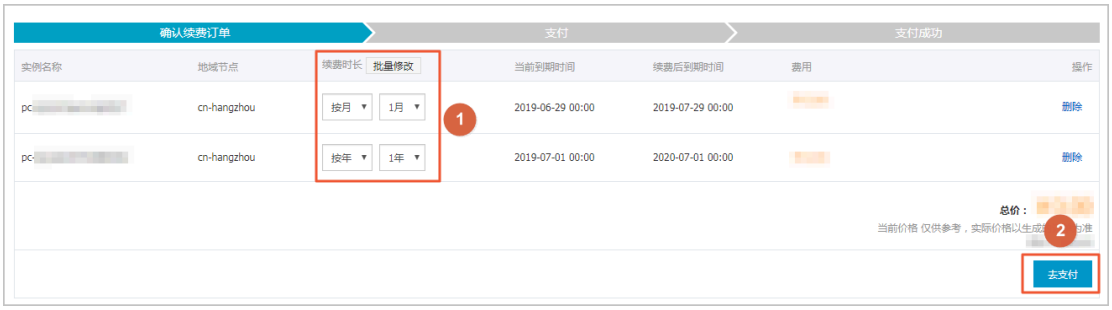
- 如果目标集群在自动续费或到期不续费页签中，您可以单击恢复手动续费，在弹出的对话框中单击确定即可恢复为手动续费。

b. 选择续费时长并勾选服务协议，单击去支付完成支付即可。

· 批量续费

a. 勾选目标集群，单击下方批量续费。

b. 选择每个集群的续费时长，单击去支付完成支付即可。



自动续费

开通自动续费可以免去您定期手动续费的烦恼，且不会因为忘记续费而导致业务中断。详情请参见自动续费集群。

7.3 自动续费集群

包年包月集群有到期时间，如果到期未续费，会导致业务中断甚至数据丢失。开通自动续费可以免去您定期手动续费的烦恼，且不会因为忘记续费而导致业务中断。

说明:

按小时付费集群没有到期时间，不涉及续费操作。

注意事项

· 自动续费将于集群到期前9天开始扣款，支持现金及代金券扣款，请保持账户余额充足。

- 若您在自动扣款日期前进行了手动续费，则系统将在下一次到期前进行自动续费。
- 自动续费功能于次日生效。若您的集群将于次日到期，为避免业务中断，请手动进行续费，详细步骤请参见手动续费集群。

购买集群时开通续费

> **说明:**
> 开通自动续费后，系统将根据您的购买时长为周期进行自动续费。例如，如果您购买了3个月的集群并勾选了自动续费，则每次自动续费时会缴纳3个月的费用。

在创建新集群时时，可以勾选自动续费。

购买集群后开通自动续费

> **说明:**
> 开通自动续费后，系统将根据您选择的续费周期进行自动续费。例如，如果您选择了3个月的续费周期，则每次自动续费时会缴纳3个月的费用。

1. 登录*POLARDB*管理控制台。
2. 在控制台右上方，选择费用 > 续费管理。

3. 在控制台左上角单击体验新版，切换到新版控制台。

4. 通过搜索过滤功能在手动续费或到期不续费页签中找到目标集群，您可以单个开通或批量开通：

- 单个开通

a. 单击右侧开通自动续费。

说明：

示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到云数据库POLARDB，然后开通自动续费。

b. 在弹出的对话框中，选择自动续费周期，单击开通自动续费。

- **批量开通**

  勾选目标集群，单击下方开通自动续费。

- **在弹出的对话框中，选择自动续费周期，单击开通自动续费。**

修改自动续费周期

1. 登录*POLARDB*管理控制台。
2. 在控制台右上方，选择费用 > 续费管理。

3. 在控制台左上角单击体验新版，切换到新版控制台。

4. 通过搜索过滤功能在自动续费页签中找到目标集群，单击右侧修改自动续费。

说明：

示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到云数据库POLARDB，然后修改自动续费。

5. 在弹出的对话框中，修改自动续费周期后，单击确定。

关闭自动续费

1. 登录*POLARDB*管理控制台。
2. 在控制台右上方，选择费用 > 续费管理。

3. 在控制台左上角单击体验新版，切换到新版控制台。

4. 通过搜索过滤功能在自动续费页签中找到目标集群，单击右侧恢复手动续费。

说明：

示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到云数据库POLARDB，然后关闭自动续费。

5. 在弹出的对话框中，单击确定。

相关API

API	描述
#unique_32	**创建数据库集群** **说明：** **创建集群时开通自动续费。**
#unique_33	**设置包年包月集群自动续费状态** **说明：** **创建集群后开通自动续费。**
#unique_34	**查询包年包月集群自动续费状态**

8 连接数据库集群

8.1 查看连接地址

POLARDB for PostgreSQL数据库连接地址包括集群访问地址和主访问地址。

查看方法

1. **登录*POLARDB*控制台。**
2. **在控制台左上角，选择集群所在地域。**
3. **单击目标集群的ID。**
4. **在基本信息页面的访问信息区域，查看连接地址。**

集群地址和主地址

地址类型	地址说明	支持的网络类型
集群地址	POLARDB包含一个默认的集群地址。应用程序只需连接一个集群地址，即可连接到多个节点。带有读写分离功能，写请求会自动发往主节点，读请求会自动根据各节点的负载发往主节点或只读节点。	公网和私网
主地址	主地址总是连接到主节点，支持读和写操作。当主节点发生故障时，只读节点会接替它成为主节点，主地址也会自动切换到新的主节点。	公网和私网
主节点地址（不推荐）	主节点的地址，由于故障时节点会不可用，不建议直连主节点。 **说明:** 主地址与主节点地址的区别： · 主地址总是连接到主节点，因此当主节点故障切换时也会自动切换到新的主节点。 · 主节点地址则不支持故障切换，当主节点故障时会不可用，因此不建议直连主节点地址。	私网
只读节点地址（不推荐）	只读节点的地址，由于故障时节点会不可用，不建议直连只读节点。	私网

私网地址和公网地址

地址类型	说明	使用场景
私网地址	· 通过私网地址访问可以发挥POLARDB的最佳性能。 · 私网地址无法被释放。	例如： · ECS与数据库集群位于同一VPC，那么ECS可以通过私网地址访问数据库集群。 · 使用DMS通过VPC访问数据库集群。
公网地址	· 需手动申请公网的连接地址，也可以释放公网的连接地址。 · 公网即因特网。通过公网访问将无法实现POLARDB最佳性能。	例如：通过公网访问数据库集群进行维护操作。

下一步

连接数据库集群

相关API

API	描述
#unique_36	查询集群的地址信息。
#unique_37	创建集群的公网地址。
#unique_38	修改集群默认访问地址。
#unique_39	释放集群地址。

8.2 连接数据库集群

本文介绍如何通过DMS和客户端连接到POLARDB for PostgreSQL集群。

前提条件

已创建数据库集群的高权限账号或普通账号。具体操作请参见创建数据库账号。

使用DMS登录POLARDB

数据管理（Data Management Service，简称DMS）是一种集数据管理、结构管理、访问安全、BI图表、数据趋势、数据轨迹、性能与优化和服务器管理于一体的数据管理服务。支持对关系型数据库（MySQL、SQL Server、PostgreSQL等）和NoSQL数据库（MongoDB、Redis等）的管理，同时还支持Linux服务器管理。

1. 登录*POLARDB*控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID，进入基本信息页面。
4. 单击页面右上角的登录数据库。

5. 在数据库登录页面，输入主地址和端口号（以英文冒号隔开），以及高权限账号或普通账号的用户名和密码，然后单击登录。

说明:

DMS登录仅支持主地址，不支持集群地址。关于如何查看连接地址，请参见*#unique_14*。

通过客户端连接

由于POLARDB for PostgreSQL暂不支持设置集群白名单，只有相同VPC内的实例才可以访问集群，所以客户端所在主机和POLARDB for PostgreSQL集群需要在同一VPC内。

1. 启动pgAdmin 4客户端。

2. 右击Servers，选择创建 > 服务器，如下图所示。

3. 在创建-服务器页面的通常标签页面中，输入服务器名称，如下图所示。

参数说明如下：

- 主机名称/地址：输入POLARDB集群的内网地址。查看POLARDB集群的地址及端口信息的步骤如下：

  a. 进入*POLARDB*控制台。

  b. 找到目标集群，单击集群的ID。

  c. 在访问信息区域查看地址及端口信息。

- **端口：需输入POLARDB集群的内网端口。**
- **用户名：POLARDB集群的高权限账号名称。**
- **密码：POLARDB集群的高权限账号所对应的密码。**

4. 选择Connection标签页，输入要连接的实例信息，如下图所示。

5. 单击保存。

6. 若连接信息无误，选择Servers > 服务器名称 > 数据库 > postgres，会出现如下界面，则表示连接成功。

说明:

postgres是POLARDB for PostgreSQL集群默认的系统数据库，请勿在这个数据库中进行任何操作。

9 集群

9.1 创建POLARDB PostgreSQL数据库集群

本文介绍如何通过POLARDB管理控制台创建POLARDB PostgreSQL数据库集群。

前提条件

已注册阿里云账号或已创建子账号。

- **如要注册阿里云账号，请点此注册。**
- **如要创建子账号，请参见*#unique_41*进行子账号的创建和授权。**

背景信息

一个集群包含一个主节点以及最多15个只读节点，只读节点最少一个，用于提供Active-Active高可用。节点是虚拟化的数据库服务器，节点中可以创建和管理多个数据库。

说明:

- **POLARDB仅支持专有网络VPC（Virtual Private Cloud）。VPC是阿里云上一种隔离的网络环境，安全性比传统的经典网络更高。**
- **POLARDB与其他阿里云产品通过内网互通时才能发挥POLARDB的最佳性能，因此，建议将POLARDB与云服务器ECS配合使用，且与ECS创建于同一个VPC，否则POLARDB无法发挥最佳性能。如果您ECS的网络类型为经典网络，需将ECS从经典网络迁移到VPC，具体请参见*ECS实例迁移*。**

优惠活动

首购折扣价：首次购买POLARDB享受折扣价。详情请参见优惠活动。

操作步骤

1. **登录阿里云。**
   - **如果使用阿里云账号，请点此登录。**
   - **如果使用子账号，请点此登录。更多信息请参见*#unique_41*。**
2. **进入*POLARDB*购买页面。**

3. 选择包年包月或按量付费。

- 包年包月：在创建集群时支付计算节点（一个主节点和一个只读节点）的费用，而存储空间会根据实际数据量按小时计费，并从账户中按小时扣除。如果您要长期使用该集群，包年包月方式更加划算，而且购买时长越长，折扣越多。
- 按量付费（按小时付费）：无需预先支付费用，计算节点和存储空间（根据实际数据量）均按小时计费，并从账户中按小时扣除。如果您只需短期使用该集群，可以选择按量付费，用完即可释放，节省费用。

4. 设置如下参数。

控制台区域	参数	说明
基本配置	地域	集群所在的地理位置。购买后无法更换地域。 说明: 请确保POLARDB与需要连接的ECS创建于同一个地域，否则它们无法通过内网互通，只能通过外网互通，无法发挥最佳性能。
	可用区	• 可用区是地域中的一个独立物理区域，不同可用区之间没有实质性区别。 • 您可以选择将POLARDB与ECS创建在同一可用区或不同的可用区。
	网络类型	• 无需选择。 • 仅支持专有网络VPC（Virtual Private Cloud）。VPC是一种隔离的网络环境，安全性和性能均高于传统的经典网络。

控制台区域	参数	说明
	VPC网络 VPC交换机	请确保POLARDB与需要连接的ECS创建于同一个VPC，否则它们无法通过内网互通，无法发挥最佳性能。 · 如果您已创建符合您网络规划的VPC，直接选择该VPC。例如，如果您已创建ECS，且该ECS所在的VPC符合您的规划，那么选择该VPC。 · 如果您未创建符合您网络规划的VPC，您可以使用默认VPC和交换机： - 默认VPC： ■ 在您选择的地域中是唯一的。 ■ 网段掩码是16位，如172.31.0.0/16，最多可提供65536个私网IP地址。 ■ 不占用阿里云为您分配的VPC配额。 - 默认交换机： ■ 在您选择的可用区中是唯一的。 ■ 网段掩码是20位，如172.16.0.0/20，最多可提供4096个私网IP地址。 ■ 不占用VPC中可创建交换机的配额。 · 如果以上默认VPC和交换机无法满足您的要求，您可以自行创建VPC和交换机。
配置	数据库引擎	· 兼容MySQL 8.0（完全兼容）。原生支持并行查询，特定场景下（TPC-H测试）性能提升十倍，详情请参见#unique_43。 · 兼容MySQL 5.6（完全兼容）。 · 兼容PostgreSQL 11（完全兼容）。 · 兼容Oracle（高度兼容）。
	节点规格	按需选择。所有POLARDB节点均为独享型，性能稳定可靠。关于各规格的具体信息，请参见#unique_45。
	节点个数	· 无需选择。系统将自动创建一个与主节点规格相同的只读节点。 · 如果主节点故障，系统会自动将只读节点切换为新的主节点，并重新生成一个只读节点。 · 关于只读节点的更多信息，请参见#unique_46。
	存储费用	无需选择。系统会根据实际数据使用量按小时计费，详情请参见产品价格。 说明： 创建集群时无需选择存储容量，存储容量随数据量的增减而自动弹性伸缩。

5. 设置购买时长（仅针对包年包月集群）和集群数量，然后单击右侧的立即购买。

> 说明:
> 最多可以一次性创建50个集群，适用于游戏批量开服等业务场景。

6. 在确认订单页面，确认订单信息，阅读和勾选服务协议，单击去支付。

   完成支付后，集群将在十分钟左右创建成功，在集群列表中可以看到创建的集群。

> 说明:
> - 当集群中的节点状态为运行中时，整个集群可能仍未创建完成，此时集群不可用。只有当集群状态为运行中时，集群才可以正常使用。
> - 请确认已选中正确的地域，否则无法看到您创建的集群。
> - 当您的数据量较大时，推荐您购买POLARDB存储包，相比按小时付费，预付费购买存储包有折扣，购买的容量越大，折扣力度就越大。

下一步

创建数据库账号

相关API

9.2 临时升配

POLARDB的包年包月集群支持临时升配，可以帮助您轻松应对短时间的业务高峰期。

功能说明

临时升配是指临时升级规格，提升整体性能。到达指定的还原时间后，集群的规格会自动还原到临时升配前的状态。

> 说明:
> 不支持临时降级，如需降级请参见#unique_49。

前提条件

- 集群为包年包月集群。
- 集群没有尚未生效的续费变配订单。
- 集群没有尚未生效的临时升配订单。

影响

还原过程可能会出现闪断，请确保应用程序具备重连机制。

注意事项

- 还原时间不能晚于集群到期时间的前1天。例如集群1月10日到期，则临时升配的还原时间最多为1月9日。
- 临时升配的最短时间为1天，由于设置还原时间后无法修改，建议升配时间为14天以内。
- 临时升配期间不支持普通的#unique_49。
- 临时升配后如果性能不够，在还原时间到达之前最多可以再进行1次升配，此次设置的还原时间不能早于第1次。

计费

临时升配的价格是新老配置差价的1.5倍。计算公式如下：

临时升配N天，费用 =（新规格包月价格 - 老规格包月价格）/30 x 1.5 x N。

临时升配

1. 进入POLARDB控制台。
2. 选择地域。
3. 进入升降配向导页面。您可以按照如下两种方式操作：
   - 找到目标集群，在操作列单击升降配。

   - 找到目标集群，单击集群ID，在基本信息页面下方单击升降配。

4. 勾选临时升级配置，单击确定。

> **说明：**

仅包年包月集群支持临时升级配置。

5. **选择节点规格。**
6. **选择还原时间。**

说明:

- **临时升配后如果性能不够，在还原时间到达之前最多可以再进行1次升配，此次设置的还原时间不能早于第1次。**
- **临时升配的最短时间为1天，由于设置还原时间后无法修改，建议升配时间为14天以内。**

- 还原时间不能晚于集群到期时间的前一天。

7. 勾选服务协议，单击去支付完成支付。

9.3 使用存储包

为了更好地帮助您降低存储成本，POLARDB推出了预付费形式的存储包。

POLARDB的存储空间无需手动配置，根据数据量自动伸缩，您只需为实际使用的数据量按量付费。当您的数据量较大时，推荐您使用POLARDB存储包，相比按量付费，预付费购买存储包有折扣，购买的容量越大，折扣力度就越大。

存储包和按量付费的价格对比

下表为按月计费存储包和按量付费的价格对比。 存储容量越大，使用存储包越经济。

容量（GB）	中国大陆		中国香港及海外	
	按量付费（元/月）	存储包（元/月）	按量付费（元/月）	存储包（元/月）
100	350	350	390	390
500	1,750	1,750	1,950	1,950
1,000	3,500	3,150	3,900	3,510
3,000	10,500	9,450	11,700	10,530
5,000	17,500	15,750	19,500	17,550
10,000	35,000	31,500	39,000	35,100
30,000	105,000	84,000	117,000	93,600
50,000	175,000	140,000	195,000	156,000

容量（GB）	中国大陆		中国香港及海外	
	按量付费（元/月）	存储包（元/月）	按量付费（元/月）	存储包（元/月）
100,000	350,000	280,000	390,000	312,000

注意事项

- 每种类型资源包只允许购买一个，存储包容量不够时，可以升级存储包。
- 存储包暂不支持降级。
- 存储包由资源包类型规定的地域内的所有集群共享使用。
- 超出存储包容量的部分以按量付费的方式收取费用。例如购买了1000G的存储包，有三个POLARDB集群，存储容量分别是300G、400G和500G，那么300+400+500-1000=200G，存储包抵扣后，超出的200G需要以按量付费的方式收取费用，您可以在费用中心查看存储包抵扣量和按量付费的存储量。

说明:

更多关于存储包的说明，请参见常见问题。

购买存储包

1. 登录购买*POLARDB*存储包页面。
2. 设置如下参数。

参数	说明
资源包类型	• 中国大陆通用：购买后可用于中国大陆地域的POLARDB集群。 • 中国香港及海外通用：购买后可用于中国香港及海外地域的POLARDB集群。
存储包规格	存储包的容量大小。

参数	说明
购买时长	购买存储包的时长。

3. 单击立即购买。
4. 勾选服务协议，单击去支付完成支付。

查看存储包抵扣量和按量付费的存储量

1. 登录费用中心。

> **说明:**
> 如果跳转到新版费用中心，请在左侧导航栏单击返回旧版。

2. 在左侧导航栏选择消费记录 > 消费明细。

3. 设置搜索条件查询POLARDB相关账单，单击查询。

4. 单击目标账单右侧详情。

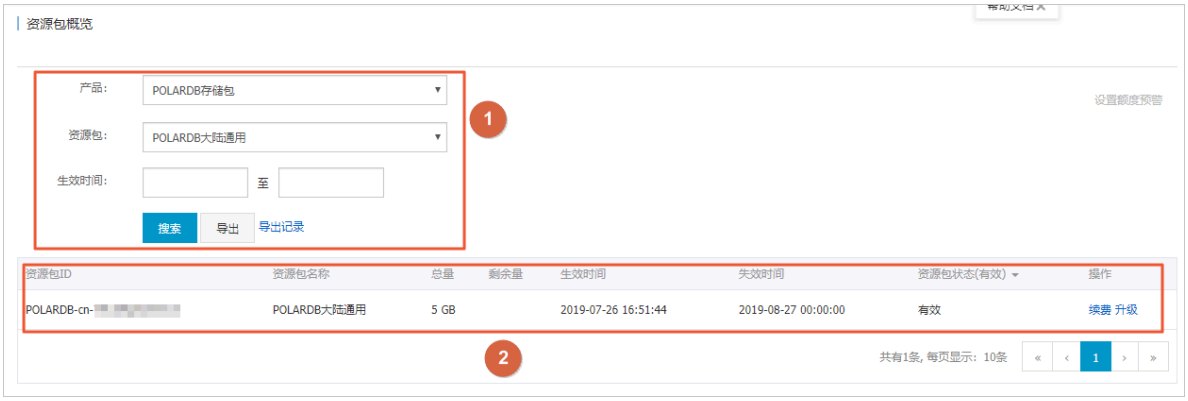
5. 展开费用详单查看抵扣详情。

续费/升级存储包

1. 登录**费用中心**。

> **说明：**
> 如果跳转到新版费用中心，请在左侧导航栏单击返回旧版。

2. 在左侧导航栏选择资源包管理 > 资源概览。
3. 在产品栏选择POLARDB存储包，设置搜索条件，单击搜索。



4. 您可以按如下方法进行续费或升级操作：

- 续费

  a. 单击操作列的续费。

  b. 选择续费时长，勾选服务协议，单击去支付完成支付。

- 升级

  a. 单击操作列的升级。

  b. 选择存储包规格，勾选服务协议，单击去支付完成支付。

常见问题

- **存储包是否跟集群绑定售卖?**

  答：不绑定。您需要单独购买存储包，购买后会自动抵扣相应地域内的集群存储空间。
- **存储包是否可以被多个集群共享?**

  答：可以。存储包由资源包类型（中国大陆或中国香港及海外）规定的地域内的所有集群共享使用。
- **存储包是否可以被不同引擎的集群共享?**

  答：可以。存储包可以同时用于POLARDB for MySQL/PostgreSQL/Oracle的集群。
- **存储包容量不够了怎么办？可以再买一个吗?**

  答：每种类型资源包只允许购买一个。存储包容量不够时可以进行升级。
- **当前数据量超出存储包容量的部分如何计费?**

  答：超出存储包容量的部分以按量付费的方式收取费用，详情请参见*#unique_45/unique_45_Connect_42_section_u9d_n9d_3jt*。
- **存储包支持3TB和5TB，业务只需要4TB，如何选购?**

  答：您可以先购买3TB，存储量接近5TB时再升级为5TB。

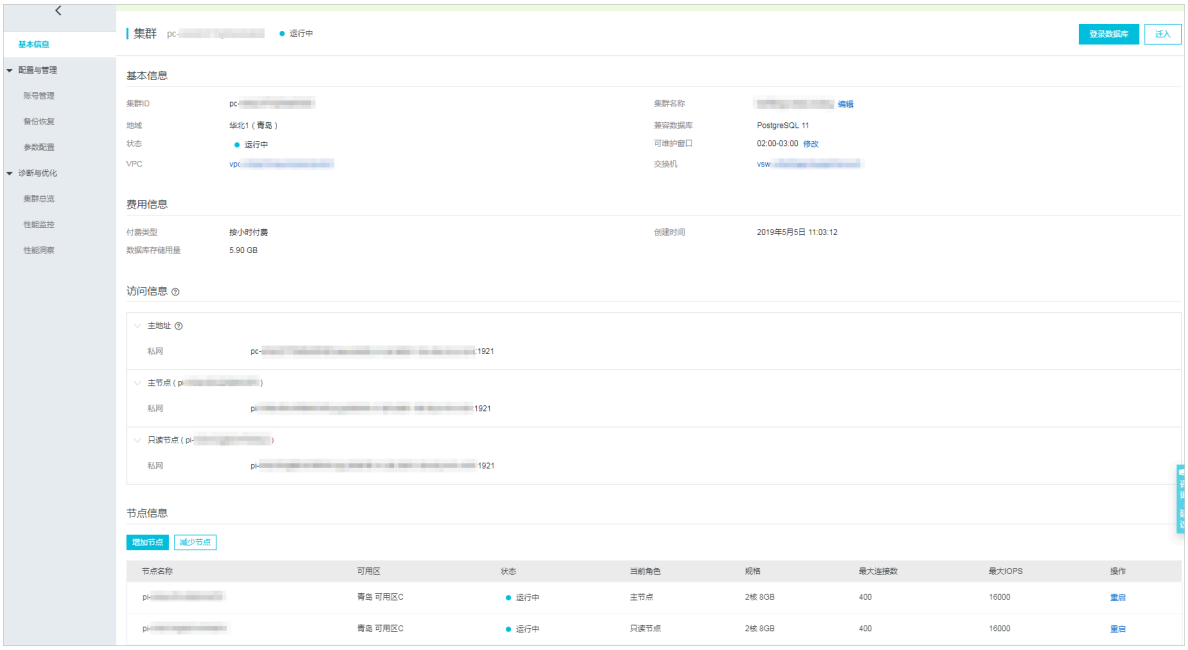
9.4 查看数据库集群

查看集群列表和集群详细信息

1. **进入*POLARDB*控制台。**
2. **在左上角选择地域，即可显示您账号下在该地域的所有集群。**

3. 单击一个集群的ID，即可进入该集群的详细信息页面。

集群的详细信息包括基本信息、费用信息、访问信息和该集群中包含的节点信息。



相关API

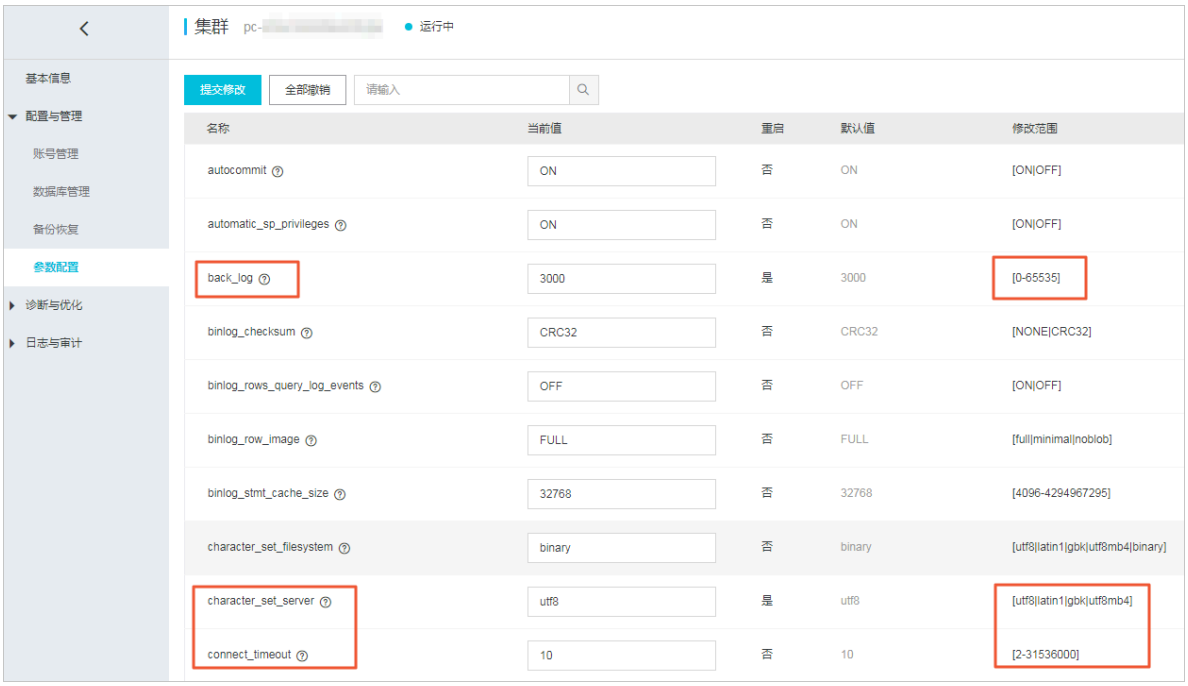
API	描述
#unique_32	创建数据库集群
#unique_51	查看集群列表
#unique_52	查看数据库集群的属性

9.5 设置集群参数

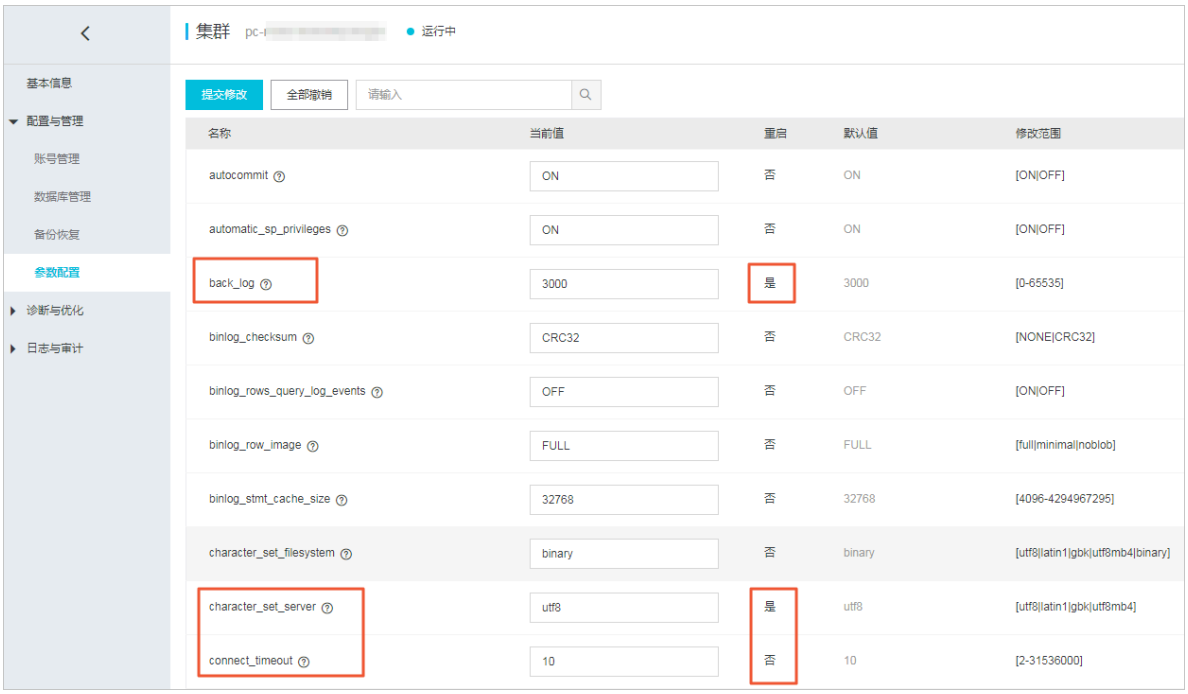
本文将介绍如何通过控制台修改集群参数。

注意事项

- **请按照控制台上规定的修改范围修改参数值。**



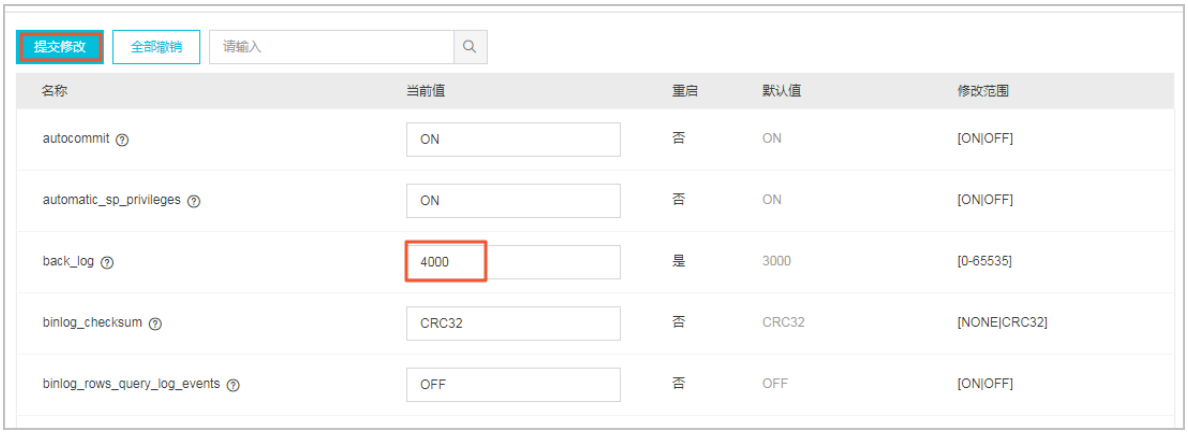
- **部分参数修改后需要重启全部节点，重启前请做好业务安排，谨慎操作。详情请参见参数设置页面中的重启列。**



操作步骤

1. **进入*POLARDB*控制台。**
2. **选择地域。**
3. **找到目标集群，单击集群名称列的集群ID。**

4. 在左侧导航栏中，选择配置与管理 > 参数配置。
5. 修改一个或多个参数的当前值，单击提交修改。



6. 在弹出的保存改动对话框中，单击确定。

相关API

API	描述
#unique_54	查看集群的参数
#unique_55	修改集群的参数

9.6 变更配置

您可以根据业务需求变更集群的配置。

前提条件

集群没有正在进行的配置变更时，才可以变更集群规格。

背景信息

POLARDB支持三维扩展能力：

- 计算能力纵向扩展：集群规格升降配。本文介绍详细信息。

- 计算能力横向扩展：增加或减少只读节点。具体操作说明，请参见增加或删除节点。
- 存储空间横向扩展：POLARDB采用Serverless架构，无需手动设置容量或扩缩容，容量随用户数据量的变化而自动在线调整。

本文介绍如何升级或降级集群的规格，新规格会立即开始生效（每个节点需要5到10分钟）。

变更配置的费用说明

详情请参见#unique_58。

注意事项

- 您只能对整个集群进行规格升降级，无法对集群中的单个节点进行规格升降级。
- 集群规格的升降级不会对集群中已有数据造成任何影响。
- 在集群规格变更期间， POLARDB服务会出现几秒钟的闪断且部分操作不能执行的状况，建议您在业务低谷期执行变更。闪断后需在应用端重新连接。

操作步骤

1. 登录POLARDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 进入升降配向导页面。您可以按照如下两种方式操作：
   - 找到目标集群，在操作列单击升降配。

   - 找到目标集群，单击集群ID，在基本信息页面下方单击升降配。

4. 勾选升级配置或降级配置，单击确定。

说明:

仅包年包月集群支持临时升级配置，详情请参见*#unique_59*。

5. 选择所需的规格。

说明:

同一集群中，所有节点的规格总是保持一致。

6. 勾选服务协议，单击去支付并完成支付。

说明:

规格变更预计需要10分钟生效。

9.7 增加或删除节点

创建POLARDB集群后，您可以手动增加或删除只读节点。集群最多包含15个只读节点，最少一个只读节点（用于保障集群的高可用）。同一集群中，所有节点的规格总是保持一致。

节点费用

增加节点时的计费方式如下：

- **如果集群为包年包月（预付费），则增加的节点也是包年包月。**
- **如果集群为按小时付费（后付费），则增加的节点也是按小时付费。**

说明：

- **包年包月和按小时付费的只读节点都可以随时释放，释放后会退款或停止计费。**
- **增加节点仅收取节点规格的费用（详情请参见*#unique_45*），存储费用仍然按实际使用量收费，与节点数量无关。**

注意事项

- **仅当集群没有正在进行的配置变更时，才可以增加或删除只读节点。**
- **为避免操作失误，每次操作只能增加或删除一个只读节点，增加或删除多个只读节点请多次操作。**
- **增加或删除节点需要5分钟左右生效。**

增加只读节点

1. **进入*POLARDB*控制台。**
2. **选择地域。**

3. 进入增删节点向导页面。您可以按照如下两种方式操作：

   - 找到目标集群，在操作列单击增删节点。

   - 找到目标集群，单击集群ID，在基本信息页面下方单击增删节点。

4. 勾选增加节点并单击确定。

5. 单击 + 增加一个只读节点，勾选服务协议，单击去支付并完成支付。

删除只读节点

1. 进入*POLARDB*控制台。
2. 选择地域。
3. 进入增删节点向导页面。您可以按照如下两种方式操作：
   - 找到目标集群，在操作列单击增删节点。

   - 找到目标集群，单击集群ID，在基本信息页面下方单击增删节点。

4. 勾选删除节点并单击确定。

5. 单击想要删除的节点后面的 — ，并在弹出对话框中单击确定。

> **说明：**
> 集群中必须保留至少一个只读节点，以保障集群的高可用。

6. 勾选服务协议，单击确认。

相关API

API	描述
#unique_60	增加POLARDB集群节点
#unique_61	变更POLARDB集群节点规格
#unique_62	重启POLARDB集群节点
#unique_63	删除POLARDB集群节点

9.8 设置可维护窗口

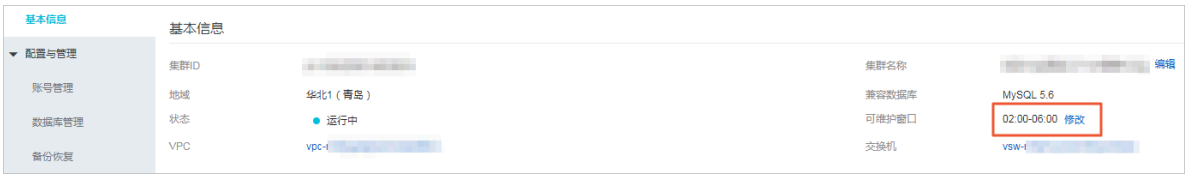
在阿里云平台上，为保障云数据库POLARDB的稳定性，后端系统会不定期对集群进行维护操作，确保集群平稳运行。您可以根据业务规律，将可维护窗口设置在业务低峰期，以免维护过程中对业务造成影响。

注意事项

- 在进行正式维护前，POLARDB 会给阿里云账号中设置的联系人发送短信和邮件，请注意查收。
- 集群维护当天，为保障整个维护过程的稳定性，集群会在所设置的可维护窗口之前的一段时间，进入集群维护中的状态，当集群处于该状态时，数据库本身正常的数据访问不会受到任何影响，但该集群的控制台上，除了账号管理、数据库管理和添加 IP 白名单外，其他涉及变更类的功能均无法使用（如常用的升降级、重启等操作均无法重启），查询类如性能监控等可以正常查阅。
- 在进入集群所设置的可维护窗口后，集群会在该段时间内发生1到2次的连接闪断，请确保您的应用程序具有重连机制。闪断后，集群即可恢复到正常状态。

操作步骤

1. 登录*POLARDB控制台*。
2. 选择地域。
3. 找到目标集群，单击集群名称列的集群ID。
4. 在基本信息中的可维护窗口后单击修改。



5. 在编辑可维护窗口中选择集群的可维护窗口，单击提交。

相关API

API	描述
CreateDBCluster	创建数据库集群
ModifyDBClusterMaintainTime	修改集群可运维时间

9.9 重启节点

当节点出现连接数满或性能问题时，您可以手动重启节点。重启节点会造成连接中断，重启前请做好业务安排，谨慎操作。

操作步骤

1. 进入*POLARDB控制台*。
2. 选择地域。
3. 找到目标集群，单击集群名称列的集群ID。

4. 在基本信息页面下方节点信息里找到需要重启的节点。
5. 单击操作列中的重启。

6. 在弹出的对话框中，单击确认。

相关API

API	描述
#unique_62	重启数据库节点

9.10 释放集群

根据业务需求，您可以手动释放后付费（按小时付费）的集群。

注意事项

- 预付费（包年包月）集群不支持手动释放，集群到期后会自动被释放。
- 只有在运行状态下的集群才能被手动释放。
- 集群被释放后，数据将无法找回，请谨慎操作。
- 本功能用于释放整个集群，包括集群中的所有节点。如要释放单个只读节点，请参考**增加或删除节点**。
- 按小时付费的集群可以直接转为包年包月，具体请参见**按小时付费转包年包月**。

操作步骤

1. 登录*POLARDB***控制台**。
2. 选择地域。
3. 找到目标集群，单击操作列的··· > 释放。

4. 在弹出的提示框中，单击确认。

相关API

API	描述
#unique_51	查看集群列表
#unique_68	删除数据库集群

9.11 克隆集群

您可以根据已有的POLARDB集群的数据（包括账号信息，不包括集群参数配置信息），克隆出相同的POLARDB集群。

被克隆的是执行克隆动作时的数据。克隆开始后，新写入的数据不会被克隆。

操作步骤

1. **进入*POLARDB*控制台。**
2. **选择地域。**
3. **找到要克隆的集群，单击操作列的… > 恢复到新集群。**
4. **在配置页面设置以下参数：**

参数	说明
克隆源类型	这里选择本集群。
地域	指集群所在的地理位置。克隆集群的地域和原集群相同，不支持修改。
可用区	· 可用区是地域中的一个独立物理区域，不同可用区之间没有实质性区别。 · 您可以选择将POLARDB与ECS创建在同一可用区或不同的可用区。
网络类型	· 无需选择。 · 仅支持专有网络VPC（Virtual Private Cloud）。VPC是一种隔离的网络环境，安全性和性能均高于传统的经典网络。
VPC网络 VPC交换机	从下拉菜单中选择VPC和交换机，或者*创建新的VPC和交换机*。 **说明:** 请确保POLARDB与需要连接的ECS创建于同一个VPC，否则它们无法通过内网互通，无法发挥最佳性能。
数据库引擎	无需选择。
节点规格	按需选择。不同规格有不同的最大存储容量和性能，具体请参见*#unique_45*。
节点个数	无需选择。系统将自动创建一个与主节点规格相同的只读节点。

参数	说明
集群名称	· 可选。 · 如果留空，系统将为您自动生成一个集群名称。创建集群后还可以修改集群名称。
购买时长	预付费集群需要填写此参数。
集群数量	默认为1，无法修改。

5. 阅读并勾选《云数据库 POLARDB服务协议》，然后完成支付。

10 账号

10.1 账号概述

控制台账号

您可以使用以下账号登录控制台：

- **阿里云账号：该账号是阿里云资源的归属和计费主体。购买阿里云产品之前，您需要先注册阿里云账号。**
- **子账号（可选）：如果其他人需要使用您账号下的资源，您可以使用RAM控制台创建和管理子账号。子账号本身不拥有资源，且以主账号作为计费主体。**

数据库集群账号

您可以使用以下账号登录数据库集群。更多信息请参见*#unique_72*。

账号类型	说明
高权限账号	· 只能通过控制台创建和管理。 · 一个集群只能有一个高权限账号，可以管理所有普通账号和数据库。 · 开放了更多权限，可满足个性化和精细化的权限管理需求，比如可按用户分配不同表的查询权限。 · 拥有集群中所有数据库的所有权限。 · 可以断开任意账号的连接。

相关API

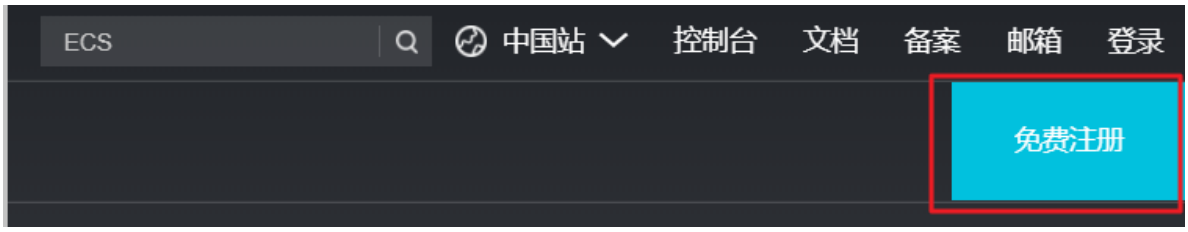
API	描述
#unique_73	创建账号
#unique_74	查看账号列表
#unique_75	修改账号备注
#unique_76	修改账号密码
#unique_77	账号授权
#unique_78	撤销账号权限
#unique_79	重置账号权限
#unique_80	删除账号

10.2 注册和登录阿里云账号

注册阿里云账号

您可以通过两种方式注册阿里云账号：

- **进入阿里云官网（*https://www.aliyun.com*），单击右上角的免费注册。**



- **直接访问注册页面（*https://account.aliyun.com/register/register.htm*）。**

登录阿里云账号

阿里云账号的登录入口与子账号不同。

- **阿里云账号的登录入口：*https://account.aliyun.com/login/login.htm*。**

- **子账号的登录入口：** *https://signin.aliyun.com/login.htm*

10.3 创建和使用子账号

如果只有您本人使用POLARDB，那么使用阿里云账号即可。如果需要让其他人使用您账号下的资源，请创建子账号。

创建RAM子账号

1. **您可以使用阿里云账号或有RAM权限的子账号来创建子账号，首先需要登录RAM控制台。具体操作如下：**
   - **如果使用阿里云账号，请点此登录。**
   - **如果使用子账号，请点此登录。**

   > **说明:**
   > **子账号登录的格式为子账号名@公司别名。**
2. **在左侧导航栏的人员管理菜单下，单击用户。**

3. 单击新建用户。

> **说明:**
> 单击添加用户，可一次性创建多个RAM用户。

4. 输入登录名称和显示名称。
5. 在访问方式区域下，选择控制台密码登录。
6. 控制台密码选择自动生成默认密码或自定义登录密码。
7. 要求重置密码选择用户在下次登录时必须重置密码或无需重置。
8. 多因素认证选择不要求。
9. 单击确认。

在授权页面下为RAM用户授权

1. 在左侧导航栏的权限管理菜单下，单击授权。
2. 单击新增授权。
3. 在被授权主体区域下，输入RAM用户名称后，单击需要授权的RAM用户。
4. 在左侧权限策略名称列表下，单击需要授予RAM用户的权限策略。

> **说明:**
> 在右侧区域框，选择某条策略并单击×，可撤销该策略。

5. 单击确定。
6. 单击完成。

在用户页面下为RAM用户授权

1. 在左侧导航栏的人员管理菜单下，单击用户。
2. 在用户登录名称/显示名称列表下，找到目标RAM用户。
3. 单击添加权限，被授权主体会自动填入。
4. 在左侧权限策略名称列表下，单击需要授予RAM用户的权限策略。

> **说明:**
> 在右侧区域框，选择某条策略并单击×，可撤销该策略。

5. 单击确定。
6. 单击完成。

登录子账号

前提条件：您已完成上述账号授权步骤。

您可以通过两种地址登录子账号：

- **通用登录地址：***https://signin.aliyun.com/login.htm*

  如果通过此地址登录，您需手动输入子账号名以及公司别名。格式为子账号名@公司别名。

- **专用登录地址：如果您可以登录*RAM*控制台，可以在RAM控制台查看到您公司的子账号登录地址。**

如果通过此地址登录，系统将自动为您填写公司别名，您只需输入子账号名。

更多操作

您还可以对子账号进行更多的操作，如把子账号添加到用户组、为子账号分配角色、为用户组或角色授权等。详情请参见*RAM*用户指南。

10.4 创建数据库账号

本文为您介绍如何创建数据库账号，以及高权限账号与普通账号的区别。

背景信息

POLARDB支持两种数据库账号：高权限账号和普通账号。您可以在控制台管理所有账号。

账号类型

账号类型	说明
高权限账号	· 只能通过控制台或API创建和管理。 · 一个集群可以创建多个高权限账号，高权限账号可以管理所有普通账号和数据库。 · 开放了更多权限，可满足个性化和精细化的权限管理需求，例如可按用户分配不同表的查询权限。 · 拥有集群中所有数据库的所有权限。 · 可以断开任意账号的连接。
普通账号	· 可以通过控制台或者SQL语句创建和管理。 · 一个集群可以创建多个普通账号，具体的数量与数据库内核有关 。 · 需要手动给普通账号授予特定数据库的权限。 · 普通账号不能创建和管理其他账号，也不能断开其他账号的连接。

创建账号

1. 登录*POLARDB*控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击配置与管理 > 账号管理。
5. 单击创建账号。
6. 设置以下参数：

参数	说明
账号名	填写账号名称。要求如下： · 以小写字母开头，以字母或数字结尾。 · 由小写字母、数字或下划线组成。 · 长度为2~16个字符。 · 不能使用某些预留的用户名，如root、admin。
账号类型	· 选择高权限账号，创建高权限账号。 · 选择普通账号，创建普通账号。
密码	设置账号的密码。要求如下： · 由大写字母、小写字母、数字或特殊字符组成，至少包含其中三类。 · 长度为8~32个字符。 · 特殊字符为!@#$%^&*()_+-=。

参数	说明
确认密码	再次输入密码。
备注	备注该账号的相关信息，便于后续账号管理。要求如下： · 不能以http://或https://开头。 · 必须以大小写字母或中文开头。 · 可以包含大小写字母、中文、数字、下划线"_"或连字符"-"。 · 长度为2~256个字符。

7. 单击确定。

下一步

#unique_14

相关API

API	描述
#unique_73	创建账号
#unique_74	查看账号列表
#unique_75	修改账号备注
#unique_76	修改账号密码
#unique_77	账号授权
#unique_78	撤销账号权限
#unique_79	重置账号权限

10.5 管理数据库账号

本文为您介绍如何管理数据库账号，包括修改密码、锁定账号、解锁账号和删除账号等。

背景信息

POLARDB支持两种数据库账号：高权限账号和普通账号。您可以在控制台管理所有账号。

创建数据库账号

具体操作请参见**创建数据库账号**。

修改密码

1. 登录*POLARDB***控制台**。
2. 在页面左上角，选择地域。

3. 单击目标集群ID。
4. 在左侧导航栏中，单击配置与管理 > 账号管理。
5. 在目标账号操作栏中，单击修改密码。

6. 在弹出的对话框中，输入新密码和确认新密码，单击确定。

锁定账号

您可以通过锁定账号功能，锁定目标账号，禁止使用该账号登录数据库。

1. 登录*POLARDB*控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击配置与管理 > 账号管理。
5. 在目标账号操作栏中，单击锁定。
6. 在弹出的对话框中，单击确定。

解锁账号

1. 登录*POLARDB*控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击配置与管理 > 账号管理。
5. 在目标账号操作栏中，单击解锁。
6. 在弹出的对话框中，单击确定。

删除账号

1. 登录*POLARDB*控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击配置与管理 > 账号管理。
5. 在目标账号操作栏中，单击删除账号。

6. 在弹出的对话框中，单击确定。

相关API

API	描述
#unique_73	创建账号
#unique_74	查看账号列表
#unique_75	修改账号备注
#unique_76	修改账号密码
#unique_77	账号授权
#unique_78	撤销账号权限
#unique_79	重置账号权限
#unique_80	删除账号

11 数据库

本文为您介绍如何创建数据库以及删除数据库。

背景信息

您可以在控制台创建和管理所有的数据库。

创建数据库

1. **登录*POLARDB*控制台。**
2. **在控制台左上角，选择地域。**
3. **单击目标集群ID。**
4. **在左侧导航栏中，选择配置与管理 > 数据库管理。**
5. **单击创建数据库。**

6. 设置以下参数。

参数	说明
数据库（DB）名称	· 以字母开头，以字母或数字结尾； · 由小写字母、数字、下划线或中划线组成； · 长度为2~64个字符。 · 数据库名称在实例内必须是唯一的。
数据库Owner	数据库的所有者，对数据库拥有ALL权限。
支持字符集	数据库支持的字符集，默认为UTF8。 如果需要其他字符集，请在下拉列表中选择需要的字符集。
Collate	字符串排序规则。
Ctype	字符分类。
备注说明	用于备注该数据库的相关信息，便于后续数据库管理。要求如下： · 不能以http://或https://开头。 · 必须以大小写字母或中文开头。 · 可以包含大小写字母、中文、数字、下划线"_"或连字符"-"。 · 长度为2~256个字符。

7. 单击确定。

删除数据库

1. 登录*POLARDB*控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择配置与管理 > 数据库管理。
5. 单击目标数据库操作栏中的删除。
6. 在弹出的对话框中，单击确认。

相关API

API	描述
#unique_85	创建数据库
#unique_86	查看数据库列表信息
#unique_87	修改数据库描述
#unique_88	删除数据库

12 备份与恢复

12.1 备份数据

POLARDB采用物理备份（快照备份），每天自动备份一次，您也可以手动发起备份。自动备份和手动备份都不会影响集群的运行。备份文件保留的时间为7天。

备份类型

备份类型	说明
自动备份	· 默认为每天一次，您可以设置自动执行备份的时间段和周期。具体请参见设置自动备份。 · 备份文件不可删除。
手动备份	· 可随时发起。每个集群最多可以有3个手动创建的备份。具体操作请参见手动创建备份。 · 备份文件可删除。

费用

POLARDB备份文件占用的存储空间暂不收费。

设置自动备份

1. **进入*POLARDB*控制台。**
2. **选择地域。**
3. **找到目标集群，单击集群名称列的集群ID。**
4. **在左侧导航栏中，选择配置与管理 > 备份恢复。**
5. **单击备份设置。**

6. **在弹出的对话框中，设置自动执行备份的时间段和周期。**

> **说明:**
> **出于安全考虑，自动备份的频率为每周至少两次。**

手动创建备份

1. 进入*POLARDB*控制台。
2. 选择地域。
3. 找到目标集群，单击集群名称列的集群ID。
4. 在左侧导航栏中，选择配置与管理 > 备份恢复。
5. 单击创建备份。

6. 在弹出的对话框中，单击确认。

> **说明:**
> 每个集群最多可以有3个手动创建的备份。

恢复数据

请参见恢复数据。

相关API

API	描述
#unique_94	创建POLARDB集群全量快照备份。
#unique_95	查询POLARDB集群备份信息。
#unique_96	删除POLARDB集群备份。
#unique_97	查询POLARDB集群自动备份策略。
#unique_98	修改POLARDB集群自动备份策略。

12.2 恢复数据

POLARDB for PostgreSQL支持按备份集（快照）恢复，将历史数据恢复到新集群中。

> **说明:**
> 恢复后的集群包含原集群的数据和账号信息，不包含原集群的参数设置。

按备份集（快照）恢复

1. 进入*POLARDB*控制台。

2. 选择集群所在的地域。
3. 找到目标集群，单击集群ID。
4. 在左侧导航栏中，选择配置与管理 > 备份恢复。
5. 找到目标备份集（快照），单击恢复备份，在弹出的对话框中单击确认。
6. 在弹出的页面中，选择新集群的计费方式：
   - 预付费：在创建集群时需要支付计算实例（一个主实例和一个只读实例）的费用，而存储空间会根据实际数据量按小时计费，并从账户中按小时扣除。如果您要长期使用该集群，预付费方式更加划算，而且购买时长越长，折扣越多。
   - 按量付费：无需预先支付费用，计算实例和存储空间（实际数据量）均按小时计费，并从账户中按小时扣除。如果您只需短期使用该集群，可以选择按量付费，用完即可释放，节省费用。
7. 设置以下参数：
   - 克隆源类型：选择备份集。
   - 克隆源备份集：请确认是否为您要恢复的备份集。
   - 地域：无需修改，与原集群相同。
   - 可用区：无需修改。
   - 网络类型：无需修改。
   - VPC网络和VPC交换机：建议保持不变，即原集群所在的VPC网络和交换机。
   - 数据库引擎：无需修改。
   - 节点规格：不同规格有不同的最大存储容量和性能，具体请参见节点规格。
   - 节点个数：无需修改。系统将自动创建一个与主节点规格相同的只读节点。
   - 集群名称：如果留空，系统将为您自动生成一个集群名称。创建集群后还可以修改集群名称。
   - 购买时长：预付费集群需要填写此参数。
   - 集群数量：默认为1，无法修改。
8. 阅读并勾选《云数据库 POLARDB服务协议》，然后完成支付。

相关主题

备份数据

相关API

API	描述
#unique_32	**创建POLARDB集群。** **说明:** **克隆集群时，参数CreationOption取值需要为CloneFromPolarDB。**

13 诊断与优化

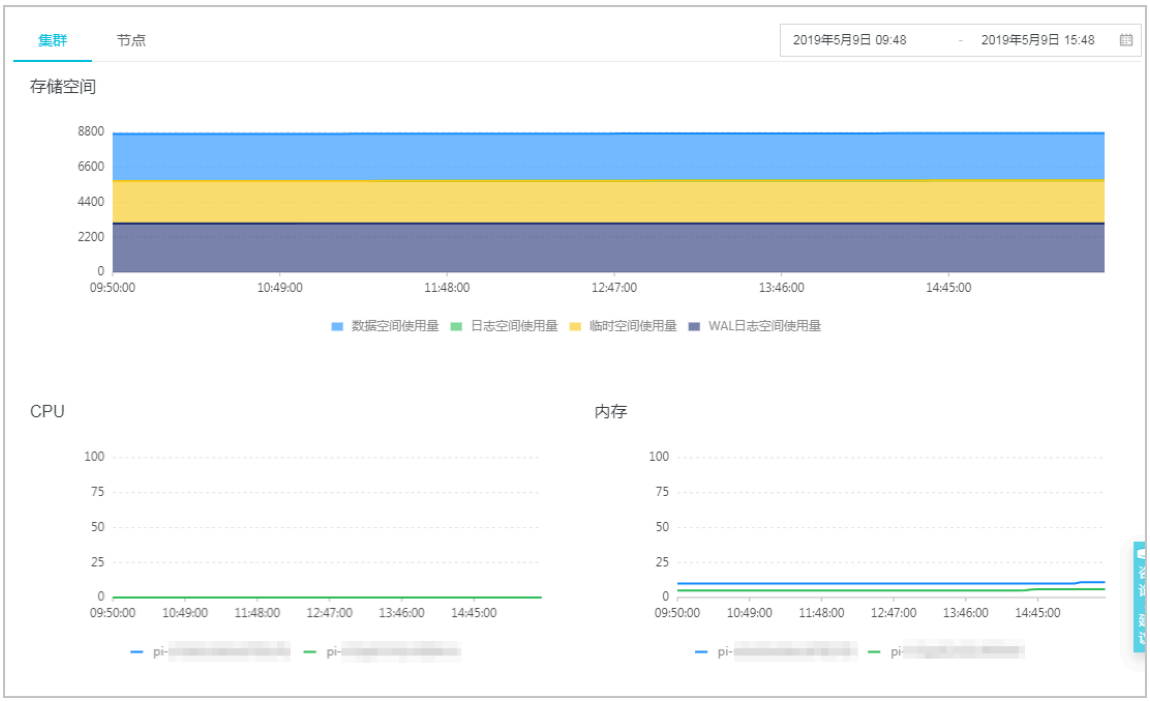
13.1 性能监控与报警

为方便您掌握实例的运行状态，POLARDB控制台提供了丰富的性能监控项。

性能监控

1. **进入*POLARDB*控制台。**
2. **选择地域。**
3. **找到目标集群，单击集群名称列的集群ID。**
4. **在左侧导航栏中选择诊断与优化 > 性能监控。**
5. **根据您的需求，可以查看集群或节点的监控信息。详细说明请参见监控项说明。**

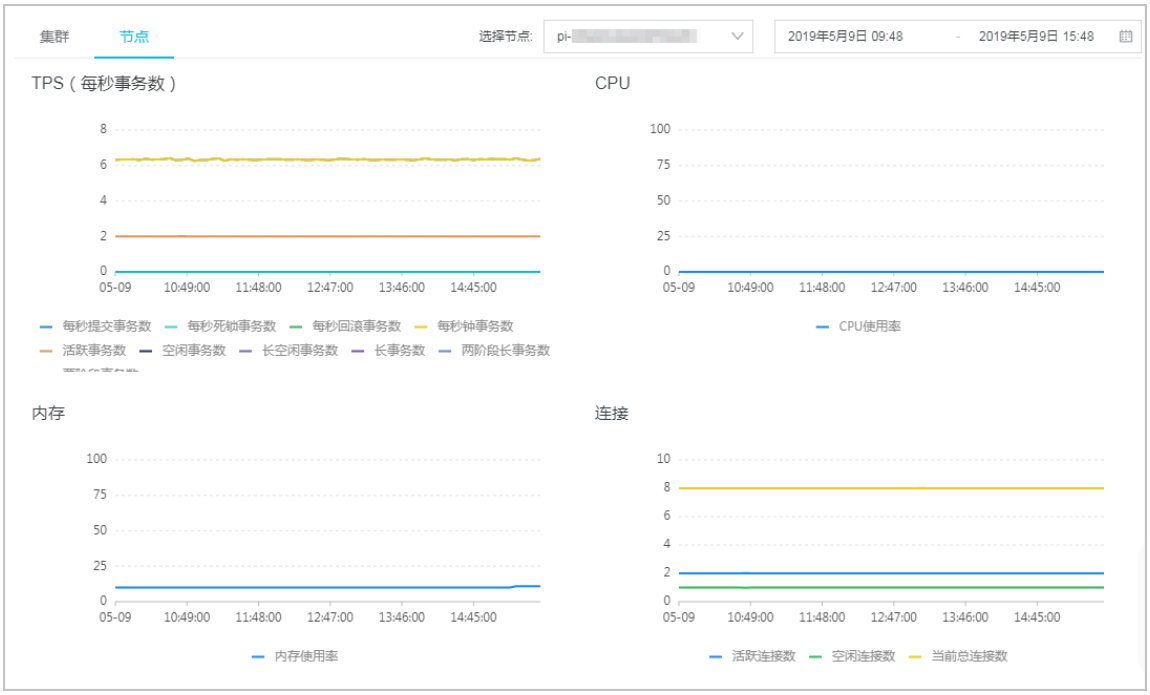
   - **集群性能监控：单击集群，在右侧设置时间段后单击确定。**



   - **节点性能监控：单击节点，在右侧选择节点并设置时间段后单击确定。**

说明:

节点页面下方单击显示更多，会显示更多监控项。



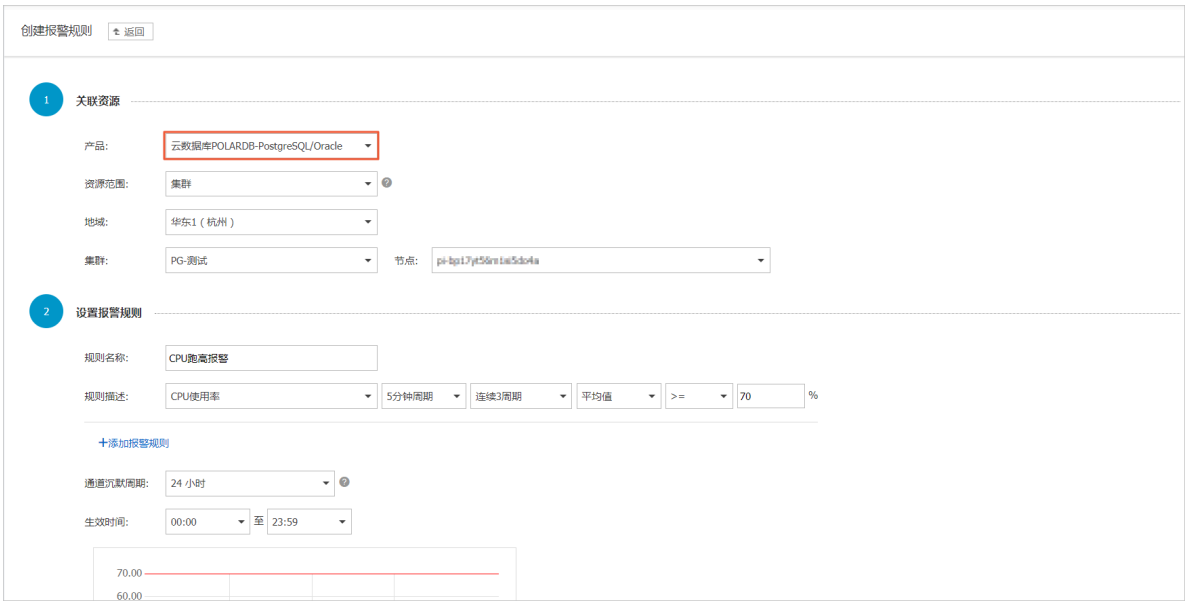
监控项说明

类别	监控项	说明
集群	存储空间	展示数据空间、日志空间、临时空间和WAL日志空间的使用量。
	CPU	展示各节点的CPU使用率。
	内存	展示各节点的内存使用率。
节点	TPS	展示所选择节点的每秒事务数，包括每秒提交事务数、每秒死锁事务数、每秒回滚事务数等等。
	CPU	展示所选择节点的CPU使用率。
	内存	展示所选择节点的内存使用率。
	连接	展示所选择节点的当前总连接数、活跃连接数和空闲连接数。
	扫描行数	展示所选择节点每秒插入、读取、更新、删除、返回的行数。
	数据库最大年龄	数据库最旧和最新的两个事务之间的事务ID差值。
	I/O吞吐量	展示所选择节点的总I/O吞吐量、读I/O吞吐量、写I/O吞吐量。
	IOPS	展示所选择节点的每秒读写次数，包括每秒读写总次数、每秒读次数、每秒写次数。
	缓存	展示所选择节点每秒缓存读取次数和每秒磁盘读取次数。
	缓存命中率	展示所选择节点的缓存命中率。

类别	监控项	说明
	临时文件	展示所选择节点的临时文件数量和总大小。

设置报警

1. 进入云监控控制台。
2. 在左侧导航栏中，选择报警服务 > 报警规则。
3. 在报警规则列表页面，单击创建报警规则，进入创建报警规则页面。



4. 在产品下拉列表中，选择云数据库POLARDB-PostgreSQL/Oracle，选择资源范围，设置报警规则和通知方式后，单击确认即可。

> **说明:**
> 报警规则相关说明，请参见*#unique_103*。

13.2 性能洞察

性能洞察（Performance Insights）专注于POLARDB集群负载监控、关联分析、性能调优的利器，以简单直观的方式帮助用户迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。

典型使用场景

性能洞察可以在以下场景中，为您提供帮助。

- 概要分析集群性能指标。

  帮助您监控集群的关键性能指标，从宏观角度帮助您确认数据库集群负载情况和变化趋势。根据集群关键性能指标趋势图，可以帮助您发现集群负载来源以及负载分布的时间规律。

- 轻松评估数据库负载。

  您无需综合分析复杂繁多的性能指标趋势图，平均活跃会话趋势图中展示了所有核心性能信息，这些信息帮助您轻松地评估数据库负载来源和瓶颈类型，例如是高CPU使用率，还是锁定等待，又或者是I/O延迟等，并且可以直接定位具体是哪些SQL语句。

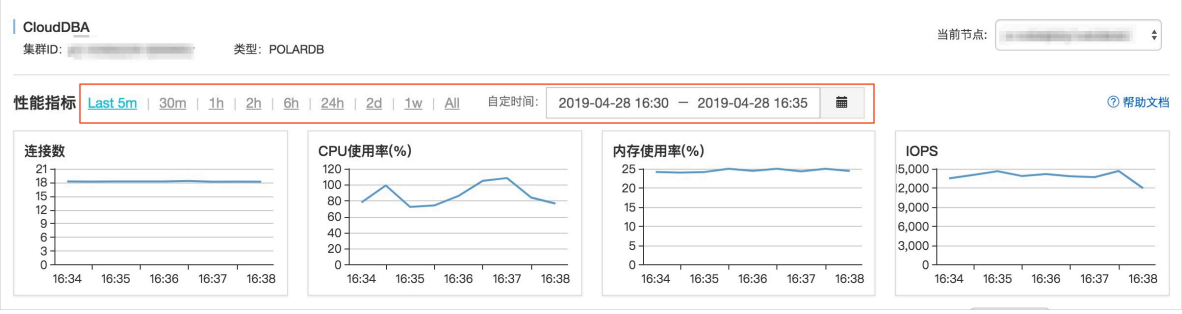
  > **说明:**
  > 平均活跃会话（Average Active Sessions，AAS），是指用户POLARDB集群一段时间内的平均活跃会话数，AAS的数量变化趋势反映了用户POLARDB集群负载的变化情况。因此，性能洞察功能使用AAS来做为POLARDB集群负载高低的衡量指标。

- 简单查找性能问题源头。

  结合AAS趋势图和多维度负载详情进行分析，您可以迅速确定性能问题是集群规格配置导致的，或者是数据库本身设计导致的，并找到是哪些SQL语句导致了性能问题。

操作步骤

1. 登录*POLARDB*控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择诊断与优化 > 性能洞察。
5. 选择过滤条件。

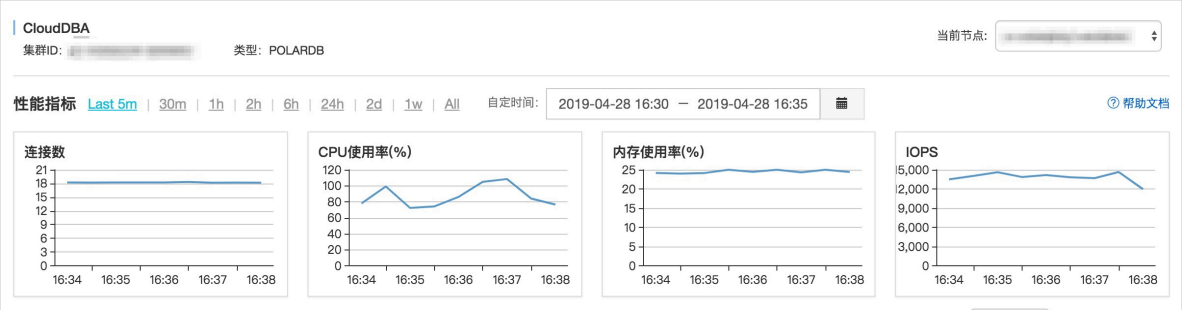


页面介绍

- **关键性能指标趋势图**

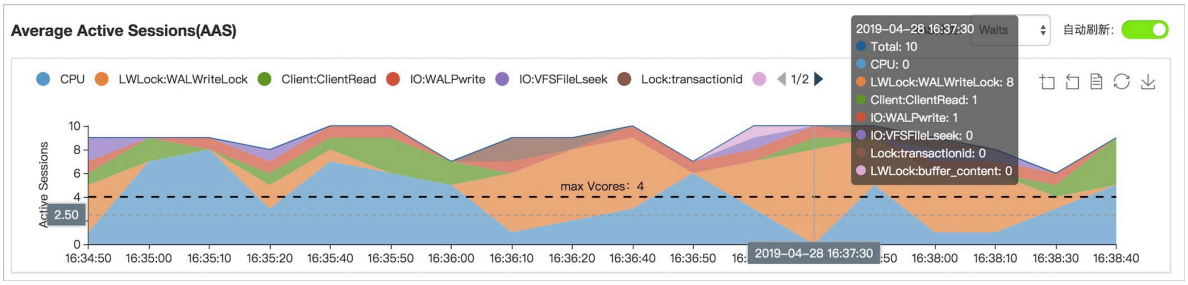
  用户可以通过关键性能指标的趋势图确认集群负载的情况和资源瓶颈。

  您还可以切换时间段或者选择自定义时间范围，来获取相应时间段的关键性能指标趋势图。



- **平均活跃会话（AAS）**

  通过关键性能指标的趋势图，宏观确认数据库的负载情况后，可以进一步确认负载来源。



> **说明:**
> **max Vcores是指用户POLARDB集群最多可以使用的CPU核数，这个值的大小决定了集群CPU的处理能力。**

从实时AAS变化趋势图中，可以清楚地发现POLARDB集群中的负载来源和时间，以及变化规律。

- **多维度负载源详情**

  通过分析性能洞察中的实时AAS变化趋势，掌握了集群负载变化的规律，接下来可以从多个维度找出影响性能的具体SQL语句，以及相关联的用户、主机、数据库等。

从以上截图的下半部分，我们可以方便地找出与AAS变化趋势关联负载对应的SQL查询语句，以及每个语句对AAS的使用占比情况。

性能洞察支持6个维度的AAS分类，您可以通过右侧的AAS分类下拉框来切换。

类别	说明
SQL	业务TOP 10 SQL的AAS变化趋势。
Waits	活跃会话资源等待的AAS变化趋势。
Users	登录用户的AAS变化趋势。
Hosts	客户端主机名或者主机IP AAS变化趋势。
Databases	业务所在数据库的AAS变化趋势。
Status	活跃会话状态的AAS变化趋势。

14 SQL洞察

SQL洞察功能为您的数据库提供安全审计、性能诊断等增值服务。

费用说明

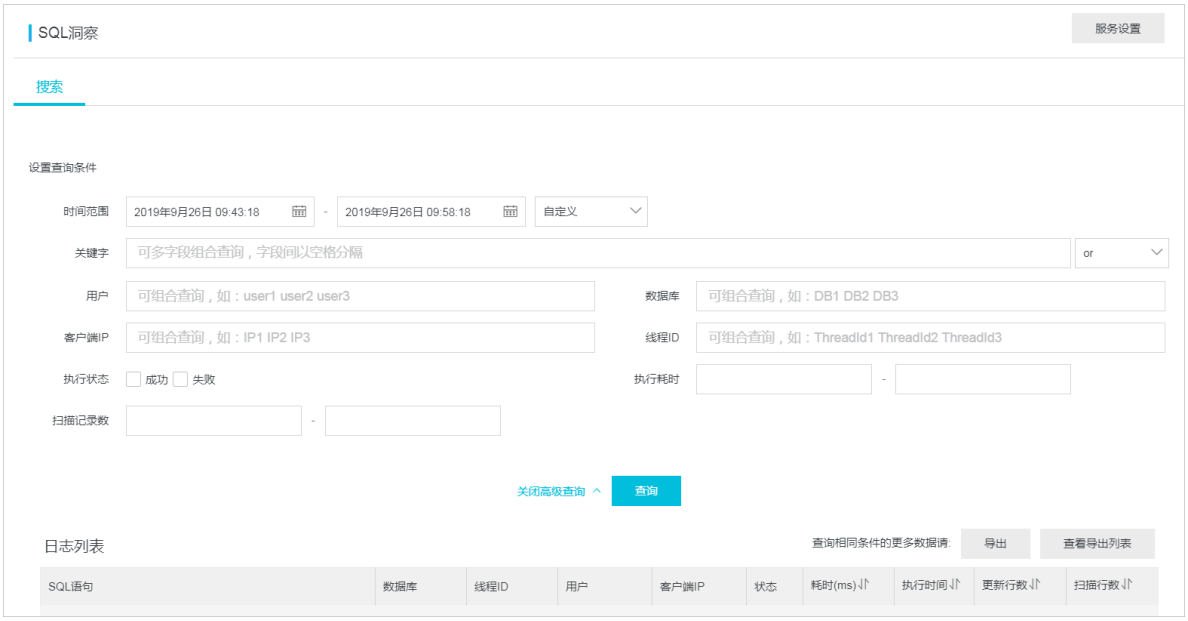
- **试用版：免费使用，审计日志仅保存一天，即只能查询一天范围内的数据，不支持数据导出等高级功能，不保障数据完整性。**
- **30天或以上： 详情请参见*#unique_45*。**

功能说明

- **SQL审计日志**

  记录对数据库执行的所有操作。通过审计日志记录，您可以对数据库进行故障分析、行为分析、安全审计等操作。
- **增强搜索**

  可以按照数据库、用户、客户端IP、线程ID、执行耗时、执行状态等进行多维度检索，并支持导出和下载搜索结果。



开通SQL洞察

1. **登录*POLARDB*控制台。**
2. **在控制台左上角，选择地域。**
3. **单击目标集群ID。**
4. **在左侧导航栏中，选择日志与审计 > SQL洞察。**

5. 单击立即开通。

6. 选择SQL审计日志的保存时长，单击开通服务。

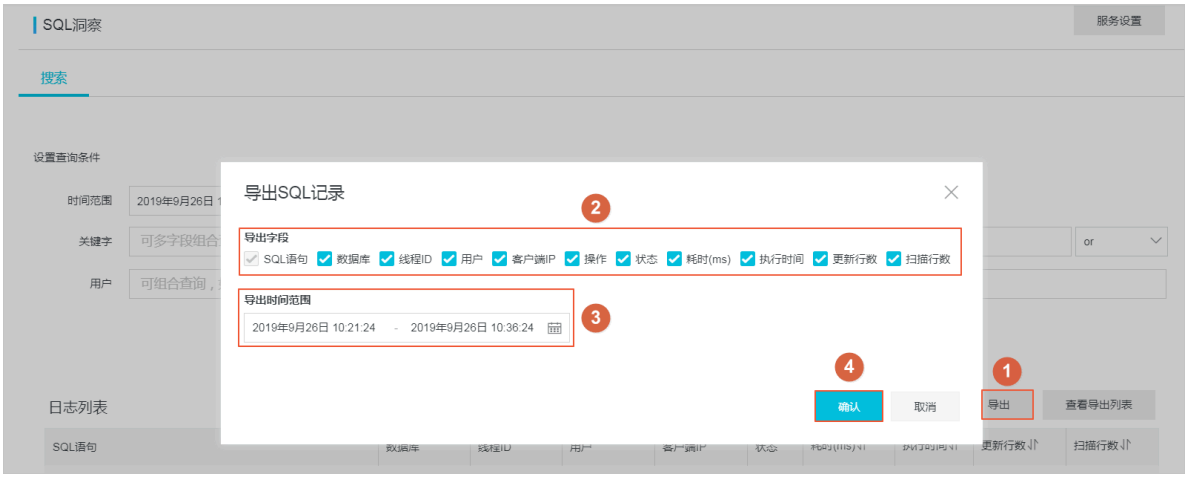
修改SQL日志的存储时长

1. 登录*POLARDB*控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择日志与审计 > SQL洞察。
5. 单击右上角服务设置。
6. 修改存储时长，单击确认。

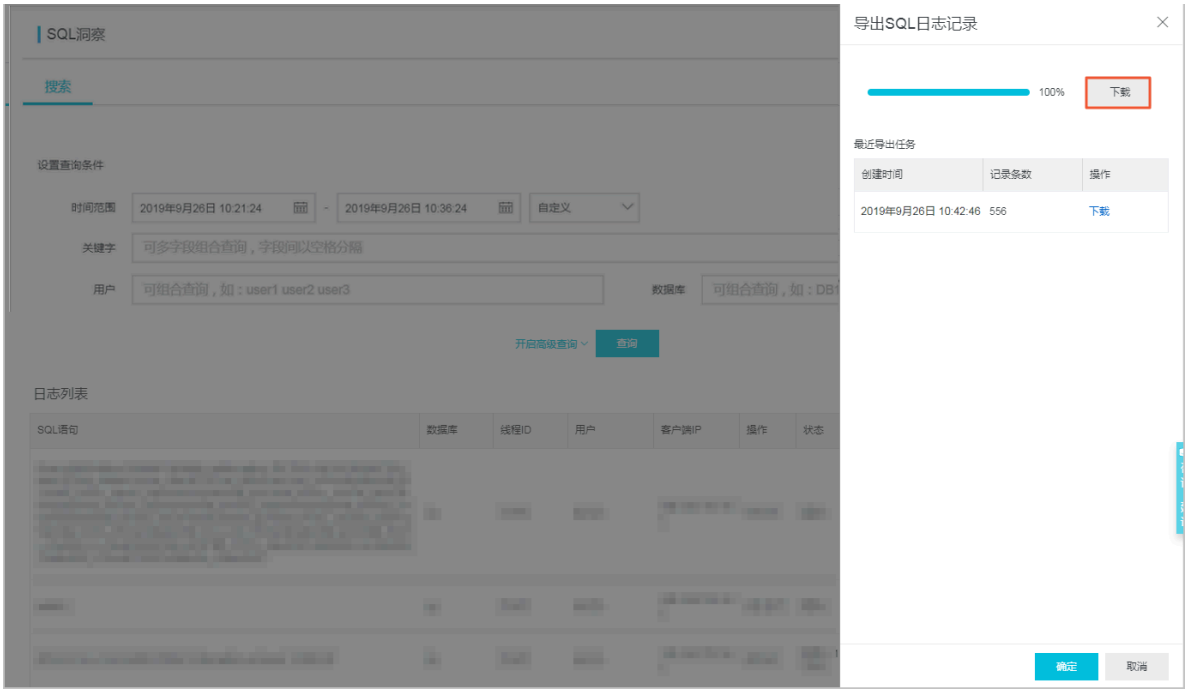
导出SQL记录

1. 登录*POLARDB*控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择日志与审计 > SQL洞察。
5. 单击右侧导出。

6. 在弹出的对话框中，选择导出字段和导出时间范围，单击确认。



7. 导出完成后，在导出SQL日志记录中，下载已导出的文件并妥善保存。



关闭SQL洞察

说明:

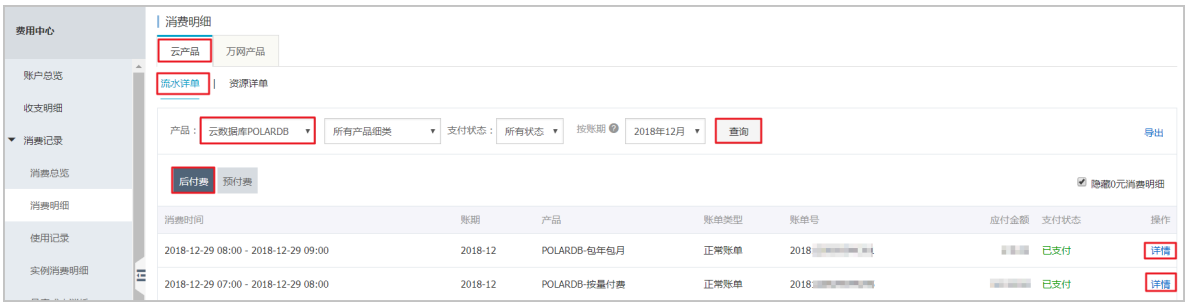
SQL洞察功能关闭后，SQL审计日志会被清空。请将*SQL*审计日志导出后，再关闭SQL洞察功能。

1. 登录*POLARDB*控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择日志与审计 > SQL洞察。
5. 单击右上角服务设置。

6. 修改存储时长，单击确认。
7. 单击滑块关闭SQL洞察。

查看审计日志的大小和消费明细

1. 登录阿里云管理控制台。
2. 在页面右上角，选择费用 > 进入费用中心。
3. 在左侧导航栏中，选择消费记录 > 消费明细。
4. 选择云产品页签。

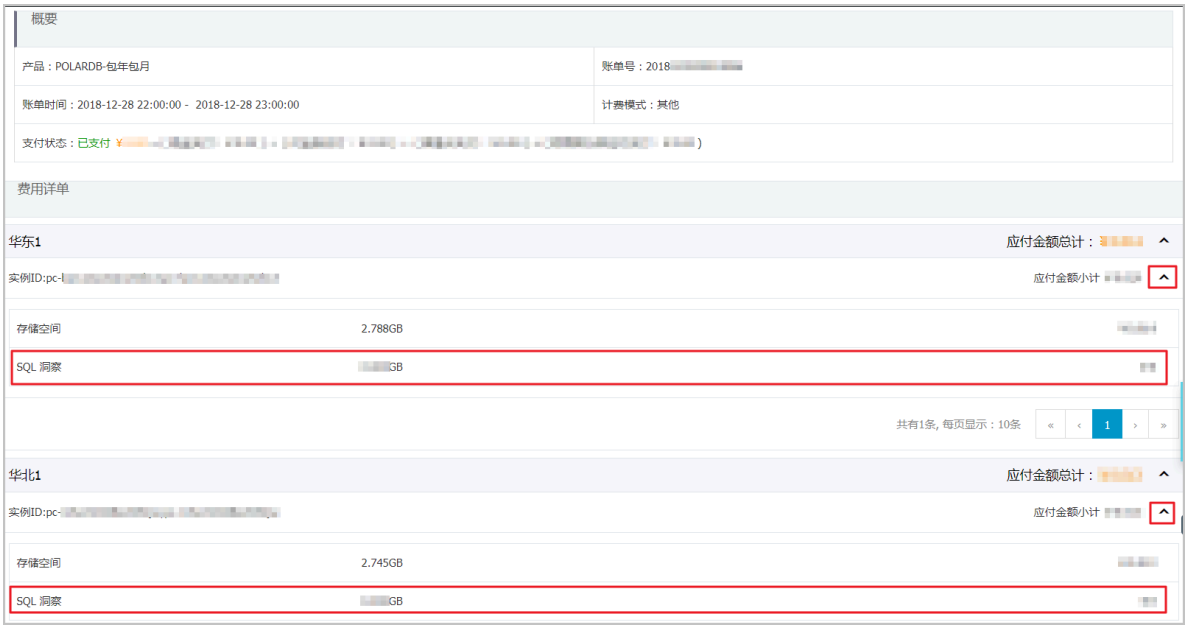


5. 选择流水详单。
6. 选择后付费。
7. 设置查询条件，然后单击查询。

> **说明:**
> 若要查询超过12个月前的记录，请提交工单。

8. 根据POLARDB集群的计费方式，找到POLARDB-按量付费或POLARDB-包年包月，单击最右侧的详情。

9. 单击最右侧的箭头符号，即可查看SQL洞察的审计日志大小以及费用。



15 使用oss_fdw读写外部数据文本文件

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

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/csv、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

- 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： OSSBucket，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。

- 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。

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

- 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。
- 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用例

```
# 创建插件
create extension 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;
# 可以看到
# 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)
```

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POLARDB for PostgreSQL集群的资源（CPU IO MEM MET）相关，也和OSS相关。**
- **为保证数据导入的性能，请确保云数据库POLARDB for PostgreSQL与OSS所在Region相同，相关信息请参考*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*。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。**

请参考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。

最终的加密后的信息如下：

```
postgres=# select * from pg_foreign_server ;
  srvname  | srvowner | srvfdw | srvtype | srvversion | srvacl |

  srvoptions
-----------+----------+--------+---------+------------+--------
+-----------------------------------------------------------------------------------
----------------------------------
 ossserver |       10 |  16390 |         |            |        | {host
=oss-cn-hangzhou-zmf.aliyuncs.com, id=MD5xxxxxxxx, 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。

16 使用TimescaleDB插件

POLARDB for PostgreSQL数据库集群新增TimescaleDB插件1.3.0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POLARDB for PostgreSQL 11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*。

添加TimescaleDB插件

使用pgAdmin客户端*#unique_109*，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;
```

```
69  SELECT time_bucket('15 minutes', time) AS fifteen_min,
70      location, COUNT(*),
71      MAX(temperature) AS max_temp,
72      MAX(humidity) AS max_hum
73    FROM conditions
74    WHERE time > NOW() - interval '3 hours'
75    GROUP BY fifteen_min, location
76    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

您可也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;
```

```
SELECT percentile_cont(0.5)
  WITHIN GROUP (ORDER BY temperature)
  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;
```

```
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;
```

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

17 使用pg_pathman插件

本文介绍pg_pathman插件的一些常见用法。

背景信息

为了提高分区表的性能，POLARDB for PostgreSQL引入了pg_pathman插件。该插件一款分区管理插件，提供了分区优化机制。

创建pg_pathman插件扩展

```
test=# create extension pg_pathman;
CREATE EXTENSION
```

查看已安装的扩展

以下命令可以查看已安装的扩展，还可以查看到pg_pathman 的具体版本。

```
test=# \dx
                    List of installed extensions
    Name    | Version |   Schema   |            Description
------------+---------+------------+-----------------------------------
 pg_pathman | 1.5     | public     | Partitioning tool for PostgreSQL
 plpgsql    | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)
```

插件升级

POLARDB for PostgreSQL会定期对插件进行升级，以提供更优质的数据库服务。而当您需要升级插件版本时，需要：

- 升级对应集群到最新版本。
- 执行SQL版本更新命令，如下所示：

```
ALTER EXTENSION pg_pathman UPDATE;
SET pg_pathman.enable = t;
```

插件特性

- 目前支持HASH分区、RANGE分区。
- 支持自动分区管理（通过函数接口创建分区，自动将主表数据迁移到分区表），或手工分区管理（通过函数实现，将已有的表绑定到分区表，或者从分区表剥离）。
- 支持的分区字段类型包括int、float、date以及其他常用类型，包括自定义的domain。
- 有效的分区表查询计划（JOINs、subselects 等）。
- 使用`RuntimeAppend` & `RuntimeMergeAppend` 自定义计划节点实现了动态分区选择。
- `PartitionFilter`：一种有效的插入触发器替换方法。

- 支持自动新增分区（目前仅支持RANGE分区表）。
- 支持`copy from/to`直接读取或写入分区表，提高效率。
- 支持分区字段的更新，需要添加触发器，如果不需要更新分区字段，则不建议添加这个触发器，会产生一定的性能影响。
- 允许用户自定义回调函数，在创建分区时会自动触发。
- 非堵塞式创建分区表，以及后台自动将主表数据非堵塞式迁移到分区表。
- 支持FDW，通过配置参数`pg_pathman.insert_into_fdw=(disabled | postgres | any_fdw)`支持postgres_fdw或任意FDW。

插件用法

- 相关视图和表
- 分区管理
- 高级分区管理

更多用法，请参见*https://github.com/postgrespro/pg_pathman*。

相关视图和表

pg_pathman使用函数来维护分区表，并且创建了一些视图，可以查看分区表的状态，具体如下：

1. pathman_config

```
CREATE TABLE IF NOT EXISTS pathman_config (
    partrel         REGCLASS NOT NULL PRIMARY KEY,  -- 主表oid
    attname         TEXT NOT NULL,  -- 分区列名
    parttype        INTEGER NOT NULL,  -- 分区类型(hash or range)
    range_interval  TEXT,  -- range分区的interval

    CHECK (parttype IN (1, 2)) /* check for allowed part types */ );
```

2. pathman_config_params

```
CREATE TABLE IF NOT EXISTS pathman_config_params (
    partrel        REGCLASS NOT NULL PRIMARY KEY,  -- 主表oid
    enable_parent  BOOLEAN NOT NULL DEFAULT TRUE,  -- 是否在优化器中过滤主表
    auto           BOOLEAN NOT NULL DEFAULT TRUE,  -- insert时是否自动扩展不存在的分区
    init_callback  REGPROCEDURE NOT NULL DEFAULT 0);  -- create partition时的回调函数oid
```

3. pathman_concurrent_part_tasks

```
-- helper SRF function
CREATE OR REPLACE FUNCTION show_concurrent_part_tasks()
RETURNS TABLE (
    userid     REGROLE,
    pid        INT,
```

```
    dbid       OID,
    relid      REGCLASS,
    processed  INT,
    status     TEXT)
AS 'pg_pathman', 'show_concurrent_part_tasks_internal'
LANGUAGE C STRICT;

CREATE OR REPLACE VIEW pathman_concurrent_part_tasks
AS SELECT * FROM show_concurrent_part_tasks();
```

4. **pathman_partition_list**

```
-- helper SRF function
CREATE OR REPLACE FUNCTION show_partition_list()
RETURNS TABLE (
    parent     REGCLASS,
    partition  REGCLASS,
    parttype   INT4,
    partattr   TEXT,
    range_min  TEXT,
    range_max  TEXT)
AS 'pg_pathman', 'show_partition_list_internal'
LANGUAGE C STRICT;

CREATE OR REPLACE VIEW pathman_partition_list
AS SELECT * FROM show_partition_list();
```

分区管理

1. **RANGE分区**

有四个管理函数用来创建范围分区。其中两个可以指定起始值、间隔、分区个数，其函数定义如下：

```
create_range_partitions(relation       REGCLASS,  -- 主表OID
                        attribute      TEXT,      -- 分区列名
                        start_value    ANYELEMENT,  -- 开始值
                        p_interval     ANYELEMENT,  -- 间隔；任意类
型，适合任意类型的分区表
                        p_count        INTEGER DEFAULT NULL,   --
  分多少个区
                        partition_data BOOLEAN DEFAULT TRUE)   --
  是否立即将数据从主表迁移到分区，不建议这么使用，建议使用非堵塞式的迁移( 调用
partition_table_concurrently() )

create_range_partitions(relation       REGCLASS,  -- 主表OID
                        attribute      TEXT,      -- 分区列名
                        start_value    ANYELEMENT,  -- 开始值
                        p_interval     INTERVAL,    -- 间隔；interval
 类型，用于时间分区表
                        p_count        INTEGER DEFAULT NULL,   --
  分多少个区
                        partition_data BOOLEAN DEFAULT TRUE)   --
  是否立即将数据从主表迁移到分区，不建议这么使用，建议使用非堵塞式的迁移( 调用
partition_table_concurrently() )
```

另外两个可以指定起始值、终值、间隔，其定义如下：

```
create_partitions_from_range(relation       REGCLASS,  -- 主表OID
                             attribute      TEXT,      -- 分区列名
```

```
                                start_value    ANYELEMENT,  -- 开始值
                                end_value      ANYELEMENT,  -- 结束值
                                p_interval     ANYELEMENT,  -- 间隔; 任意
类型, 适合任意类型的分区表
                                partition_data BOOLEAN DEFAULT TRUE)
 --  是否立即将数据从主表迁移到分区, 不建议这么使用, 建议使用非堵塞式的迁移( 调
用partition_table_concurrently() )

create_partitions_from_range(relation       REGCLASS,  -- 主表OID
                                attribute      TEXT,      -- 分区列名
                                start_value    ANYELEMENT,  -- 开始值
                                end_value      ANYELEMENT,  -- 结束值
                                p_interval     INTERVAL,    -- 间隔;
interval 类型, 用于时间分区表
                                partition_data BOOLEAN DEFAULT TRUE)
 --  是否立即将数据从主表迁移到分区, 不建议这么使用, 建议使用非堵塞式的迁移( 调
用partition_table_concurrently() )
```

示例如下所示:

```
创建需要分区的主表
postgres=# create table part_test(id int, info text, crt_time
timestamp not null);  -- 分区列必须有not null约束
CREATE TABLE

插入一批测试数据, 模拟已经有数据了的主表
postgres=# insert into part_test select id,md5(random()::text),
clock_timestamp() + (id||' hour')::interval from generate_series(1,
10000) t(id);
INSERT 0 10000
postgres=# select * from part_test limit 10;
 id |               info               |          crt_time
----+----------------------------------+----------------------------
  1 | 36fe1adedaa5b848caec4941f87d443a | 2016-10-25 10:27:13.206713
  2 | c7d7358e196a9180efb4d0a10269c889 | 2016-10-25 11:27:13.206893
  3 | 005bdb063550579333264b895df5b75e | 2016-10-25 12:27:13.206904
  4 | 6c900a0fc50c6e4da1ae95447c89dd55 | 2016-10-25 13:27:13.20691
  5 | 857214d8999348ed3cb0469b520dc8e5 | 2016-10-25 14:27:13.206916
  6 | 4495875013e96e625afbf2698124ef5b | 2016-10-25 15:27:13.206921
  7 | 82488cf7e44f87d9b879c70a9ed407d4 | 2016-10-25 16:27:13.20693
  8 | a0b92547c8f17f79814dfbb12b8694a0 | 2016-10-25 17:27:13.206936
  9 | 2ca09e0b85042b476fc235e75326b41b | 2016-10-25 18:27:13.206942
 10 | 7eb762e1ef7dca65faf413f236dff93d | 2016-10-25 19:27:13.206947
(10 rows)

注意:
1. 分区列必须有not null约束
2. 分区个数必须能覆盖已有的所有记录

创建分区, 每个分区包含1个月的跨度数据
postgres=# select
create_range_partitions('part_test'::regclass,             -- 主表OID
                        'crt_time',                        -- 分区列
名
                        '2016-10-25 00:00:00'::timestamp,  -- 开始值
                        interval '1 month',                -- 间隔;
interval 类型, 用于时间分区表
                        24,                                -- 分多少
个区
                        false) ;                           -- 不迁移
数据
NOTICE:  sequence "part_test_seq" does not exist, skipping
 create_range_partitions
```

```
-------------------------
                       24
(1 row)
postgres-# \d+ part_test
                                    Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Child tables: part_test_1,
              part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_2,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_3,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
              part_test_8,
              part_test_9


由于不迁移数据，所以数据还在主表
postgres=# select count(*) from only part_test;
 count
-------
 10000
(1 row)


使用非堵塞式的迁移接口
partition_table_concurrently(relation   REGCLASS,              -- 主
表OID
                             batch_size INTEGER DEFAULT 1000,  -- 一
个事务批量迁移多少记录
                             sleep_time FLOAT8 DEFAULT 1.0)    -- 获
得行锁失败时，休眠多久再次获取，重试60次退出任务。


postgres=# select partition_table_concurrently('part_test'::regclass
,
                             10000,
                             1.0);
NOTICE:  worker started, you can stop it with the following command
: select stop_concurrent_part_task('part_test');
```

```
 partition_table_concurrently
------------------------------

(1 row)


迁移结束后，主表数据已经没有了，全部在分区中
postgres=# select count(*) from only part_test;
 count
-------
     0
(1 row)


数据迁移完成后，建议禁用主表，这样执行计划就不会出现主表了
postgres=# select set_enable_parent('part_test'::regclass, false);
 set_enable_parent
-------------------

(1 row)

postgres=# explain select * from part_test where crt_time = '2016-10
-25 00:00:00'::timestamp;
                                    QUERY PLAN

--------------------------------------------------------------------------------
 Append  (cost=0.00..16.18 rows=1 width=45)
   ->  Seq Scan on part_test_1  (cost=0.00..16.18 rows=1 width=45)
         Filter: (crt_time = '2016-10-25 00:00:00'::timestamp
without time zone)
(3 rows)
```

说明:

在RANGE分区表使用过程中，建议您：

- **分区列必须有not null约束。**
- **分区个数必须能覆盖已有的所有记录。**
- **使用非堵塞式迁移接口。**
- **数据迁移完成后，禁用主表。**

2. HASH分区

有一个管理函数用来创建范围分区，可以指定起始值、间隔、分区个数，具体如下：

```
create_hash_partitions(relation         REGCLASS,  -- 主表OID
                       attribute        TEXT,      -- 分区列名
                       partitions_count INTEGER,   -- 打算创建多少个分
区
                       partition_data   BOOLEAN DEFAULT TRUE)   --
  是否立即将数据从主表迁移到分区，不建议这么使用，建议使用非堵塞式的迁移( 调用
partition_table_concurrently() )
```

示例如下所示：

```
创建需要分区的主表
```

```
postgres=# create table part_test(id int, info text, crt_time
timestamp not null);    -- 分区列必须有not null约束
CREATE TABLE

插入一批测试数据，模拟已经有数据了的主表
postgres=# insert into part_test select id,md5(random()::text),
clock_timestamp() + (id||' hour')::interval from generate_series(1,
10000) t(id);
INSERT 0 10000
postgres=# select * from part_test limit 10;
 id |               info               |          crt_time
----+----------------------------------+----------------------------
  1 | 29ce4edc70dbfbe78912beb7c4cc95c2 | 2016-10-25 10:47:32.873879
  2 | e0990a6fb5826409667c9eb150fef386 | 2016-10-25 11:47:32.874048
  3 | d25f577a01013925c203910e34470695 | 2016-10-25 12:47:32.874059
  4 | 501419c3f7c218e562b324a1bebfe0ad | 2016-10-25 13:47:32.874065
  5 | 5e5e22bdf110d66a5224a657955ba158 | 2016-10-25 14:47:32.87407
  6 | 55d2d4fd5229a6595e0dd56e13d32be4 | 2016-10-25 15:47:32.874076
  7 | 1dfb9a783af55b123c7a888afe1eb950 | 2016-10-25 16:47:32.874081
  8 | 41eeb0bf395a4ab1e08691125ae74bff | 2016-10-25 17:47:32.874087
  9 | 83783d69cc4f9bb41a3978fe9e13d7fa | 2016-10-25 18:47:32.874092
 10 | affc9406d5b3412ae31f7d7283cda0dd | 2016-10-25 19:47:32.874097
(10 rows)

注意:
1. 分区列必须有not null约束

创建128个分区
postgres=# select
create_hash_partitions('part_test'::regclass,              -- 主表OID
                       'crt_time',                         -- 分区列
名
                       128,                                -- 打算创
建多少个分区
                       false) ;                            -- 不迁移
数据
 create_hash_partitions
------------------------
                    128
(1 row)

postgres=# \d+ part_test
                                  Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Child tables: part_test_0,
              part_test_1,
              part_test_10,
              part_test_100,
              part_test_101,
              part_test_102,
              part_test_103,
              part_test_104,
              part_test_105,
              part_test_106,
              part_test_107,
```

```
                part_test_108,
                part_test_109,
                part_test_11,
                part_test_110,
                part_test_111,
                part_test_112,
                part_test_113,
                part_test_114,
                part_test_115,
                part_test_116,
                part_test_117,
                part_test_118,
                part_test_119,
                part_test_12,
                part_test_120,
                part_test_121,
                part_test_122,
                part_test_123,
                part_test_124,
                part_test_125,
                part_test_126,
                part_test_127,
                part_test_13,
                part_test_14,
                part_test_15,
                part_test_16,
                part_test_17,
                part_test_18,
                part_test_19,
                part_test_2,
                part_test_20,
                part_test_21,
                part_test_22,
                part_test_23,
                part_test_24,
                part_test_25,
                part_test_26,
                part_test_27,
                part_test_28,
                part_test_29,
                part_test_3,
                part_test_30,
                part_test_31,
                part_test_32,
                part_test_33,
                part_test_34,
                part_test_35,
                part_test_36,
                part_test_37,
                part_test_38,
                part_test_39,
                part_test_4,
                part_test_40,
                part_test_41,
                part_test_42,
                part_test_43,
                part_test_44,
                part_test_45,
                part_test_46,
                part_test_47,
                part_test_48,
                part_test_49,
                part_test_5,
                part_test_50,
```

```
                part_test_51,
                part_test_52,
                part_test_53,
                part_test_54,
                part_test_55,
                part_test_56,
                part_test_57,
                part_test_58,
                part_test_59,
                part_test_6,
                part_test_60,
                part_test_61,
                part_test_62,
                part_test_63,
                part_test_64,
                part_test_65,
                part_test_66,
                part_test_67,
                part_test_68,
                part_test_69,
                part_test_7,
                part_test_70,
                part_test_71,
                part_test_72,
                part_test_73,
                part_test_74,
                part_test_75,
                part_test_76,
                part_test_77,
                part_test_78,
                part_test_79,
                part_test_8,
                part_test_80,
                part_test_81,
                part_test_82,
                part_test_83,
                part_test_84,
                part_test_85,
                part_test_86,
                part_test_87,
                part_test_88,
                part_test_89,
                part_test_9,
                part_test_90,
                part_test_91,
                part_test_92,
                part_test_93,
                part_test_94,
                part_test_95,
                part_test_96,
                part_test_97,
                part_test_98,
                part_test_99
```

由于不迁移数据，所以数据还在主表

```
postgres=# select count(*) from only part_test;
 count
-------
 10000
(1 row)
```

使用非堵塞式的迁移接口

```
partition_table_concurrently(relation   REGCLASS,               -- 主
表OID
                             batch_size INTEGER DEFAULT 1000,  -- 一
个事务批量迁移多少记录
                             sleep_time FLOAT8 DEFAULT 1.0)    -- 获
得行锁失败时，休眠多久再次获取，重试60次退出任务。


postgres=# select partition_table_concurrently('part_test'::regclass
,
                             10000,
                             1.0);
NOTICE:  worker started, you can stop it with the following command
: select stop_concurrent_part_task('part_test');
 partition_table_concurrently
------------------------------

(1 row)


迁移结束后，主表数据已经没有了，全部在分区中
postgres=# select count(*) from only part_test;
 count
-------
     0
(1 row)


数据迁移完成后，建议禁用主表，这样执行计划就不会出现主表了
postgres=# select set_enable_parent('part_test'::regclass, false);
 set_enable_parent
-------------------

(1 row)

只查单个分区
postgres=# explain select * from part_test where crt_time = '2016-10
-25 00:00:00'::timestamp;
                                  QUERY PLAN

------------------------------------------------------------------------------
 Append  (cost=0.00..1.91 rows=1 width=45)
   ->  Seq Scan on part_test_122  (cost=0.00..1.91 rows=1 width=45)
         Filter: (crt_time = '2016-10-25 00:00:00'::timestamp
without time zone)
(3 rows)

分区表约束如下
很显然pg_pathman自动完成了转换，如果是传统的继承，select * from part_test
where crt_time = '2016-10-25 00:00:00'::timestamp; 这种写法是不能筛选分
区的。
postgres=# \d+ part_test_122
                                Table "public.part_test_122"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
```

```
    "pathman_part_test_122_3_check" CHECK (get_hash_part_idx(
timestamp_hash(crt_time), 128) = 122)
Inherits: part_test
```

说明:

在HASH分区表使用过程中，建议您：

- **分区列必须有not null约束。**
- **使用非堵塞式迁移接口。**
- **数据迁移完成后，禁用主表。**
- **pg_pathman不会受制于表达式的写法，所以**`select * from part_test where crt_time = '2016-10-25 00:00:00'::timestamp;`**这样的写法也能用于HASH分区的。**
- **HASH分区列不局限于int类型的列，会使用HASH函数自动转换。**

3. **数据迁移到分区**

   如果创建分区表时，未将主表数据迁移到分区，那么可以使用非堵塞式的迁移接口，将数据迁移到分区。用法如下：

```
with tmp as (delete from 主表 limit xx nowait returning *) insert
into 分区 select * from tmp

或者使用 select array_agg(ctid) from 主表 limit xx for update nowati
 进行标示 然后执行delete和insert。
```

   函数接口如下：

```
partition_table_concurrently(relation   REGCLASS,               -- 主
表OID
                             batch_size INTEGER DEFAULT 1000,   -- 一
个事务批量迁移多少记录
                             sleep_time FLOAT8 DEFAULT 1.0)     -- 获
得行锁失败时，休眠多久再次获取，重试60次退出任务。
```

   示例如下所示：

```
postgres=# select partition_table_concurrently('part_test'::regclass
,
                             10000,
                             1.0);
NOTICE:  worker started, you can stop it with the following command
: select stop_concurrent_part_task('part_test');
 partition_table_concurrently
------------------------------
```

```
(1 row)
```

如果停止迁移任务，调用如下函数接口：

```
stop_concurrent_part_task(relation REGCLASS)
```

查看后台的数据迁移任务。

```
postgres=# select * from pathman_concurrent_part_tasks;
 userid | pid | dbid | relid | processed | status
--------+-----+------+-------+-----------+--------
(0 rows)
```

4. 分裂范围分区

如果某个分区太大，想分裂为两个分区，可以使用如下方法（目前仅支持RANGE分区表）：

```
split_range_partition(partition      REGCLASS,            -- 分区oid
                      split_value    ANYELEMENT,          -- 分裂值
                      partition_name TEXT DEFAULT NULL)   -- 分裂后新
增的分区表名
```

示例如下所示：

```
postgres=# \d+ part_test
                                  Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Child tables: part_test_1,
              part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_2,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_3,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
              part_test_8,
              part_test_9
```

```
postgres=# \d+ part_test_1
                                  Table "public.part_test_1"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:
00:00'::timestamp without time zone AND crt_time < '2016-11-25 00:00
:00'::timestamp without time zone)
Inherits: part_test
```

分裂

```
postgres=# select split_range_partition('part_test_1'::regclass,
          -- 分区oid
                      '2016-11-10 00:00:00'::timestamp,     -- 分裂值
                      'part_test_1_2');                     -- 分区表
名
             split_range_partition
-----------------------------------------------
 {"2016-10-25 00:00:00","2016-11-25 00:00:00"}
(1 row)
```

分裂后的两个表如下:

```
postgres=# \d+ part_test_1
                                  Table "public.part_test_1"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:
00:00'::timestamp without time zone AND crt_time < '2016-11-10 00:00
:00'::timestamp without time zone)
Inherits: part_test

postgres=# \d+ part_test_1_2
                                Table "public.part_test_1_2"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
```

```
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_1_2_3_check" CHECK (crt_time >= '2016-11-10
00:00:00'::timestamp without time zone AND crt_time < '2016-11-25 00
:00:00'::timestamp without time zone)
Inherits: part_test
```

数据会自动迁移到另一个分区。

```
postgres=# select count(*) from part_test_1;
 count
-------
   373
(1 row)

postgres=# select count(*) from part_test_1_2;
 count
-------
   360
(1 row)
```

继承关系如下：

```
postgres=# \d+ part_test
                                  Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Child tables: part_test_1,
              part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_1_2,    -- 新增的表
              part_test_2,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_3,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
              part_test_8,
```

```
                part_test_9
```

5. **合并范围分区**

   目前仅支持RANGE 分区，调用如下接口：

```
指定两个需要合并分区，必须为相邻分区
merge_range_partitions(partition1 REGCLASS, partition2 REGCLASS)
```

   示例如下所示：

```
postgres=# select merge_range_partitions('part_test_2'::regclass, '
part_test_12'::regclass) ;
ERROR:  merge failed, partitions must be adjacent
CONTEXT:  PL/pgSQL function merge_range_partitions_internal(regclass
,regclass,regclass,anyelement) line 27 at RAISE
SQL statement "SELECT public.merge_range_partitions_internal($1, $2,
 $3, NULL::timestamp without time zone)"
PL/pgSQL function merge_range_partitions(regclass,regclass) line 44
at EXECUTE
不是相邻分区，报错

相邻分区可以合并
postgres=# select merge_range_partitions('part_test_1'::regclass, '
part_test_1_2'::regclass) ;
 merge_range_partitions
------------------------

(1 row)
```

   合并后，会删掉其中一个分区表。

```
postgres=# \d part_test_1_2
Did not find any relation named "part_test_1_2".

postgres=# \d part_test_1
             Table "public.part_test_1"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 id       | integer                     |
 info     | text                        |
 crt_time | timestamp without time zone | not null
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:
00:00'::timestamp without time zone AND crt_time < '2016-11-25 00:00
:00'::timestamp without time zone)
Inherits: part_test

postgres=# select count(*) from part_test_1;
 count
-------
   733
```

```
(1 row)
```

6. **向后添加范围分区**

   如果已经对主表进行了分区，将来需要增加分区的话，有几种方法，一种是向后新增分区（即在末尾追加分区）。

   新增分区时，会使用初次创建该分区表时的interval作为间隔。可以在pathman_config中查询每个分区表初次创建时的interval，如下：

```
postgres=# select * from pathman_config;
  partrel  | attname  | parttype | range_interval
-----------+----------+----------+----------------
 part_test | crt_time |        2 | 1 mon
(1 row)
```

   添加分区接口（目前不支持指定表空间）

```
append_range_partition(parent         REGCLASS,             -- 主表OID
                       partition_name TEXT DEFAULT NULL,    -- 新增的
分区表名，默认不需要输入
                       tablespace     TEXT DEFAULT NULL)    -- 新增的
分区表放到哪个表空间，默认不需要输入
```

   示例如下所示：

```
postgres=# select append_range_partition('part_test'::regclass);
 append_range_partition
------------------------
 public.part_test_25
(1 row)

postgres=# \d+ part_test_25
                                Table "public.part_test_25"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_25_3_check" CHECK (crt_time >= '2018-10-25 00
:00:00'::timestamp without time zone AND crt_time < '2018-11-25 00:
00:00'::timestamp without time zone)
Inherits: part_test

postgres=# \d+ part_test_24
                                Table "public.part_test_24"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
```

```
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_24_3_check" CHECK (crt_time >= '2018-09-25 00
:00:00'::timestamp without time zone AND crt_time < '2018-10-25 00:
00:00'::timestamp without time zone)
Inherits: part_test
```

7. 向前添加范围分区

在头部追加分区，接口如下：

```
prepend_range_partition(parent         REGCLASS,
                        partition_name TEXT DEFAULT NULL,
                        tablespace     TEXT DEFAULT NULL)
```

示例如下所示：

```
postgres=# select prepend_range_partition('part_test'::regclass);
 prepend_range_partition
-------------------------
 public.part_test_26
(1 row)

postgres=# \d+ part_test_26
                                Table "public.part_test_26"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_26_3_check" CHECK (crt_time >= '2016-09-25 00
:00:00'::timestamp without time zone AND crt_time < '2016-10-25 00:
00:00'::timestamp without time zone)
Inherits: part_test

postgres=# \d+ part_test_1
                                 Table "public.part_test_1"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:
00:00'::timestamp without time zone AND crt_time < '2016-11-25 00:00
:00'::timestamp without time zone)
```

```
Inherits: part_test
```

8. 添加分区

指定分区起始值的方式添加分区，只要创建的分区和已有分区不会存在数据交叉就可以创建成功。也就是说使用这种方法，不要求强制创建连续的分区，例如已有分区覆盖了2010-2015的范围，您可以直接创建一个2020年的分区表，不需要覆盖2015到2020的范围。接口如下：

```
add_range_partition(relation       REGCLASS,    -- 主表OID
                    start_value    ANYELEMENT,  -- 起始值
                    end_value      ANYELEMENT,  -- 结束值
                    partition_name TEXT DEFAULT NULL,  -- 分区名
                    tablespace     TEXT DEFAULT NULL)  -- 分区创建在哪
个表空间下
```

示例如下所示：

```
postgres=# select add_range_partition('part_test'::regclass,
 -- 主表OID
                    '2020-01-01 00:00:00'::timestamp,  -- 起始值
                    '2020-02-01 00:00:00'::timestamp); -- 结束值
 add_range_partition
---------------------
 public.part_test_27
(1 row)

postgres=# \d+ part_test_27
                                Table "public.part_test_27"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_27_3_check" CHECK (crt_time >= '2020-01-01 00
:00:00'::timestamp without time zone AND crt_time < '2020-02-01 00:
00:00'::timestamp without time zone)
Inherits: part_test
```

9. 删除分区

删除单个范围分区，接口如下：

```
drop_range_partition(partition TEXT,   -- 分区名称
                     delete_data BOOLEAN DEFAULT TRUE)  -- 是否删除分区
数据，如果false，表示分区数据迁移到主表。

Drop RANGE partition and all of its data if delete_data is true.
```

示例如下所示：

```
删除分区，  数据迁移到主表
postgres=# select drop_range_partition('part_test_1',false);
```

```
NOTICE:  733 rows copied from part_test_1
 drop_range_partition
----------------------
 part_test_1
(1 row)

postgres=# select drop_range_partition('part_test_2',false);
NOTICE:  720 rows copied from part_test_2
 drop_range_partition
----------------------
 part_test_2
(1 row)

postgres=# select count(*) from part_test;
 count
-------
 10000
(1 row)

删除分区，分区数据也删除，不迁移到主表
postgres=# select drop_range_partition('part_test_3',true);
 drop_range_partition
----------------------
 part_test_3
(1 row)

postgres=# select count(*) from part_test;
 count
-------
  9256
(1 row)

postgres=# select count(*) from only part_test;
 count
-------
  1453
(1 row)
```

删除所有分区，并且指定是否要将数据迁移到主表。接口如下：

```
drop_partitions(parent      REGCLASS,
                delete_data BOOLEAN DEFAULT FALSE)

Drop partitions of the parent table (both foreign and local
relations).
If delete_data is false, the data is copied to the parent table
first.
Default is false.
```

示例如下所示：

```
postgres=# select drop_partitions('part_test'::regclass, false);
 -- 删除所有分区表，并将数据迁移到主表
NOTICE:  function public.part_test_upd_trig_func() does not exist,
skipping
NOTICE:  744 rows copied from part_test_4
NOTICE:  672 rows copied from part_test_5
NOTICE:  744 rows copied from part_test_6
NOTICE:  720 rows copied from part_test_7
NOTICE:  744 rows copied from part_test_8
NOTICE:  720 rows copied from part_test_9
NOTICE:  744 rows copied from part_test_10
```

```
NOTICE:  744 rows copied from part_test_11
NOTICE:  720 rows copied from part_test_12
NOTICE:  744 rows copied from part_test_13
NOTICE:  507 rows copied from part_test_14
NOTICE:  0 rows copied from part_test_15
NOTICE:  0 rows copied from part_test_16
NOTICE:  0 rows copied from part_test_17
NOTICE:  0 rows copied from part_test_18
NOTICE:  0 rows copied from part_test_19
NOTICE:  0 rows copied from part_test_20
NOTICE:  0 rows copied from part_test_21
NOTICE:  0 rows copied from part_test_22
NOTICE:  0 rows copied from part_test_23
NOTICE:  0 rows copied from part_test_24
NOTICE:  0 rows copied from part_test_25
NOTICE:  0 rows copied from part_test_26
NOTICE:  0 rows copied from part_test_27
 drop_partitions
-----------------
              24
(1 row)

postgres=# select count(*) from part_test;
 count
-------
  9256
(1 row)

postgres=# \dt part_test_4
No matching relations found.
```

10绑定分区（已有的表加入分区表）

将已有的表，绑定到已有的某个分区主表。已有的表与主表要保持一致的结构，包括dropped columns (查看pg_attribute的一致性)。接口如下：

```
attach_range_partition(relation    REGCLASS,    -- 主表OID
                       partition   REGCLASS,    -- 分区表OID
                       start_value ANYELEMENT,  -- 起始值
                       end_value   ANYELEMENT)  -- 结束值
```

示例如下所示：

```
postgres=# create table part_test_1 (like part_test including all);
CREATE TABLE
postgres=# \d+ part_test
                                  Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |

postgres=# \d+ part_test_1
                                Table "public.part_test_1"
```

```
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |

postgres=# select attach_range_partition('part_test'::regclass, '
part_test_1'::regclass, '2019-01-01 00:00:00'::timestamp, '2019-02-
01 00:00:00'::timestamp);
 attach_range_partition
------------------------
 part_test_1
(1 row)
```

绑定分区时，
自动创建继承关系，自动创建约束

```
postgres=# \d+ part_test_1
                                  Table "public.part_test_1"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2019-01-01 00:
00:00'::timestamp without time zone AND crt_time < '2019-02-01 00:00
:00'::timestamp without time zone)
Inherits: part_test
```

11解绑分区（将分区变成普通表）

将分区从主表的继承关系中删除，不删数据，删除继承关系，删除约束。接口如下：

```
detach_range_partition(partition REGCLASS)  -- 指定分区名，转换为普通表
```

示例如下所示：

```
postgres=# select count(*) from part_test;
 count
-------
  9256
(1 row)

postgres=# select count(*) from part_test_2;
 count
-------
   733
(1 row)

postgres=# select detach_range_partition('part_test_2');
 detach_range_partition
------------------------
```

```
 part_test_2
(1 row)

postgres=# select count(*) from part_test_2;
 count
-------
   733
(1 row)

postgres=# select count(*) from part_test;
 count
-------
  8523
(1 row)
```

12永久禁止分区表pg_pathman插件

您可以针对单个分区主表禁用pg_pathman。接口函数如下：

```
disable_pathman_for(relation TEXT)

Permanently disable pg_pathman partitioning mechanism for the
specified parent table and remove the insert trigger if it exists.
All partitions and data remain unchanged.

postgres=# \sf disable_pathman_for
CREATE OR REPLACE FUNCTION public.disable_pathman_for(parent_relid
regclass)
 RETURNS void
 LANGUAGE plpgsql
 STRICT
AS $function$
BEGIN
        PERFORM public.validate_relname(parent_relid);

        DELETE FROM public.pathman_config WHERE partrel = parent_rel
id;
        PERFORM public.drop_triggers(parent_relid);

        /* Notify backend about changes */
        PERFORM public.on_remove_partitions(parent_relid);
END
$function$
```

示例如下所示：

```
postgres=# select disable_pathman_for('part_test');
NOTICE:  drop cascades to 23 other objects
DETAIL:  drop cascades to trigger part_test_upd_trig on table
part_test_3
drop cascades to trigger part_test_upd_trig on table part_test_4
drop cascades to trigger part_test_upd_trig on table part_test_5
drop cascades to trigger part_test_upd_trig on table part_test_6
drop cascades to trigger part_test_upd_trig on table part_test_7
drop cascades to trigger part_test_upd_trig on table part_test_8
drop cascades to trigger part_test_upd_trig on table part_test_9
drop cascades to trigger part_test_upd_trig on table part_test_10
drop cascades to trigger part_test_upd_trig on table part_test_11
drop cascades to trigger part_test_upd_trig on table part_test_12
drop cascades to trigger part_test_upd_trig on table part_test_13
drop cascades to trigger part_test_upd_trig on table part_test_14
drop cascades to trigger part_test_upd_trig on table part_test_15
```

```
drop cascades to trigger part_test_upd_trig on table part_test_16
drop cascades to trigger part_test_upd_trig on table part_test_17
drop cascades to trigger part_test_upd_trig on table part_test_18
drop cascades to trigger part_test_upd_trig on table part_test_19
drop cascades to trigger part_test_upd_trig on table part_test_20
drop cascades to trigger part_test_upd_trig on table part_test_21
drop cascades to trigger part_test_upd_trig on table part_test_22
drop cascades to trigger part_test_upd_trig on table part_test_23
drop cascades to trigger part_test_upd_trig on table part_test_24
drop cascades to trigger part_test_upd_trig on table part_test_25
 disable_pathman_for
---------------------

(1 row)

postgres=# \d+ part_test
                                  Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Child tables: part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_25,
              part_test_26,
              part_test_27,
              part_test_28,
              part_test_29,
              part_test_3,
              part_test_30,
              part_test_31,
              part_test_32,
              part_test_33,
              part_test_34,
              part_test_35,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
              part_test_8,
              part_test_9

postgres=# \d+ part_test_10
                                Table "public.part_test_10"
```

```
  Column   |            Type             | Modifiers | Storage  |
Stats target | Description
-----------+-----------------------------+-----------+----------
+--------------+-------------
 id        | integer                     |           | plain    |
           |
 info      | text                        |           | extended |
           |
 crt_time  | timestamp without time zone | not null  | plain    |
           |
Check constraints:
    "pathman_part_test_10_3_check" CHECK (crt_time >= '2017-06-25 00
:00:00'::timestamp without time zone AND crt_time < '2017-07-25 00:
00:00'::timestamp without time zone)
Inherits: part_test
```

禁用pg_pathman插件后，继承关系和约束不会变化，只是pg_pathman插件不介入custom scan执行计划。禁用pg_pathman插件后的执行计划如下：

```
postgres=# explain select * from part_test where crt_time='2017-06-
25 00:00:00'::timestamp;
                                  QUERY PLAN

-------------------------------------------------------------------------------
 Append  (cost=0.00..16.00 rows=2 width=45)
   ->  Seq Scan on part_test  (cost=0.00..0.00 rows=1 width=45)
         Filter: (crt_time = '2017-06-25 00:00:00'::timestamp
without time zone)
   ->  Seq Scan on part_test_10  (cost=0.00..16.00 rows=1 width=45)
         Filter: (crt_time = '2017-06-25 00:00:00'::timestamp
without time zone)
(5 rows)
```

注意：

`disable_pathman_for`**没有可逆操作，请慎使用。**

高级分区管理

1. **禁用主表**

   当主表的数据全部迁移到分区后，可以禁用主表。接口函数如下：

```
set_enable_parent(relation REGCLASS, value BOOLEAN)


Include/exclude parent table into/from query plan.

In original PostgreSQL planner parent table is always included into
 query plan even if it's empty which can lead to additional overhead
.

You can use disable_parent() if you are never going to use parent
table as a storage.

Default value depends on the partition_data parameter that was
specified during initial partitioning in create_range_partitions()
or create_partitions_from_range() functions.
```

```
If the partition_data parameter was true then all data have already
been migrated to partitions and parent table disabled.

Otherwise it is enabled.
```

示例如下所示：

```
select set_enable_parent('part_test', false);
```

2. **自动扩展分区**

范围分区表，允许自动扩展分区。如果新插入的数据不在已有的分区范围内，会自动创建分区。

```
set_auto(relation REGCLASS, value BOOLEAN)

Enable/disable auto partition propagation (only for RANGE partitioni
ng).

It is enabled by default.
```

示例如下所示：

```
postgres=# \d+ part_test
                                    Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Child tables: part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_25,
              part_test_26,
              part_test_3,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
              part_test_8,
              part_test_9

postgres=# \d+ part_test_26
                                  Table "public.part_test_26"
```

```
  Column   |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_26_3_check" CHECK (crt_time >= '2018-09-25 00
:00:00'::timestamp without time zone AND crt_time < '2018-10-25 00:
00:00'::timestamp without time zone)
Inherits: part_test

postgres=# \d+ part_test_25
                                Table "public.part_test_25"
  Column   |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Check constraints:
    "pathman_part_test_25_3_check" CHECK (crt_time >= '2018-08-25 00
:00:00'::timestamp without time zone AND crt_time < '2018-09-25 00:
00:00'::timestamp without time zone)
Inherits: part_test
```

插入一个不在已有分区范围的值，会根据创建分区时的interval自动扩展若干个分区，这个操作可能很久。

```
postgres=# insert into part_test values (1,'test','2222-01-01'::
timestamp);
```

插入结束后，扩展了好多分区，原因是插入的值跨度范围太大了。

```
postgres=# \d+ part_test
                                  Table "public.part_test"
  Column   |            Type             | Modifiers | Storage  |
Stats target | Description
----------+-----------------------------+-----------+----------
+--------------+-------------
 id       | integer                     |           | plain    |
          |
 info     | text                        |           | extended |
          |
 crt_time | timestamp without time zone | not null  | plain    |
          |
Child tables: part_test_10,
              part_test_100,
              part_test_1000,
              part_test_1001,
        ....................................
```

很多

说明:

不建议开启自动扩展范围分区，不合理的自动扩展可能会消耗大量的时间。

3. 回调函数（创建每个分区时都会触发）

回调函数是在每创建一个分区时会自动触发调用的函数。例如，可以用在DDL逻辑复制中，将DDL语句记录下来，存放到表中。回调函数如下：

```
set_init_callback(relation REGCLASS, callback REGPROC DEFAULT 0)

Set partition creation callback to be invoked for each attached or
created partition (both HASH and RANGE).

The callback must have the following signature:

part_init_callback(args JSONB) RETURNS VOID.

Parameter arg consists of several fields whose presence depends on
partitioning type:

/* RANGE-partitioned table abc (child abc_4) */
{
    "parent":    "abc",
    "parttype":  "2",
    "partition": "abc_4",
    "range_max": "401",
    "range_min": "301"
}

/* HASH-partitioned table abc (child abc_0) */
{
    "parent":    "abc",
    "parttype":  "1",
    "partition": "abc_0"
}
```

示例如下所示：

```
回调函数
postgres=# create or replace function f_callback_test(jsonb) returns
 void as
$$
declare
begin
  create table if not exists rec_part_ddl(id serial primary key,
parent name, parttype int, partition name, range_max text, range_min
 text);
  if ($1->>'parttype')::int = 1 then
    raise notice 'parent: %, parttype: %, partition: %', $1->>'
parent', $1->>'parttype', $1->>'partition';
    insert into rec_part_ddl(parent, parttype, partition) values (($
1->>'parent')::name, ($1->>'parttype')::int, ($1->>'partition')::
name);
  elsif ($1->>'parttype')::int = 2 then
    raise notice 'parent: %, parttype: %, partition: %, range_max:
 %, range_min: %', $1->>'parent', $1->>'parttype', $1->>'partition',
 $1->>'range_max', $1->>'range_min';
    insert into rec_part_ddl(parent, parttype, partition, range_max
, range_min) values (($1->>'parent')::name, ($1->>'parttype')::int,
 ($1->>'partition')::name, $1->>'range_max', $1->>'range_min');
  end if;
end;
```

```
$$ language plpgsql strict;

测试表
postgres=# create table tt(id int, info text, crt_time timestamp not
 null);
CREATE TABLE

设置测试表的回调函数
select set_init_callback('tt'::regclass, 'f_callback_test'::regproc
);

创建分区
postgres=# select

create_range_partitions('tt'::regclass,                    -- 主表OID
                        'crt_time',                        -- 分区列
名
                        '2016-10-25 00:00:00'::timestamp,  -- 开始值
                        interval '1 month',                -- 间隔;
interval 类型，用于时间分区表
                        24,                                -- 分多少
个区
                        false) ;
 create_range_partitions
-------------------------
                      24
(1 row)

检查回调函数是否已调用
postgres=# select * from rec_part_ddl;
 id | parent | parttype | partition |      range_max      |
range_min
----+--------+----------+-----------+---------------------
+---------------------
  1 | tt     |        2 | tt_1      | 2016-11-25 00:00:00 | 2016-10-
25 00:00:00
  2 | tt     |        2 | tt_2      | 2016-12-25 00:00:00 | 2016-11-
25 00:00:00
  3 | tt     |        2 | tt_3      | 2017-01-25 00:00:00 | 2016-12-
25 00:00:00
  4 | tt     |        2 | tt_4      | 2017-02-25 00:00:00 | 2017-01-
25 00:00:00
  5 | tt     |        2 | tt_5      | 2017-03-25 00:00:00 | 2017-02-
25 00:00:00
  6 | tt     |        2 | tt_6      | 2017-04-25 00:00:00 | 2017-03-
25 00:00:00
  7 | tt     |        2 | tt_7      | 2017-05-25 00:00:00 | 2017-04-
25 00:00:00
  8 | tt     |        2 | tt_8      | 2017-06-25 00:00:00 | 2017-05-
25 00:00:00
  9 | tt     |        2 | tt_9      | 2017-07-25 00:00:00 | 2017-06-
25 00:00:00
 10 | tt     |        2 | tt_10     | 2017-08-25 00:00:00 | 2017-07-
25 00:00:00
 11 | tt     |        2 | tt_11     | 2017-09-25 00:00:00 | 2017-08-
25 00:00:00
 12 | tt     |        2 | tt_12     | 2017-10-25 00:00:00 | 2017-09-
25 00:00:00
 13 | tt     |        2 | tt_13     | 2017-11-25 00:00:00 | 2017-10-
25 00:00:00
 14 | tt     |        2 | tt_14     | 2017-12-25 00:00:00 | 2017-11-
25 00:00:00
 15 | tt     |        2 | tt_15     | 2018-01-25 00:00:00 | 2017-12-
25 00:00:00
```

```
 16 | tt     |         2 | tt_16     | 2018-02-25 00:00:00 | 2018-01-
25 00:00:00
 17 | tt     |         2 | tt_17     | 2018-03-25 00:00:00 | 2018-02-
25 00:00:00
 18 | tt     |         2 | tt_18     | 2018-04-25 00:00:00 | 2018-03-
25 00:00:00
 19 | tt     |         2 | tt_19     | 2018-05-25 00:00:00 | 2018-04-
25 00:00:00
 20 | tt     |         2 | tt_20     | 2018-06-25 00:00:00 | 2018-05-
25 00:00:00
 21 | tt     |         2 | tt_21     | 2018-07-25 00:00:00 | 2018-06-
25 00:00:00
 22 | tt     |         2 | tt_22     | 2018-08-25 00:00:00 | 2018-07-
25 00:00:00
 23 | tt     |         2 | tt_23     | 2018-09-25 00:00:00 | 2018-08-
25 00:00:00
 24 | tt     |         2 | tt_24     | 2018-10-25 00:00:00 | 2018-09-
25 00:00:00
(24 rows)
```

18 使用中文分词

本文为您介绍POLARDB for PostgreSQL如何启用中文分词以及自定义中文分词词典。

启用中文分词

可以使用下面的命令，启用中文分词：

```
CREATE EXTENSION zhparser;
CREATE TEXT SEARCH CONFIGURATION testzhcfg (PARSER = zhparser);
ALTER TEXT SEARCH CONFIGURATION testzhcfg ADD MAPPING FOR n,v,a,i,e,l
WITH simple;
--可选的参数设定
alter role all set zhparser.multi_short=on;
--简单测试
SELECT * FROM ts_parse('zhparser', 'hello world! 2010年保障房建设在全国范
围内获全面启动，从中央到地方纷纷加大了保障房的建设和投入力度。2011
年，保障房进入了更大规模的建设阶段。住房城乡建设部党组书记、部长姜伟新去年底在全国住
房城乡建设工作会议上表示，要继续推进保障性安居工程建设。');
SELECT to_tsvector('testzhcfg','“今年保障房新开工数量虽然有所下调，但实际的年
度在建规模以及竣工规模会超以往年份，相对应的对资金的需求也会创历史纪录。”陈国强说。
在他看来，与2011年相比，2012年的保障房建设在资金配套上的压力将更为严峻。');
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

利用分词进行全文索引的方法如下：

```
--为T1表的name字段创建全文索引
create index idx_t1 on t1 using gin (to_tsvector('zhcfg',upper(name
) ));
--使用全文索引
 select * from t1 where to_tsvector('zhcfg',upper(t1.name)) @@
to_tsquery('zhcfg','(防火)') ;
```

自定义中文分词词典

自定义中文分词词典，示例如下：

```
-- 确实的分词结果
SELECT to_tsquery('testzhcfg', '保障房资金压力');
-- 往自定义分词词典里面插入新的分词
insert into pg_ts_custom_word values ('保障房资');
-- 使新的分词生效
select zhprs_sync_dict_xdb();
-- 退出此连接
\c
-- 重新查询，可以得到新的分词结果
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

使用自定义分词的注意事项如下：

- **最多支持一百万条自定义分词，超出部分不做处理，必须保证分词数量在这个范围之内。自定义分词与缺省的分词词典将共同产生作用。**
- **每个词的最大长度为128字节，超出部分将会截取。**

- **增删改分词之后必须执行**`select zhprs_sync_dict_xdb();`**并且重新建立连接才会生效**。

19 支持的插件列表

本文介绍POLARDB for PostgreSQL支持的插件列表。

POLARDB会随着内核版本的不断更新，不断支持新的插件或者新的插件版本，您可以执行以下语句来获取当前支持的插件列表。

```
show polar_supported_extensions;
```

以下是目前支持的插件和对应最新的版本号：

插件名	版本号	插件描述	兼容插件名（版本号）
btree_gin	1.3	提供一些通用类型的GIN索引。	-
btree_gist	1.5	提供一些通用类型的GIST索引。	-
citext	1.5	提供不区分大小写字符串类型。	-
cube	1.4	提供表示多维立方体的数据类型。	-
dict_int	1.0	支持数字的全文搜索词典模板。	-
earthdistance	1.1	提供两个不同方法计算地球表面上的大圆弧距离。	-
fuzzystrmatch	1.1	提供字符串之间的相似度计算。	-
hstore	1.5	提供键值对存储的类型。	-
intagg	1.1	提供一个整数聚集器和一个枚举器。	-
intarray	1.2	提供一维数组的相关运算符和函数，也支持使用其中的一些运算符执行索引搜索。	-
isn	1.2	提供国际产品标准序列号的数据类型。	-
ltree	1.1	提供树形结构的数据类型。	-
pg_buffercache	1.3	支持实时检查共享缓冲区。	-
pg_pathman	1.5	提供分区表管理。	-
pg_prewarm	1.2	提供表数据预热。	-
pg_stat_statements	1.6	提供查询所有执行过的SQL语句的统计信息。	-

插件名	版本号	插件描述	兼容插件名（版本号）
pg_trgm	1.4	提供基于三元模型（trigram）匹配的字母数字文本相似度的函数和操作符。	-
pg_wait_sampling	1.1	提供等待事件采样。	-
pgcrypto	1.3	提供加密函数。	-
pgrowlocks	1.2	提供表级锁信息查询。	-
pgstattuple	1.5	提供查询元组级别的统计信息。	-
plperl	1.0	提供PL/Perl函数语言。	-
plpgsql	1.0	提供PL/pgSQL函数语言。	-
pltcl	1.0	提供PL/Tcl函数语言。	-
sslinfo	1.2	提供查询当前客户端提供的SSL证书相关信息。	-
tablefunc	1.0	提供一组包含crosstab函数的整表操作。	-
timescaledb	1.3.0	提供时序数据库SQL查询。	-
unaccent	1.1	提供去除重音的文本搜索字典。	-
uuid-ossp	1.1	提供产生通用唯一标识符。	-
zhparser	1.0	提供中文分词。	-
ganos_geometry	2.3	提供地理几何数据类型。	postgis （2.5.0）
ganos_raster	2.3	提供地理栅格数据类型。	postgis （2.5.0）
ganos_geometry_sfcgal	2.3	提供地理几何3D模型计算。	postgis_sfcgal （2.5.0）
ganos_geometry_topology	2.3	提供地理几何拓扑关系计算。	postgis_topology （2.5.0）
ganos_tiger_geocode	2.3	提供tiger地理编码计算。	postgis_tiger_geocode （2.5.0）
ganos_address_standardizer	2.3	提供地址标准化计算。	address_standardizer （2.5.0）
ganos_address_standardizer_data_us	2.3	提供地址标准化美国数据。	address_standardizer_data_us （2.5.0）

插件名	版本号	插件描述	兼容插件名（版本号）
ganos_networking	2.3	提供几何网络计算。	pgrouting （2.6.2）
ganos_pointcloud	2.3	提供点云数据存储和计算。	-
ganos_trajectory	2.3	提供移动对象数据类型。	-

如果您还有其他插件的需求，可以提交工单联系我们。