云数据库 RDS for PostgreSQL

最佳实践

文档版本 01

发布日期 2025-09-04

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 最佳实践汇总

本文汇总了云数据库 RDS for PostgreSQL常见应用场景的操作实践，并为每个实践提供详细的方案描述和操作指导，帮助您轻松使用RDS for PostgreSQL。

表 1-1 RDS for PostgreSQL 最佳实践

相关文档	说明
RDS for PostgreSQL搭建跨区域容灾关系	介绍如何搭建RDS for PostgreSQL实例跨区域容灾。
RDS for PostgreSQL发布与订阅	介绍RDS for PostgreSQL提供的发布和订阅功能。
RDS for PostgreSQL自定义数据类型转换	介绍RDS for PostgreSQL自定义数据类型转换的语法使用。
使用客户端驱动程序实现故障转移和读写分离	介绍如何使用PostgreSQL客户端驱动程序实现故障转移和读写分离。
使用pg_waldump解析RDS for PostgreSQL的wal日志	介绍如何通过开源pg_waldump解析RDS for PostgreSQL的wal日志。
PoWA插件使用最佳实践	介绍如何使用PoWA插件，用于对RDS for PostgreSQL数据库进行性能监控。
pg_dump使用最佳实践	介绍pg_dump备份工具的使用方法。
PgBouncer使用最佳实践	介绍PgBouncer连接池工具的安装配置和使用方法。
创建只读用户最佳实践	介绍RDS for PostgreSQL创建只读用户的方法。
基于RDS for PostgreSQL的表设计入门	介绍针对业务场景设计RDS for PostgreSQL表结构的方法。
RDS for PostgreSQL权限管理最佳实践	介绍RDS for PostgreSQL的权限配置最佳实践。
WAL日志堆积问题定位及处理方法	介绍RDS for PostgreSQL WAL日志堆积的问题定位及处理方法。

相关文档	说明
批量更新、删除或插入数据	介绍如何高效实现RDS for PostgreSQL数据的批量插入、批量更新和批量删除，帮助开发者优化数据库性能。
使用事件触发器实现DDL回收站、防火墙、增量订阅同步	介绍基于PostgreSQL事件触发器实现DDL回收站、防火墙、增量订阅同步的最佳实践方案示例。
使用Replication Slot创建数据订阅	介绍在RDS for PostgreSQL实例上开启数据订阅的配置方法。
基于Pgpool实现读写分离	介绍RDS for PostgreSQL实例及只读实例如何结合Pgpool实现读写分离。
用户喜好推荐系统	介绍如何设计RDS for PostgreSQL用户推荐系统。
RDS for PostgreSQL指标告警配置建议	介绍设置RDS for PostgreSQL指标告警规则的配置及建议。
RDS for PostgreSQL安全最佳实践	提供RDS for PostgreSQL安全配置的规范性指导。

2 RDS for PostgreSQL 搭建跨区域容灾关系

2.1 方案概述

场景描述

当生产机发生损坏或因其他不可抗力造成业务系统宕机的情况下，异地跨区域容灾实例可以保证生产系统的数据不丢失，保持生产系统的业务不间断地运行，从而提高系统的可用性。

本实践主要包含以下内容：

- 介绍如何创建RDS for PostgreSQL实例。
- 介绍如何搭建RDS for PostgreSQL实例跨区域容灾。

前提条件

- 拥有华为云实名认证账号。
- 账户余额大于等于0美元。

约束条件

- 主实例和灾备实例状态正常，主实例和灾备实例在不同云或不同区域上，且主实例为主备实例，灾备实例为单机实例。
- 主实例配置容灾能力成功后才能配置灾备实例容灾能力，否则容灾关系会建立失败。
- 灾备实例的规格要大于等于主实例的规格。
- RDS for PostgreSQL 12及以上支持建立跨云或跨区域容灾关系。
- 灾备实例的底层架构和数据库大版本要与主实例一致。
- 如果灾备实例与主实例的小版本不一致，则在灾备搭建完成后，会自动重置灾备实例小版本，并与主实例的小版本保持一致。
- 不支持跨大版本建立跨云或跨区域容灾关系。
- 主实例和灾备实例的容灾关系已建立完成，才能进行灾备实例升主和查询容灾复制状态。

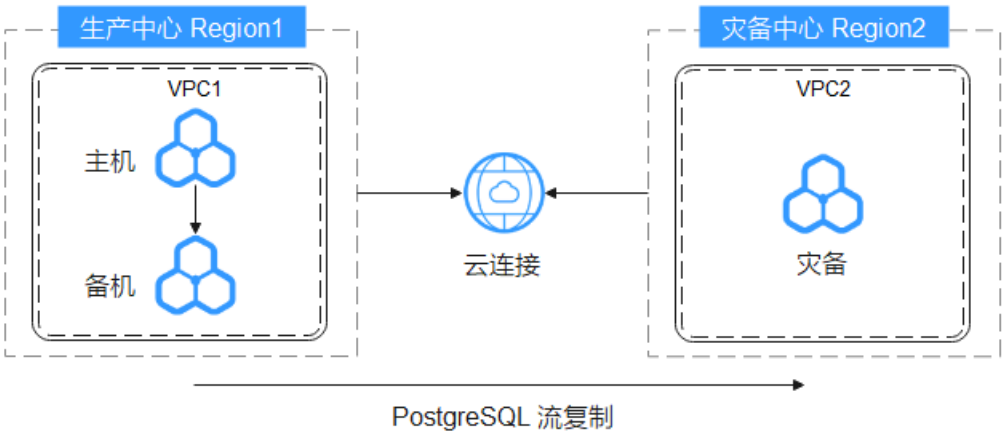
- 实施前确认需要搭建的主实例和灾备实例所在区域，处于云连接/虚拟专有网络服务已上线区域内。
- 灾备实例不支持PITR恢复和CBR快照备份功能，如需使用此功能，请在主实例上完成。

实现原理

RDS for PostgreSQL跨区域容灾实现原理说明：

在两个数据中心独立部署RDS for PostgreSQL实例，通过RDS接口将生产中心RDS for PostgreSQL库中的数据同步到灾备中心RDS for PostgreSQL库中，实现RDS for PostgreSQL主实例和跨区域灾备实例之间的实时同步。使用该功能前，必须需要确认可以使用云连接服务完成跨区域网络连通。

图 2-1 原理图



服务列表

- 云连接 CC
- 虚拟私有云 VPC
- 云数据库 RDS

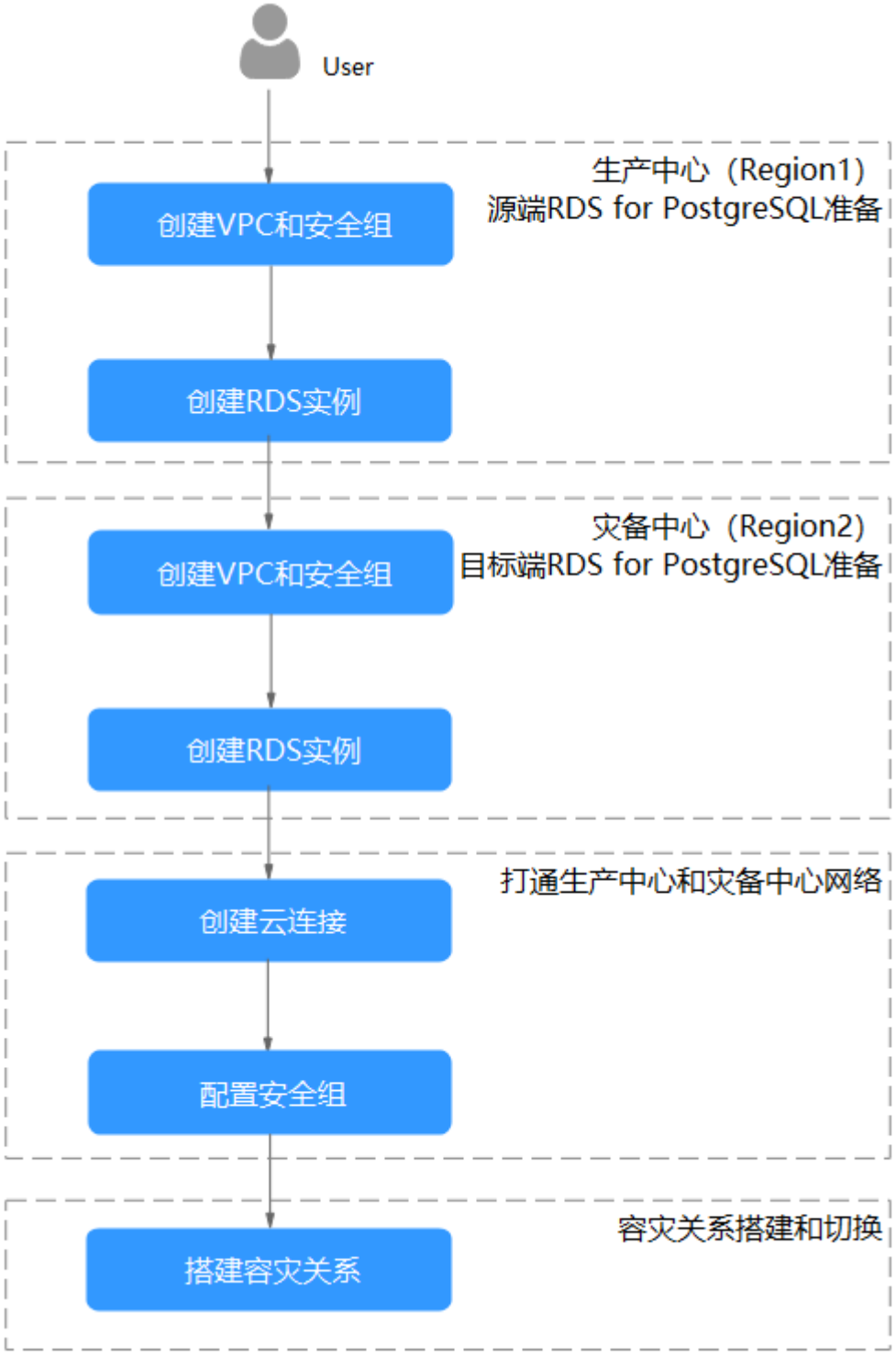
使用说明

- 本实践的资源规划仅作为演示，实际业务场景资源以用户实际需求为准。
- 本实践端到端的数据为测试数据，仅供参考。

操作流程

创建RDS for PostgreSQL业务实例以及灾备实例，并且将业务实例数据迁移到灾备实例的整个流程的主要任务流如下图所示。

图 2-2 流程图



2.2 资源规划

表 2-1 资源规划

类别	子类	规划	备注
生产中心VPC	VPC名称	vpc-pg-01	自定义，易理解可识别。
	所属Region	中国-香港	选择和自己业务区最近的Region，减少网络时延。
	可用区	可用区一	-
	子网网段	192.168.10.0/24	子网选择时建议预留足够的网络资源。
	子网名称	subnet-2aa1	自定义，易理解可识别。
灾备中心VPC	VPC名称	vpc-pg-02	自定义，易理解可识别。
	所属Region	亚太-新加坡	选择和自己业务区最近的Region，减少网络时延。
	可用区	可用区一	-
	子网网段	192.168.20.0/24	子网选择时建议预留足够的网络资源。
	子网名称	subnet-a388	自定义，易理解可识别。
生产中心RDS for PostgreSQL实例	RDS实例名称/ID	rds-pg-01 04**in03	自定义，易理解可识别。
	所属Region	中国-香港	选择和自己业务区最近的Region，减少网络时延。
	数据库版本	PostgreSQL 12	-
	内网IP	192.168.10.117	-
	实例类型	主备	生产中心实例为主备。
	存储类型	SSD云盘	-
	可用区	可用区一、可用区三	-
	性能规格	独享型 2 vCPUs | 4GB	-
	磁盘容量	40 GB	-
灾备中心RDS for PostgreSQL实例	RDS实例名称/ID	rds-pg-02 5f**in03	自定义，易理解可识别。

类别	子类	规划	备注
	所属Region	亚太-新加坡	选择和自己业务区最近的Region，减少网络时延。
	数据库版本	PostgreSQL 12	-
	内网IP	192.168.20.69	-
	实例类型	单机	灾备中心实例为单机。
	存储类型	SSD云盘	-
	可用区	可用区一	-
	性能规格	独享型 8 vCPUs | 16GB	灾备中心实例的CPU和内存规格要大于或等于主实例的规格。
	磁盘容量	100 GB	灾备中心实例的磁盘容量要大于或等于主实例的磁盘容量。

2.3 生产中心 RDS for PostgreSQL 实例准备

本章节介绍在生产中心创建RDS for PostgreSQL实例所属VPC和安全组，然后创建RDS for PostgreSQL业务实例。

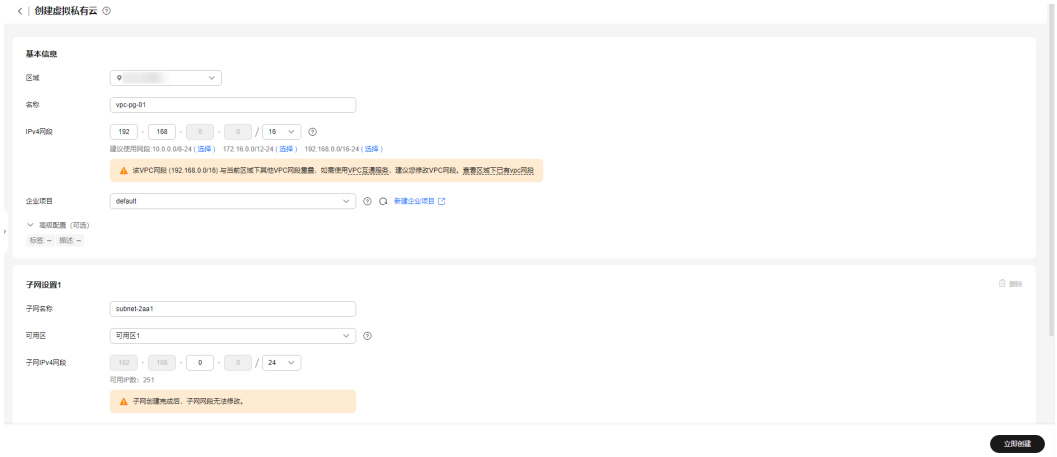
- **步骤1：创建VPC和安全组**
- **步骤2：创建RDS for PostgreSQL实例**

步骤 1：创建 VPC 和安全组

步骤1 进入**创建虚拟私有云页面**。

步骤2 在“创建虚拟私有云”页面，根据页面完成基本信息、子网配置和地址配置。选择区域“中国-香港”。

图 2-3 创建 VPC

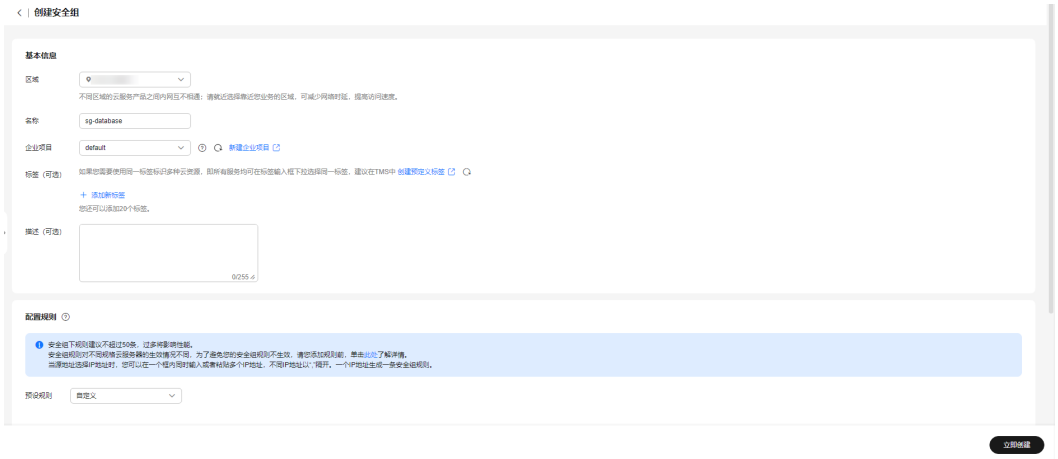


步骤3 单击“立即创建”，完成生产VPC创建。

步骤4 在网络控制台左侧导航树，选择“访问控制 > 安全组”。

步骤5 单击“创建安全组”。

图 2-4 创建安全组



步骤6 单击“立即创建”，完成生产安全组创建。

----结束

步骤 2：创建 RDS for PostgreSQL 实例

步骤1 进入**购买云数据库RDS页面**。

步骤2 选择区域“中国-香港”。填选实例信息后，单击“立即购买”。

图 2-5 选择引擎版本信息

图 2-6 选择规格信息

图 2-7 选择已规划的网络信息

图 2-8 设置管理员密码

步骤3 进行规格确认。

- 如果需要重新选择实例规格，单击“上一步”，回到上个页面修改实例信息。
- 如果规格确认无误，单击“提交”，完成购买实例的申请。

----结束

2.4 灾备中心 RDS for PostgreSQL 实例准备

本章节介绍在灾备中心创建RDS for PostgreSQL实例所属VPC和安全组，然后创建RDS for PostgreSQL业务实例。

须知

- 灾备中心实例的VPC子网网段需与生产中心VPC子网网段不同，这是实现跨区域网络连接的前提。
- 生产中心和灾备中心的安全组需要相互放通实例所属VPC子网网段的数据库端口。

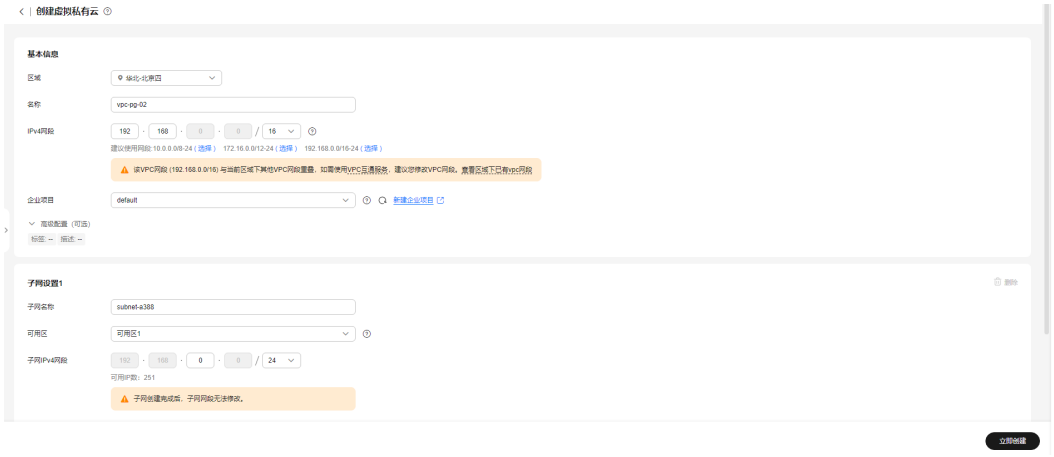
- **步骤1：创建VPC和安全组**
- **步骤2：创建RDS for PostgreSQL实例**

步骤 1：创建 VPC 和安全组

步骤1 进入**创建虚拟私有云页面**。

步骤2 在“创建虚拟私有云”页面，根据页面完成基本信息、子网配置和地址配置。选择区域“亚太-新加坡”。

图 2-9 创建 VPC



步骤3 单击“立即创建”，完成生产VPC创建。

步骤4 在网络控制台左侧导航树，选择“访问控制 > 安全组”。

步骤5 单击“创建安全组”。

图 2-10 创建安全组

步骤6 单击“立即创建”，完成生产安全组创建。

----结束

步骤 2：创建 RDS for PostgreSQL 实例

步骤1 进入**购买云数据库RDS页面**。

步骤2 选择区域“亚太-新加坡”。填选实例信息后，单击“立即购买”。

图 2-11 选择引擎版本信息

图 2-12 选择规格信息

图 2-13 选择已规划的网络信息

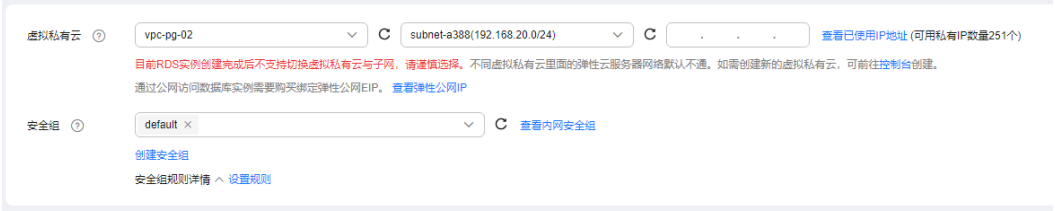


图 2-14 设置管理员密码

步骤3 进行规格确认。

- 如果需要重新选择实例规格，单击“上一步”，回到上个页面修改实例信息。
- 如果规格确认无误，单击“提交”，完成购买实例的申请。

----结束

2.5 配置跨区域网络互通

搭建异地容灾实例，首先需要完成跨区域网络的连通。当前提供**方式一：通过云连接配置跨区域VPC互通**和**方式二：通过虚拟专用网络配置跨区域VPC互通**两种方式实现跨区域网络连通。

在选择带宽大小时，建议根据事务日志生成速率监控指标进行选择，大于等于该指标最高值的10倍即可，因为网络带宽单位为Mbit/s，而事务日志生成速率监控指标的单位为MB/s。

例如事务日志生成速率监控指标最高值为10MB/s，则网络带宽建议选择100Mbit/s，以便于灾备节点有足够的带宽可以及时同步主节点的数据。

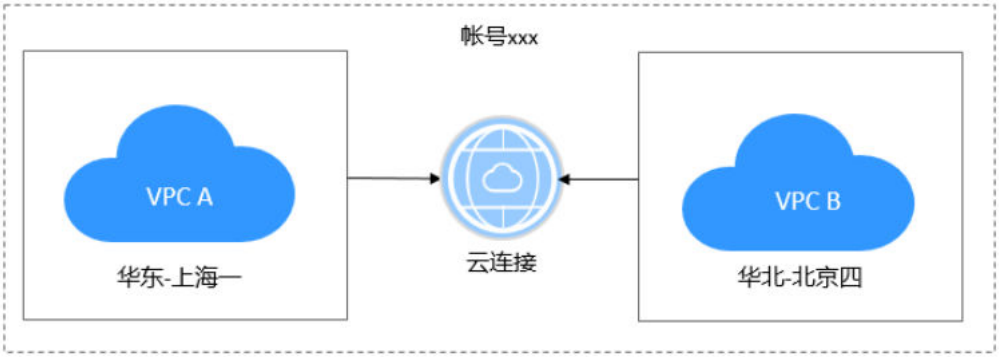
在网络打通后，需要配置主实例和容灾实例互相放通安全组，详见**配置安全组**。

方式一：通过云连接配置跨区域 VPC 互通

搭建异地容灾实例，首先需要完成跨区域网络的连通。

可以使用**云连接 CC**产品完成跨区域VPC网络连通。

图 2-15 跨区域同账号互通



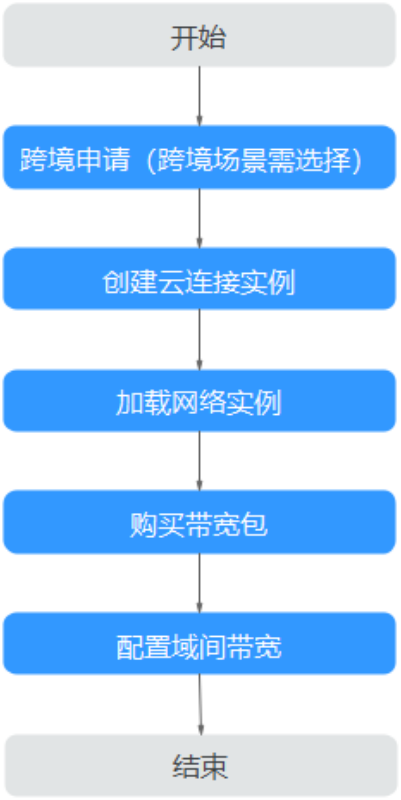
须知

配置前请确认搭建的主实例和灾备实例在云连接**已上线区域清单**列表上。

配置云连接两端需放通主实例和灾备实例所属VPC子网的网段，仅放通单个数据库IP会搭建失败。

跨区域VPC网络连通具体操作步骤可以按照**跨区域同账号VPC互通**进行配置。

图 2-16 云连接流程图



方式二：通过虚拟专用网络配置跨区域 VPC 互通

可以使用**虚拟专用网络 VPN**服务完成跨区域VPC网络连通。

须知

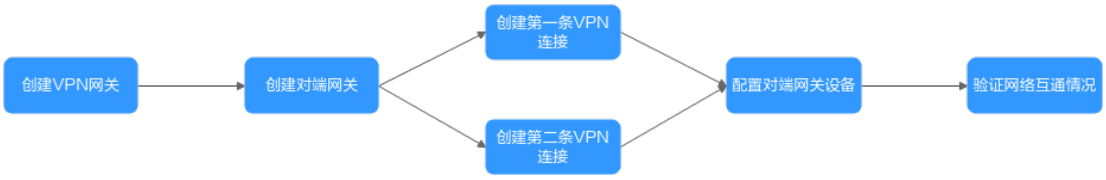
配置前请确认搭建的主实例和灾备实例在虚拟专用网络**功能支持区域**列表上。

当前在使用虚拟专用网络连通时，需要在配置完VPN服务后，联系VPN服务客服，进行网络配置调配，才能使用。

配置跨VPC两端需放通主实例和灾备实例所属VPC子网的网段，仅放通单个数据库IP会搭建失败。

虚拟专用网络的具体操作步骤可以参考**虚拟专用网络入门指引**进行配置。

图 2-17 操作流程



配置安全组

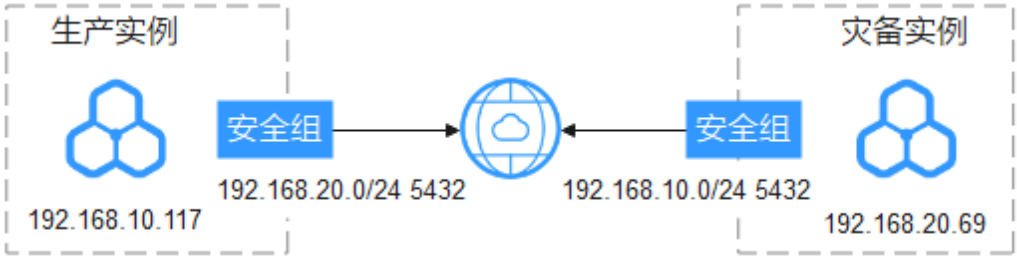
完成跨区域VPC互通后，还需要配置主实例和灾备实例的安全组，使得VPC网段间的端口互通。

假设存在表2-2所示的实例配置，其数据库端口都为默认值5432。则生产实例和灾备实例的的防火墙配置如图2-18所示。

表 2-2 实例网段

类别	VPC网段	IP地址
生产实例	192.168.10.0/24	192.168.10.117
灾备实例	192.168.20.0/24	192.168.20.69

图 2-18 防火墙配置



2.6 搭建容灾关系

操作场景

建立跨区域容灾关系后，当主实例所在区域发生突发性自然灾害等状况，主节点无法连接，可将异地灾备实例升为主实例，在应用端修改数据库连接地址后，即可快速恢复应用的业务访问。

注意事项

- 使用该功能前，必须要确保跨区域数据库实例之间的网络打通，可以使用云连接CC或虚拟专用网络 VPN产品完成跨区域VPC网络连通。
- 使用该功能前，确保主实例和灾备实例状态正常，主实例和灾备实例在不同区域上，且主实例为主备实例，灾备实例为单机实例。
- 灾备实例的CPU和内存规格以及磁盘容量要大于或等于主实例的规格以及磁盘容量。
- 灾备实例的底层架构和数据库大版本要与主实例一致。
- 不支持跨大版本建立跨云或跨Region容灾关系。
- 调用配置主实例容灾接口后直至成功搭建容灾关系，不能进行规格变更、主备倒换操作。
- 搭建容灾后，灾备实例支持变更CPU和内存规格，如需使用该功能，请提交工单申请。

- 修改主实例的端口或内网地址后需要重新搭建灾备关系。
- 灾备实例搭建成功后，不能进行小版本升级。
- RDS for PostgreSQL 12及以上支持建立跨区域容灾关系。
- 主实例参数被修改后，灾备实例无法同步修改该参数，需结合业务自行修改灾备实例参数。
- RDS for PostgreSQL灾备实例不支持PITR恢复和CBR快照备份功能，如需使用此功能，请在主实例上完成。

操作步骤

步骤1 配置生产实例容灾能力，即将灾备实例的配置信息复制到生产实例上。

1. 登录管理控制台。
2. 单击管理控制台左上角的，选择灾备实例所在的区域，例如“亚太-新加坡”。
3. 单击页面左上角的，选择“数据库>云数据库 RDS”，进入RDS信息页面。
4. 在“实例管理”页面，单击灾备实例的实例名称进入“概览”页面。
5. 单击“灾备配置信息”。
6. 在弹框中，单击“一键复制”。

图 2-19 复制灾备实例配置信息

7. 单击管理控制台左上角的，选择生产实例所在的区域，例如“中国-香港”。
8. 在“实例管理”页面，选择生产实例，单击“操作”列的“更多 > 查看容灾详情”，进入“容灾管理”页面。

9. 单击“搭建容灾”，在弹框中，将**步骤1.6**中复制的灾备配置信息粘贴至输入框中，单击“确定”，开始配置生产实例容灾能力。

图 2-20 粘贴灾备实例信息

10. 在生产实例的“容灾管理”页面查看搭建情况。当“搭建状态” 显示为“已搭建”，则表示配置生产实例容灾能力成功。确保此步骤搭建成功后再进行后续操作。

图 2-21 查看搭建容灾成功

步骤2 配置灾备实例容灾能力，即将生产实例的配置信息复制到灾备实例上。

1. 在“实例管理”页面，单击生产实例的实例名称进入“概览”页面。
2. 单击“灾备配置信息”。
3. 在弹框中，单击“一键复制”。
4. 单击管理控制台左上角的，选择灾备实例所在的区域，例如“亚太-新加坡”。
5. 在“实例管理”页面，选择灾备实例，单击“操作”列的“更多 > 查看容灾详情”，进入“容灾管理”页面。
6. 单击“搭建容灾”，在弹框中，将**步骤2.3**中复制的灾备配置信息粘贴至输入框中，单击“确定”，开始配置灾备实例容灾能力。

7. 在“容灾管理”页面查看搭建情况。当“搭建状态” 显示为“已搭建”，则表示配置灾备实例容灾能力成功，至此，容灾搭建成功。
8. 在“容灾管理”页面可以查看灾备复制状态、发送延迟大小、端到端延迟大小和回放延迟时间。

----结束

2.7 灾备升主

操作场景

当主实例所在区域发生突发性自然灾害等状况，主节点无法连接，可将异地灾备实例升为主实例，在应用端修改数据库连接地址后，即可快速恢复应用的业务访问。

注意事项

灾备升主后的新主实例和原主实例将会解除灾备关系。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择灾备实例所在的区域，例如“亚太-新加坡”。

步骤3 单击页面左上角的，选择“数据库>云数据库 RDS”，进入RDS信息页面。

步骤4 单击灾备实例名称，进入实例的概览页面。

步骤5 在左侧导航栏，选择“容灾管理”。

步骤6 在容灾关系列表单击“操作”列的“灾备升主”。

步骤7 在弹框中，单击“确认”，开始下发灾备升主任务。

步骤8 可在“任务中心”中查看任务的执行结果，当任务状态为完成时，表示灾备升主任务成功。

步骤9 需用户在应用端修改数据库连接地址，手动将业务切换到新主实例。

----结束

2.8 解除灾备

操作场景

当不需要容灾关系时，可以将搭建好的容灾关系解除。

注意事项

当前只支持解除搭建成功的容灾关系，并且需要先解除灾备实例的容灾关系，再解除主实例的容灾关系，否则可能会引起异常告警。

操作步骤

步骤1 登录管理控制台。

步骤2 首先解除灾备实例的容灾关系。

1. 进入主实例的概览页面，单击“灾备配置信息”。
2. 在弹框中，单击“一键复制”。
3. 单击灾备实例名称，进入实例的概览页面。
4. 在左侧导航栏，选择“容灾管理”。
5. 在容灾关系列表单击“操作”列的“解除灾备关系”。
6. 将刚复制的灾备配置信息粘贴进弹框中。
7. 可在“容灾管理”中查看任务的执行结果，当列表被删除时，则执行成功。

步骤3 然后解除主实例的容灾关系，步骤参考**步骤2**，复制灾备实例的“灾备配置信息”，然后在主实例的“容灾管理”页面解除。

----结束

2.9 常见问题

- 问题1：配置灾备实例容灾能力时，任务执行失败。

  排查生产中心和灾备中心的安全组是否互相放通数据库VPC子网网段的端口。如果使用VPN连接，确认是否按照**方式二：通过虚拟专用网络配置跨区域VPC互通**的注意事项配置。

- 问题2：使用云连接配置VPC时，系统提示“当前VPC存在路由冲突，此VPC路由已存在”。

  参考**加载网络实例时出现系统异常怎么办**进行排查处理。

3 RDS for PostgreSQL 发布与订阅

逻辑定义

发布可以被定义在任何物理复制的主服务器上。定义有发布的节点被称为发布者。发布是从一个表或者一组表生成的改变的集合，也可以被描述为更改集合或者复制集合。每个发布都只存在于一个数据库中。

订阅是逻辑复制的下游端。订阅被定义在其中的节点被称为订阅者。一个订阅会定义到另一个数据库的连接以及它想要订阅的发布集合（一个或者多个）。逻辑订阅者的行为与一个普通的PostgreSQL实例（主库）无异，逻辑订阅者也可以创建自己的发布，拥有自己的订阅者。

使用权限

- 创建发布时，发布者必须有流复制权限。即replication权限。
- 创建发布时，使用all tables发布所有表时，需确保发布者是**提权起始及之后版本**的root用户。
- 创建/删除订阅时，需确保订阅者是**提权起始及之后版本**的root用户。
- 创建发布/订阅时，需确保发布端和订阅端实例在同一VPC下。

各个版本root用户提权情况参见**root用户权限说明**。

发布使用限制

- 发布目前只能包含表（即：索引，序列号，物化视图这些不会被发布），每个表可以添加到多个发布中。
- 一个publication允许有多个订阅者。
- 允许使用all tables发布所有表。
- 在同一个数据库中，可以创建多个publication，但是不能重名。已创建的publication可以通过查询pg_publication获取。
- 发布可以筛选所需的变更类型：包括insert、update、delete 和truncate的任意组合，类似触发器事件，默认所有变更都会被发布。

  例如：发布表t1的**update**和**delete**操作。

  ```
  CREATE PUBLICATION update_delete_only FOR TABLE t1
    WITH (publish = 'update, delete') ;
  ```

- 复制标识：当发布了表的**update**, **delete**时，表必须设置复制标识（Replica Identity），如果设置了nothing，则执行**update**, **delete**时会报错。

表上的复制标识可以通过查阅pg_class.relreplident获取。

这是一个字符类型的“枚举”，标识用于组装 “复制标识” 的列：d = default ，f = 所有的列，i 使用特定的索引，n 没有复制标识。

表上是否具有可用作复制标识的索引约束，可以通过以下查询获取：

```
SELECT quote_ident(nspname) || '.' || quote_ident(relname) AS name, con.ri AS keys,
      CASE relreplident WHEN 'd' THEN 'default' WHEN 'n' THEN 'nothing' WHEN 'f' THEN
'full' WHEN 'i' THEN 'index' END AS replica_identity
FROM pg_class c JOIN pg_namespace n ON c.relnamespace = n.oid, LATERAL (SELECT
array_agg(contype) AS ri FROM pg_constraint WHERE conrelid = c.oid) con
WHERE relkind = 'r' AND nspname NOT IN ('pg_catalog', 'information_schema', 'monitor',
'repack', 'pg_toast')
ORDER BY 2,3;
```

- 复制标识配置

  表到复制标识可以通过**ALTER TABLE**进行修改。

  ```
  ALTER TABLE table_name REPLICA IDENTITY
  { DEFAULT | USING INDEX index_name | FULL | NOTHING };
  -- 具体有四种形式
  ALTER TABLE t_normal REPLICA IDENTITY DEFAULT;                    -- 使用主键，如果没有主键则为FULL
  ALTER TABLE t_normal REPLICA IDENTITY FULL;                       -- 使用整行作为标识
  ALTER TABLE t_normal REPLICA IDENTITY USING INDEX t_normal_v_key; -- 使用唯一索引
  ALTER TABLE t_normal REPLICA IDENTITY NOTHING;                    -- 不设置复制标识
  ```

- 复制标识在实际使用中的注意事项
  - 表上有主键，使用默认的default复制标识。
  - 表上没有主键，但是有非空唯一索引，显式配置index复制标识。
  - 表上既没有主键，也没有非空唯一索引，显式配置full复制标识（运行效率非常低，仅能作为兜底方案）。
  - 其他所有情况，都无法正常完成逻辑复制功能。输出的信息不足，可能会报错。
  - 特别需要注意：如果nothing复制标识的表纳入到逻辑复制中，对其进行删改会导致发布端报错。

订阅使用限制

- 为了确保使用Failover Slot，必须在发布端手工创建逻辑复制槽(Failover Slot)，并通过**create_slot = false**关联已有复制槽，如下：

  CREATE SUBSCRIPTION sub1 CONNECTION 'host=192.168.0.1 port=5432 user=user1 dbname=db1' PUBLICATION pub_name with (create_slot = false,slot_name = FailoverSlot_name);

- 逻辑复制不会复制DDL变更，因此发布集中的表必须已经存在于订阅端上。
- 同一个数据库中，可以创建多个subscription，这些subscription可以来自一个或多个发布者。
- 订阅者的同一张表，不能接受来自同一个源的多个发布。
- 在创建subscription或者alter subscription时，可以使用enable来启用该订阅，或者使用disable暂停该订阅。
- 如果要完全删除订阅，使用**DROP SUBSCRIPTION**，注意，删除订阅后，本地的表不会被删除，数据也不会清除，仅仅是不再接收该订阅的上游信息。

须知

如果订阅与复制槽相关联，就不能在事务块内部执行**DROP SUBSCRIPTION**。可以使用**ALTER SUBSCRIPTION**取消关联复制槽。

删除订阅可参考以下步骤：

a. 在订阅端查询订阅关联的的复制槽。

select subname,subconninfo,subslotname from pg_subscription where subname = 'sub2';

- subname为订阅者名称。
- subconninfo为连接远程主机信息。
- subslotname 为远程主机复制槽名称。

b. 在订阅端执行**ALTER SUBSCRIPTION**取消关联复制槽并删除。

ALTER SUBSCRIPTION subname SET (slot_name = NONE);

DROP SUBSCRIPTION subname;

c. 在发布端删除关联的复制槽。

select pg_drop_replication_slot(' slot_name);

语法参考

- 发布

  CREATE PUBLICATION用于创建发布，**DROP PUBLICATION**用于移除发布，**ALTER PUBLICATION**用于修改发布。

  发布创建之后，可以通过**ALTER PUBLICATION**动态地向发布中添加或移除表，这些操作都是事务性的。

- 订阅

  CREATE SUBSCRIPTION用于创建订阅，**DROP SUBSCRIPTION**用于移除订阅，**ALTER SUBSCRIPTION**用于修改订阅。

  订阅创建之后，可以通过**ALTER SUBSCRIPTION**随时**暂停**与**恢复**订阅。移除并重建订阅会导致**同步信息丢失**，这意味着相关数据需要重新进行同步。

具体使用说明请参考以下官方文档，以PostgreSQL 13版本为例：

- 创建发布：**https://www.postgresql.org/docs/13/sql-createpublication.html**
- 删除发布：**https://www.postgresql.org/docs/13/sql-droppublication.html**
- 修改发布：**https://www.postgresql.org/docs/13/sql-alterpublication.html**

4 RDS for PostgreSQL 自定义数据类型转换

简介

PostgreSQL数据类型有三种转换方式：隐式转换，赋值转换，显式转换。对应的转换类型在系统表“pg_cast”中分别对应：i（Implicit）、a（Assignment）、e（Explicit）。

- 隐式转换（Implicit）：同一类型间，低字节到高字节为隐式转换，比如int到bigint。
- 赋值转换（Assignment）：同一类型间，高字节到低字节为赋值转换，比如smallint到int。
- 显式转换（Explicit）：不同类型间，称为显示转换。

基本使用

1. 在进行数据类型转换前，可以通过如下命令查看RDS for PostgreSQL是否已经支持数据类型转换。

```
select * from pg_catalog.pg_cast ;
oid  | castsource | casttarget | castfunc | castcontext | castmethod
-------+------------+------------+----------+-------------+------------
 11277 |         20 |         21 |      714 | a           | f
 11278 |         20 |         23 |      480 | a           | f
 11279 |         20 |        700 |      652 | i           | f
 11280 |         20 |        701 |      482 | i           | f
.....
```

2. 通过如下命令查询**int4**是否支持转换**text**。

```
select * from pg_catalog.pg_cast where castsource = 'int4'::regtype and casttarget = 'bool'::regtype;
  oid  | castsource | casttarget | castfunc | castcontext | castmethod
-------+------------+------------+----------+-------------+------------
 11311 |         23 |         16 |     2557 | e           | f
(1 row)
```

此时查出结果是默认支持的，转换类型是隐式转换。

如果没有内置的转换函数，需要自定义转换函数来支持这种转换，具体参考**自定义类型转换**。

自定义类型转换

- 通过双冒号方式进行强制转换

```
select '10'::int,'2023-10-05'::date;
 int4 |    date
------+------------
```

```
  10 | 2023-10-05
(1 row)
```

- 通过类型转换函数**CAST**进行转换

```
select CAST('10' as int),CAST('2023-10-05' as date);
 int4 |    date
------+------------
  10 | 2023-10-05
(1 row)
```

- 自定义类型转换

  具体语法及使用可查看：**https://www.postgresql.org/docs/14/sql-createcast.html**

须知

由于新增自定义类型转换会影响RDS for PostgreSQL已有的执行计划，一般不建议自定义类型转换。

  - 时间与字符类型的转换

    CREATE CAST(varchar as date) WITH INOUT AS IMPLICIT;
  - boolean类型与数值类型转换

    create cast(boolean as numeric) with INOUT AS IMPLICIT;
  - 数值类型与字符类型转换

    create cast(varchar as numeric) with INOUT AS IMPLICIT;

  示例：将**text**转换为**date**

```
create or replace function public.text_to_date(text) returns date as
$$
  select to_date($1,'yyyy-mm-dd');
$$
language sql strict;

create cast (text as date) with function public.text_to_date(text) as implicit;

select text '2023-09-09' + 1;
  ?column?
------------
 2023-09-10
(1 row)
```

5 使用客户端驱动程序实现故障转移和读写分离

从PostgreSQL 10（libpq.so.5.10）开始，libpq驱动层开始支持故障转移和读写分离，JDBC驱动层则支持读写分离、故障转移和负载均衡。

PostgreSQL客户端连接程序向下兼容，对于RDS for PostgreSQL 9.5及9.6版本，使用新版本的libpq驱动程序也可以实现故障转移。

说明

本章节中故障转移指的是读业务的故障转移。

- libpq是PostgreSQL的C应用程序接口，包含一组库函数，允许客户端程序将查询请求发送给PostgreSQL后端服务器并接收这些查询的结果。
- JDBC是Java语言中用来规范客户端程序如何访问数据库的应用程序接口，在PostgreSQL中JDBC支持故障转移和负载均衡。

表 5-1 libpq 和 JDBC 驱动支持的功能

驱动	读写分离	负载均衡	故障转移
libpq驱动	√	×	√
JDBC驱动	√	√	√

libpq 实现故障转移和读写分离

通过libpq函数连接多个数据库，当出现故障时会自动切换到可用的数据库。

postgresql://[user[:password]@][netloc][:port][,...][/dbname][?param1=value1&...]

示例：连接1个RDS for PostgreSQL主实例数据库和对应的2个只读实例数据库，只要确保至少有一个数据库可用，读请求就不会失败。

postgres://
<instance_ip>:*<instance_port>*,*<instance_ip>*:*<instance_port>*,*<instance_ip>*:*<instance_port>*/*<database_name>*?**target_session_attrs=any**

表 5-2 参数说明

参数	说明	取值样例
<instance_ip>	数据库的主机IP。	如果通过内网连接，“instance_ip”是主机IP，即“概览”页面该实例的“内网地址”。 如果通过连接了公网的设备访问，“instance_ip”为该实例已绑定的“弹性公网IP”。
<instance_port>	数据库端口。	默认5432，当前端口，参考“概览”页面该实例的“数据库端口”。
<database_name>	数据库名，即需要连接的数据库名。	默认的管理数据库是postgres，可根据业务实际情况填写数据库名。
target_session_attrs	允许连接到指定状态的数据库。	• any：默认值，表示允许连接到任意数据库，会连接到第一个允许连接的数据库，如果连接的数据库出现故障导致连接断开，会尝试连接其他数据库，从而实现故障转移。 • read-write：只会连接到支持读写的数据库，即从第一个数据库开始尝试连接，如果连接后发现不支持读写，则会断开连接，然后尝试连接第二个数据库，以此类推，直至连接到支持读写的数据库。 • read-only：只会连接只读数据库，即从第一个数据库开始尝试连接，如果连接后发现支持读写，则会断开连接，然后尝试连接第二个数据库，以此类推，直至连接到只读的数据库。RDS for PostgreSQL 13（libpq.so.5.13）及以下版本，不支持该取值。

更多libpq的使用方法和参数说明请参见**Connection Strings**。

您还可以在应用程序中结合pg_is_in_recovery()函数，判断连接的数据库是主实例数据库（结果为“f”表示主数据库）还是只读实例数据库，进而实现读写分离。

使用Python代码的示例如下（psycopg2使用的为libpq）：

```
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。

import psycopg2
import os

username = os.getenv("EXAMPLE_USERNAME_ENV")
password = os.getenv("EXAMPLE_PASSWORD_ENV")
conn = psycopg2.connect(database=<database_name>,host=<instance_ip>, user=username,
password=password, port=<instance_port>, target_session_attrs="read-write")
cur = conn.cursor()
cur.execute("select pg_is_in_recovery()")
row = cur.fetchone()
print("recovery =", row[0])
```

JDBC 实现故障转移和读写分离

您可以在连接URL中定义多个数据库（主机和端口），并用逗号分隔，驱动程序将尝试按顺序连接到它们中的每一个，直到连接成功。如果没有成功，会返回连接异常报错。

jdbc:postgresql://node1,node2,node3/${database}?targetServerType=preferSecondary&loadBalanceHosts=true

示例：

jdbc:postgresql://*<instance_ip>*:*<instance_port>*,*<instance_ip>*:*<instance_port>*,*<instance_ip>*:*<instance_port>*/*<database_name>*?**targetServerType=preferSecondary&loadBalanceHosts=true**

JDBC连接实例java实现代码请参考：**通过JDBC连接RDS for PostgreSQL实例**。

表 5-3 参数说明

参数	说明	取值样例
targetServerType	允许连接到指定状态的数据库。	• any：任何数据库。 • primary：主数据库（可写可读）。JDBC 42.2.0以下版本请使用参数值“master”。 • secondary：从数据库（可读）。JDBC 42.2.0以下版本请使用参数值“slave”。 • preferSecondary：优先从数据库，如果没有从数据库才连接到主数据库。JDBC 42.2.0以下版本请使用参数值“preferSlave”。
loadBalanceHosts	尝试连接数据库的顺序。	• False：默认值，按URL中的定义顺序连接数据库。 • True：随机连接数据库。

说明

区别数据库主从的方式是通过查询数据库是否允许写入，允许写入数据的判断为主数据库，不允许写入数据的判断为从数据库。参考**libpq实现故障转移和读写分离**中通过pg_is_in_recovery()函数来判断，结果为“f”表示为主数据库。

为实现读写分离，需要在配置JDBC时设置2个数据源，首先设置targetServerType=primary，用于写操作。另一个可以根据以下情况进行设置：

- 有一个只读实例，为实现高可用设置targetServerType=preferSecondary，用于读操作。假设主实例IP为10.1.1.1，只读实例IP为10.1.1.2。

  jdbc:postgresql://10.1.1.2:5432,10.1.1.1:5432/${database}?targetServerType=preferSecondary

- 有两个以上只读实例，可设置targetServerType=any，用于读操作。假设只读实例IP分别为10.1.1.2、10.1.1.3。

jdbc:postgresql://10.1.1.2:5432,10.1.1.3:5432/${database}?targetServerType=any&loadBalanceHosts=true

6 使用 pg_waldump 解析 RDS for PostgreSQL 的 wal 日志

RDS for PostgreSQL实例购买完成后，可以先登录到Linux弹性云服务器，在ECS上安装PostgreSQL客户端，然后通过开源pg_waldump解析RDS for PostgreSQL的wal日志。

步骤 1：购买 ECS

1. **登录管理控制台**，查看是否有弹性云服务器。
   - 有Linux弹性云服务器，执行**步骤2：安装PostgreSQL客户端**。
   - 无Linux弹性云服务器，执行**2**。
2. 购买弹性云服务器时，选择Linux操作系统，例如CentOS。

   由于需要在ECS下载PostgreSQL客户端，因此需要为ECS绑定弹性公网IP（EIP）。

   购买Linux弹性云服务器请参考《弹性云服务器用户指南》中“**购买弹性云服务器**”章节。

步骤 2：安装 PostgreSQL 客户端

PostgreSQL客户端版本一定要和RDS for PostgreSQL实例版本保持一致，否则可能导致wal日志解析失败。

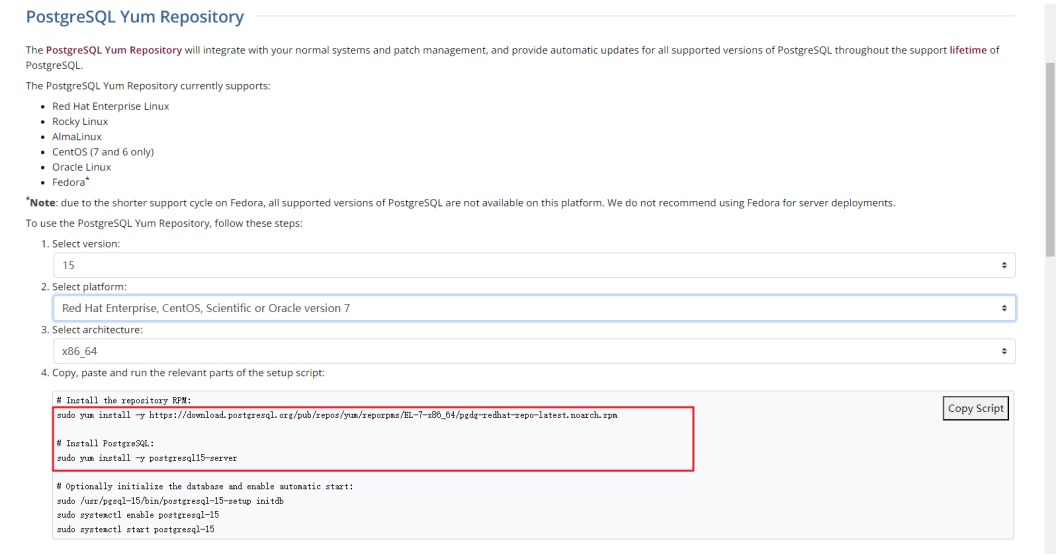
安装 PostgreSQL 客户端（15 及以下版本）

1. 登录ECS实例，请参见《弹性云服务器用户指南》中“**Linux弹性云服务器远程登录（VNC方式）**”。
2. 安装PostgreSQL客户端。

   PostgreSQL社区提供了针对不同操作系统的**客户端安装方法**。通过操作系统的安装工具直接下载安装。此安装方式比较简单，但是对ECS操作系统有要求，只有PostgreSQL社区中支持的操作系统才可以使用该安装方式。

   使用操作系统默认安装，当前使用的Linux操作系统是CentOS 7，通过工具安装最高版本是15版本。

图 6-1 获取安装工具



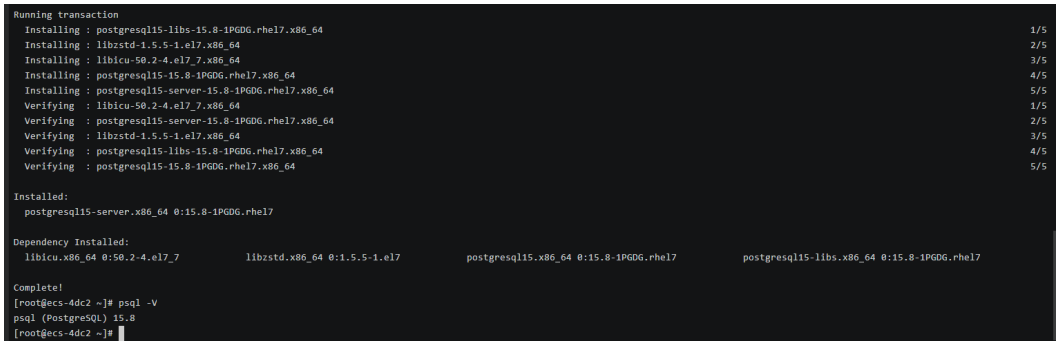
执行**安装**命令：

```
sudo yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
sudo yum install -y postgresql15-server
```

查看是否安装成功：

```
psql -V
```

图 6-2 安装成功



安装 PostgreSQL 客户端（对版本没有限制）

1. 登录ECS实例，请参见《弹性云服务器用户指南》中“**Linux弹性云服务器远程登录（VNC方式）**”。
2. 安装PostgreSQL客户端。

   源码安装方式，该安装方式对RDS for PostgreSQL实例的版本以及ECS的操作系统没有限制。

   下面以Huawei Cloud EulerOS 2.0镜像的ECS为例，安装PostgreSQL 16.4版本客户端。

图 6-3 查看 ECS 镜像

a. 要支持SSL，需要在ECS上提前下载openssl。
```
sudo yum install -y openssl-devel
```

b. 在官网获取代码下载链接，使用wget直接下载安装包或者下载到本地后上传到ECS上。
```
wget https://ftp.postgresql.org/pub/source/v16.4/postgresql-16.4.tar.gz
```

c. 解压安装包。
```
tar xf postgresql-16.4.tar.gz
```

d. 编译安装。
```
cd postgresql-16.4
./configure --without-icu --without-readline --without-zlib --with-openssl
make -j 8 && make install
```

说明

不指定--prefix，表示默认路径为“/usr/local/pgsql”，因为只安装客户端采用最简安装。

图 6-4 编译安装

```
make -C ../../../src/common all
make[4]: Entering directory '/root/postgresql-16.4/src/common'
make[4]: Nothing to be done for 'all'.
make[4]: Leaving directory '/root/postgresql-16.4/src/common'
make[3]: Leaving directory '/root/postgresql-16.4/src/interfaces/libpq'
make -C ../../../src/port all
make[3]: Entering directory '/root/postgresql-16.4/src/port'
make[3]: Nothing to be done for 'all'.
make[3]: Leaving directory '/root/postgresql-16.4/src/port'
make -C ../../../src/common all
make[3]: Entering directory '/root/postgresql-16.4/src/common'
make[3]: Nothing to be done for 'all'.
make[3]: Leaving directory '/root/postgresql-16.4/src/common'
/usr/bin/mkdir -p '/usr/local/pgsql/lib/pgxs/src/test/isolation'
/usr/bin/install -c  pg_isolation_regress '/usr/local/pgsql/lib/pgxs/src/test/isolation/pg_isolation_regress'
/usr/bin/install -c  isolationtester '/usr/local/pgsql/lib/pgxs/src/test/isolation/isolationtester'
make[2]: Leaving directory '/root/postgresql-16.4/src/test/isolation'
make -C test/perl install
make[2]: Entering directory '/root/postgresql-16.4/src/test/perl'
make[2]: Nothing to be done for 'install'.
make[2]: Leaving directory '/root/postgresql-16.4/src/test/perl'
/usr/bin/mkdir -p '/usr/local/pgsql/lib/pgxs/src'
/usr/bin/install -c -m 644 Makefile.global '/usr/local/pgsql/lib/pgxs/src/Makefile.global'
/usr/bin/install -c -m 644 Makefile.port '/usr/local/pgsql/lib/pgxs/src/Makefile.port'
/usr/bin/install -c -m 644 ./Makefile.shlib '/usr/local/pgsql/lib/pgxs/src/Makefile.shlib'
/usr/bin/install -c -m 644 ./nls-global.mk '/usr/local/pgsql/lib/pgxs/src/nls-global.mk'
make[1]: Leaving directory '/root/postgresql-16.4/src'
make -C config install
make[1]: Entering directory '/root/postgresql-16.4/config'
/usr/bin/mkdir -p '/usr/local/pgsql/lib/pgxs/config'
/usr/bin/install -c -m 755 ./install-sh '/usr/local/pgsql/lib/pgxs/config/install-sh'
/usr/bin/install -c -m 755 ./missing '/usr/local/pgsql/lib/pgxs/config/missing'
make[1]: Leaving directory '/root/postgresql-16.4/config'
```

e. 配置环境变量，在“/etc/profile”文件中添加以下内容。

```
export PATH=/usr/local/pgsql/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:$LD_LIBRARY_PATH
source /etc/profile
```

f. 测试psql是否可使用。

```
psql -V
```

图 6-5 测试 psql 可用

```
                . /etc/bashrc
        fi
fi
export PATH=/usr/local/pgsql/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:$LD_LIBRARY_PATH
[root@ecs-88a7 pgsql]# source /etc/profile
[root@ecs-88a7 pgsql]# psql -V
psql (PostgreSQL) 16.4
[root@ecs-88a7 pgsql]#
```

步骤 3：获取 RDS for PostgreSQL 实例中的增量 wal 日志

1. **下载增量备份文件**，假设将下载的增备文件存放在/download/backup。
2. 使用提供的**脚本**，解压后执行如下命令，将增量备份文件解压到一个临时目录（例如/tmp/wal）下。

```
python restore_wal.py /download/backup /tmp/wal
```

   - /download/backup：PostgreSQL增量备份文件的目录。
   - /tmp/wal：临时存放pg_wal日志的目录。

步骤 4：使用 pg_waldump 解析指定 wal 日志

示例：如下命令显示wal日志000000010000000000000003中前5行记录内容。

```
pg_waldump -n 5 000000010000000000000003
```

回显如下：

```
rmgr: Standby     len (rec/tot):     50/     50, tx:          0, lsn: 0/03000028, prev 0/02000100, desc: RUNNING_XACTS nextXid 765 latestCompletedXid 764 oldestRunningXid 765
rmgr: XLOG        len (rec/tot):    114/    114, tx:          0, lsn: 0/03000060, prev 0/03000028, desc: CHECKPOINT_SHUTDOWN redo 0/3000060; tli 1; prev tli 1; fpw true; xid 0:765; oid 16393; multi 1; offset 0; oldest xid 730 in DB 1; oldest multi 1 in DB 1; oldest/newest commit timestamp xid: 0/0; oldest running xid 0; shutdown
rmgr: XLOG        len (rec/tot):     54/     54, tx:          0, lsn: 0/030000D8, prev 0/03000060, desc: PARAMETER_CHANGE max_connections=200 max_worker_processes=8 max_wal_senders=10 max_prepared_xacts=10 max_locks_per_xact=64 wal_level=logical wal_log_hints=off track_commit_timestamp=off
rmgr: Standby     len (rec/tot):     50/     50, tx:          0, lsn: 0/03000110, prev 0/030000D8, desc: RUNNING_XACTS nextXid 765 latestCompletedXid 764 oldestRunningXid 765
rmgr: Standby     len (rec/tot):     50/     50, tx:          0, lsn: 0/03000148, prev 0/03000110, desc: RUNNING_XACTS nextXid 765 latestCompletedXid 764 oldestRunningXid 765
```

更多pg_waldump的详细操作，请参考**社区文档**。

常见问题

问：执行命令时出现如下报错怎么办?

```
pg_waldump: error: could not find a valid record after xxx/xxxxxxxx
```

答：执行**pg_waldump -V**命令查看pg_waldump的版本，查看pg_waldump的大版本是否与RDS for PostgreSQL实例的大版本一致。

如果版本不一致，重新执行**步骤2：安装PostgreSQL客户端**，安装正确的版本。

7 PoWA 插件使用最佳实践

7.1 插件介绍

PoWA是一套开源的用于对RDS for PostgreSQL数据库进行性能监控的系统。它由PoWA-archivist、PoWA-collector、PoWA-web三个部件组成工作系统，并通过安装于RDS for PostgreSQL数据库中的其他插件获取性能数据。重要组成如下所示：

- PoWA-archivist：作为PostgreSQL的一个插件，用于收集其他插件获取到的性能数据。
- PoWA-collector：是通过部署于远程服务器上的专用于存储库收集目标PostgreSQL数据库性能指标的守护进程。
- PoWA-web：通过Web图形用户界面展示PoWA-collector收集到的性能指标。
- 其他插件：性能指标数据的实际来源，安装于目标PostgreSQL数据库实例上。
- PoWA：整个系统的总称。

安全风险提醒

PoWA在进行部署配置中，会存在以下几个安全风险点：

- （远程模式时）在powa-repository中配置采集性能指标实例信息时，需要输入目标实例的IP、root用户账号、连接密码，并且可以通过powa_servers表查询到相关信息，其中连接密码以明文形式呈现，存在安全风险。
- 在PoWA-collector配置文件中，powa-repository的连接信息中无连接密码配置，表示powa-repository对于PoWA-collector的连接配置项必须为trust，存在安全风险。
- 在PoWA-web配置文件中，可选配置username、password对应powa-repository（远程模式）或者数据库实例（本地模式）的root用户及连接密码，且以明文形式存储，存在安全风险。

在使用PoWA插件时，请悉知以上安全风险。可以参考**PoWA官方手册**进行安全加固。

其他扩展插件

除了pg_stat_statements、btree_gist、powa为必须的插件，PoWA还支持以下几个插件作为性能指标采集的扩展：

- pg_qualstats
- pg_stat_kcache
- pg_wait_sampling
- pg_track_settings
- hypopg

每个插件都可以扩展不同的对应的性能指标。当前RDS for PostgreSQL支持的插件，请参见**支持的插件列表**。

7.2 支持的性能指标

7.2.1 数据库级性能指标

General Overview

图 7-1 General Overview

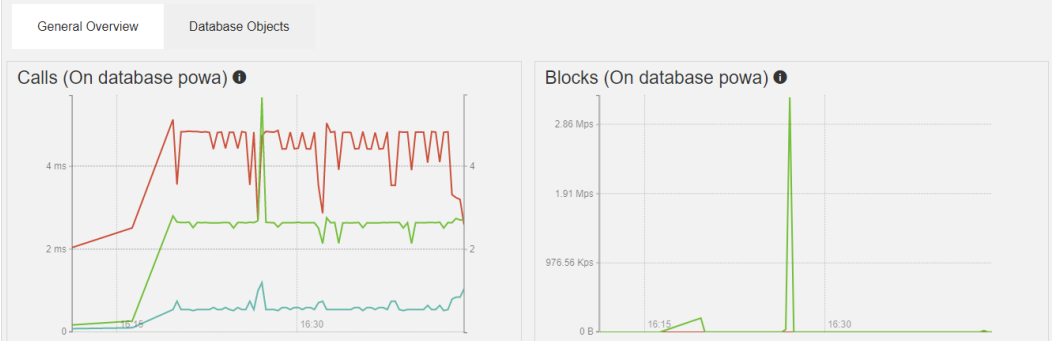


表 7-1 Calls 字段解释

字段	说明
Queries per sec	每秒执行查询的次数。
Runtime per sec	每秒内执行查询的总耗时。
Avg runtime	查询的平均耗时。

表 7-2 Blocks 字段解释

字段	说明
Total shared buffers hit	命中共享缓冲区的数据量。
Total shared buffers miss	未命中共享缓冲区的数据量。

Database Objects

图 7-2 Database Objects

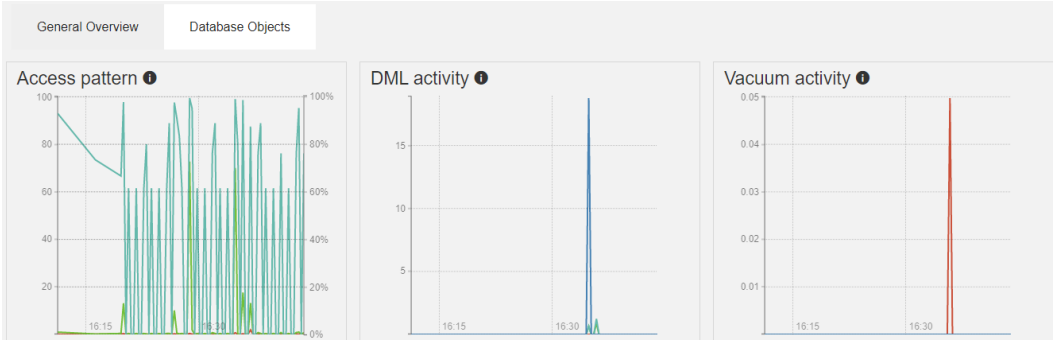


表 7-3 Access pattern 字段解释

字段	说明
Index scans ratio	索引扫描/序列扫描的比率。
Index scans	每秒索引扫描次数。
Sequential scans	每秒顺序扫描次数。

表 7-4 DML activity 字段解释

字段	说明
Tuples inserted	每秒插入的行数。
Tuples updated	每秒更新的行数。
Tuples HOT updated	每秒更新(HOT)的行数。
Tuples deleted	每秒删除的行数。

表 7-5 Vacuum activity 字段解释

字段	说明
# Vacuum	每秒手动清理的次数。
# Autovacuum	每秒自动清理的次数。
# Analyze	每秒手动分析的次数。
# Autoanalyze	每秒自动分析的次数。

Details for all databases

图 7-3 Details for all databases


Details for all queries

Export CSV

Query	Execution #	Execution Time	Execution Avg time	I/O Time Read	I/O Time Write	Blocks Read	Blocks Hit	Blocks Dirtied	Blocks Written	Temp blocks Read	Temp blocks Written
COPY (SELECT 1, * FROM public.powa_user_functions_src(0)) TO stdout	75	3 s 783 ms	50 ms 442 µs	0	0	0 B	768.00 K	0 B	0 B	0 B	0 B
COPY (SELECT 1, * FROM public.powa_all_relations_src(0)) TO stdout	75	82 ms 323 µs	1 ms 98 µs	0	0	0 B	720.00 K	0 B	0 B	0 B	0 B
COPY (SELECT 1, * FROM public.powa_statements_src(0)) TO stdout	75	66 ms 99 µs	881 µs	0	0	0 B	5.38 M	0 B	0 B	0 B	0 B
select control_extension($1, $2)	1	52 ms 437 µs	52 ms 437 µs	0	0	0 B	44.88 M	568.00 K	96.00 K	0 B	0 B
COPY (SELECT 1, * FROM public.powa_stat_bgwriter_src(0)) TO stdout	75	11 ms 886 µs	158 µs	0	0	0 B	688.00 K	0 B	0 B	0 B	0 B
COPY (SELECT 1, * FROM public.powa_databases_src(0)) TO stdout	75	11 ms 186 µs	149 µs	0	0	0 B	1.01 M	0 B	0 B	0 B	0 B
/* sql from das */select r.* from (SELECT n.nspname AS schema_name, c.…	1	5 ms 355 µs	5 ms 355 µs	0	0	0 B	12.34 M	72.00 K	0 B	0 B	0 B
COPY (SELECT 1, * FROM public.pg_track_settings_settings_src(0)) TO st…	2	2 ms 367 µs	1 ms 184 µs	0	0	0 B	0 B	0 B	0 B	0 B	0 B
/* sql from das */select count($1) as table_count from information_sch…	1	916 µs	916 µs	0	0	0 B	7.29 M	8.00 K	0 B	0 B	0 B
SELECT pg_catalog.set_config(name, $1, $2) FROM pg_catalog.pg_settings…	1	783 µs	783 µs	0	0	0 B	0 B	0 B	0 B	0 B	0 B
SELECT setting FROM pg_settings WHERE name = $1 --WHERE name = 'server…	1	696 µs	696 µs	0	0	0 B	0 B	0 B	0 B	0 B	0 B
SAVEPOINT src	384	322 µs	1 µs	0	0	0 B	0 B	0 B	0 B	0 B	0 B
SELECT $1	35	148 µs	4 µs	0	0	0 B	0 B	0 B	0 B	0 B	0 B
/* sql from das */SELECT $3 FROM pg_class c LEFT JOIN pg_namespace n O…	1	148 µs	148 µs	0	0	0 B	136.00 K	0 B	0 B	0 B	0 B
COPY (SELECT 1, * FROM public.pg_track_settings_rds_src(0)) TO stdout	2	144 µs	72 µs	0	0	0 B	0 B	0 B	0 B	0 B	0 B

表 7-6 Details for all databases 字段解释

字段	注释
Query	执行的SQL。
(Execution) #	执行该SQL次数。
(Execution) Time	执行该SQL总时间。
(Execution) Avg time	执行该SQL平均时间。
(I/O Time) Read	读I/O等待时间。
(I/O Time) Write	写I/O等待时间。
(Blocks) Read	磁盘读页面数。
(Blocks) Hit	共享缓冲区命中页面数。
(Blocks) Dirtied	脏页面数。
(Blocks) Written	磁盘写页面数。
(Temp blocks) Read	磁盘读临时页面数。
(Temp blocks) Write	磁盘写临时页面数。

7.2.2 实例级性能指标

General Overview

图 7-4 General Overview 性能指标

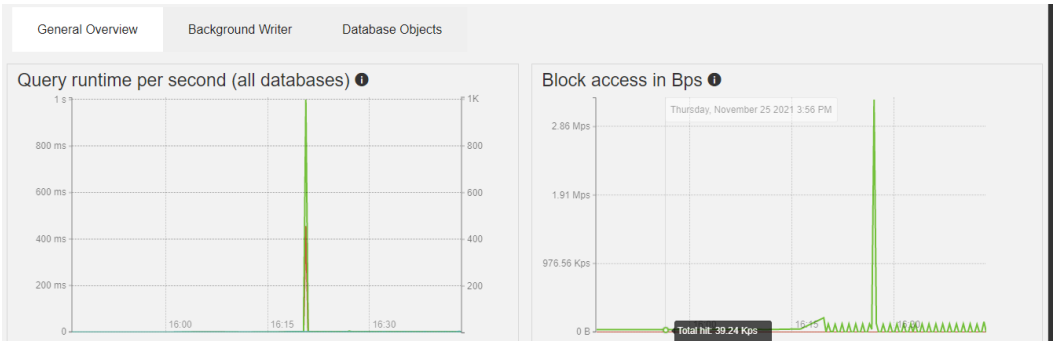


表 7-7 Query runtime per second (all databases)字段解释

字段	说明
Queries per sec	每秒执行查询的次数。
Runtime per sec	每秒执行的查询的总持续时间。
Avg runtime	平均查询时长。

表 7-8 Block access in Bps 字段解释

字段	说明
Total hit	在共享缓冲区中找到的数据量。
Total read	在操作系统缓存中找到或从磁盘读取的数据量。

Background Writer

图 7-5 Background Writer 性能指标

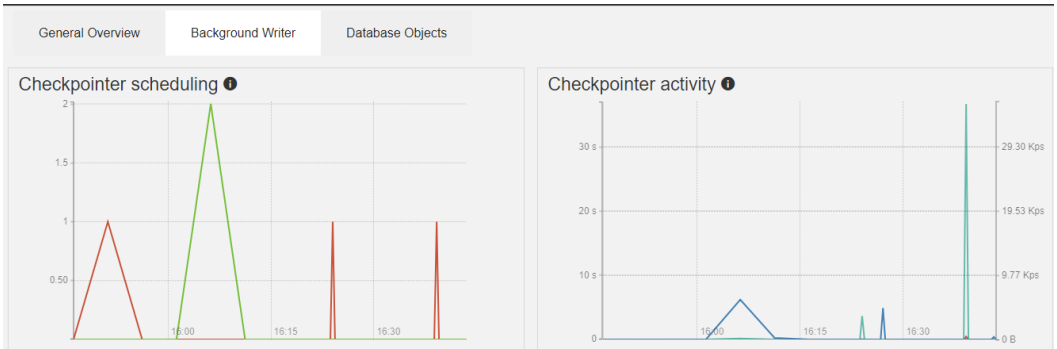


表 7-9 Checkpointer scheduling 字段解释

字段	说明
of requested checkpoints	已执行的请求检查点数。
of scheduled checkpoints	已执行的预定检查点数。

表 7-10 Checkpointer activity 字段解释

字段	说明
Buffers alloc	分配的缓冲区数。
Sync time	文件同步到磁盘的检查点处理部分所花费的总时间（单位：毫秒）。
Write time	在将文件写入磁盘的检查点处理部分中花费的总时间（单位：毫秒）。

表 7-11 Background writer 字段解释

字段	说明
Maxwritten clean	后台编写器因写入过多缓冲区而停止清理扫描的次数。
Buffers clean	后台写入器写入的缓冲区数。

表 7-12 Backends 字段解释

字段	说明
Buffers backend fsync	后端必须执行自己的 fsync 调用的次数。
Buffers backend	后端直接写入的缓冲区数。

Database Objects

图 7-6 Database Objects 性能指标

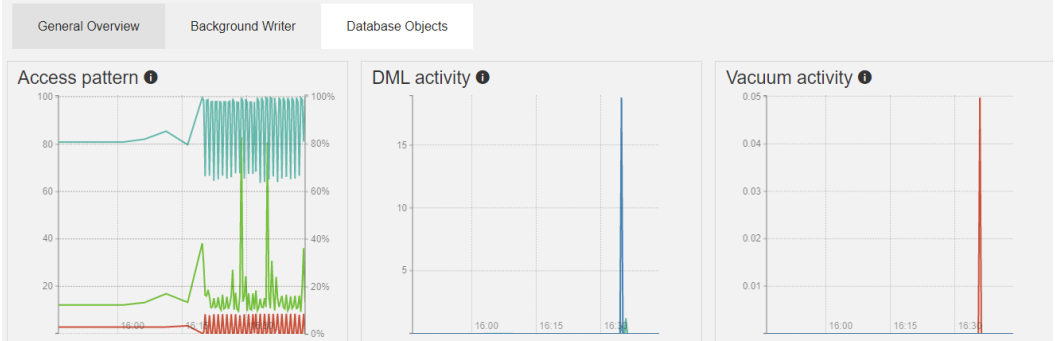


表 7-13 Access pattern 字段解释

字段	说明
Index scans ratio	索引扫描/序列扫描的比率。
Index scans	每秒索引扫描次数。
Sequential scans	每秒顺序扫描次数。

表 7-14 DML activity 字段解释

字段	说明
Tuples inserted	每秒插入的行数。
Tuples updated	每秒更新的行数。
Tuples HOT updated	每秒更新（HOT）。
Tuples deleted	每秒删除的行数。

表 7-15 Vacuum activity 字段解释

字段	说明
# Vacuum	每秒手动清理的次数。
# Autovacuum	每秒自动清理的次数。
# Analyze	每秒手动分析的次数。
# Autoanalyze	每秒自动分析的次数。

Details for all databases

图 7-7 Details for all databases 性能指标

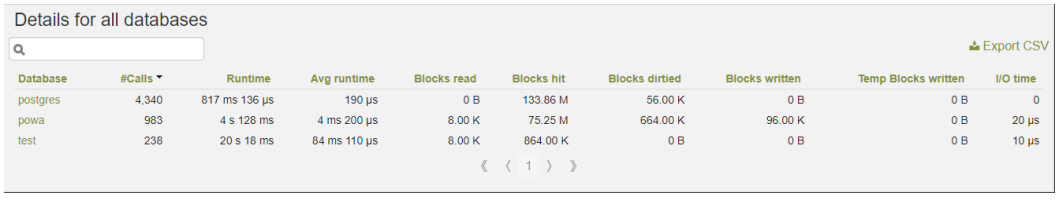


表 7-16

字段	注释
Database	数据库名称。

字段	注释
#Calls	执行SQL总数。
Runtime	执行SQL总耗时。
Avg runtime	执行SQL平均耗时。
Blocks read	磁盘读取的页面数。
Blocks hit	共享缓冲区命中的页面数。
Blocks dirtied	脏页数。
Blocks written	磁盘写页面数。
Temp Blocks written	磁盘写临时页面数。
I/O time	I/O等待时间。

7.3 部署 PoWA

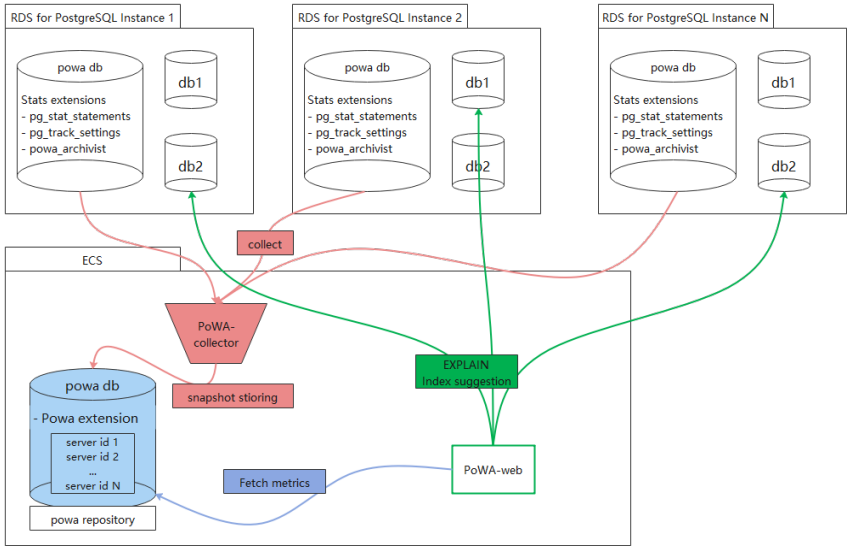
7.3.1 云上 PostgreSQL 实例部署 PoWA

华为云上进行远程模式部署PoWA，需要一台ECS，并在ECS上安装PoWA-archivist、PoWA-collector、PoWA-web。本章节主要介绍PoWA-archivist、PoWA-collector、PoWA-web的安装过程。

架构图

远程模式部署PoWA的架构如下图所示：

图 7-8 远程模式架构图



准备工作

- 已创建RDS for PostgreSQL 12.6实例。
- 已创建ECS并绑定弹性公网IP，本次演示所创建的ECS系统镜像为：CentOS 8.2 64bit。

Python3 的安装部署

安装PoWA-collecotor、PoWA-web依赖Python3环境，且使用pip3安装可以减少很多依赖环境安装的工作量。当前ECS已默认安装了 Python 3.6.8版本，由于版本偏低，安装最新版本的PoWA失败，建议安装最新的Python版本，详情请参见**安装Python 3.9.9**。

安装 PoWA-archivist

1. 通过wget命令获取**PoWA-archivist源码**：
   ```
   wget https://github.com/powa-team/powa-archivist/archive/refs/tags/REL_4_1_2.tar.gz
   ```
2. 将下载好的REL_4_1_2.tar.gz进行解压。
3. 进入解压后的目录，执行命令完成安装。
   ```
   make && make install
   ```

安装 PoWA-collector、PoWA-web

1. 切换到RDS for PostgreSQL数据库用户下，本次演示使用的是postgres。
   ```
   su - postgres
   ```
2. psycopg2 是 PoWA-collector、powa-web安装必不可少的依赖环境。
   ```
   pip install psycopg2
   pip install powa-collector
   pip install powa-web
   ```

安装完成后，查看路径树如下所示，表示PoWA-collector、PoWA-web均已安装完成。

```
/home/postgres/.local/bin
├── powa-collector.py
├── powa-web
└── __pycache__
```

创建 powa 插件

步骤1 使用root用户登录RDS for PostgreSQL实例的powa数据库。（若不存在，请先自行创建powa数据库）

步骤2 在powa数据库中创建powa插件。

```
select control_extension('create', 'pg_stat_statements');
select control_extension('create', 'btree_gist');
select control_extension('create', 'powa');
```

----结束

常见问题

Q：在执行**pip install psycopg2**时可能会遇到报错python setup.py build_ext --pg-config /path/to/pg_config build。

A：配置RDS for PostgreSQL的bin、lib路径到环境变量中，重新执行 pip install psycopg2 即可完成安装。

安装 Python 3.9.9

1. 环境准备。

   请按照以下顺序执行，否则安装Python3.9.9可能会有部分失败（SSL组件依赖失败），导致后续无法安装PoWA-collector、PoWA-web。

   ```
   yum install readline* -y
   yum install zlib* -y
   yum install gcc-c++ -y
   yum install sqlite* -y
   yum install openssl-* -y
   yum install libffi* -y
   ```

2. 安装**python 3.9.9**。

   a. 使用root用户执行下列命令。

   ```
   mkdir env
   cd env
   wget https://www.python.org/ftp/python/3.9.9/Python-3.9.9.tgz
   tar -xzvf Python-3.9.9.tgz
   cd Python-3.9.9
   ./configure --prefix=/usr/local/python3.9.9
   make && make install
   ```

   b. 创建软链接。

   ```
   ln -s /usr/local/python3.9.9/bin/python3.9 /usr/bin/python
   ln -s /usr/local/python3.9.9/bin/pip3.9 /usr/bin/pip
   ```

3. 验证安装是否成功。

   a. 验证安装，重点验证SSL功能。

   ```
   [root@ecs-ad4d Python-3.9.9]# python
   Python 3.9.9 (main, Nov 25 2021, 12:36:32)
   [GCC 8.4.1 20200928 (Red Hat 8.4.1-1)] on linux
   Type "help", "copyright", "credits" or "license" for more information.
   import ssl
   import urllib.request
   context = ssl._create_unverified_context()
   urllib.request.urlopen('https://www.example.com/',context=context).read()
   ```

   b. 如果有返回，说明安装成功。执行以下命令退出。

   ```
   quit()
   ```

7.3.2 自建 PostgreSQL 实例部署 PoWA

本章节主要介绍ECS自建PostgreSQL数据库部署PoWA的过程。

准备工作

已有自建PostgreSQL实例：

- 版本：PostgreSQL 12.6
- 管理员账号：postgres
- PostgreSQL专用存储数据库：powa-repository
- data路径：/home/postgres/data

部署 PoWA

步骤1 修改配置文件 /home/postgres/data/postgresql.conf，添加 pg_stat_statements 至参数 shared_preload_libraries，如下图所示：

```
shared_preload_libraries = 'pg_stat_statements'         # (change requires restart)
```

步骤2 执行命令重启数据库。

```
pg_ctl restart -D /home/postgres/data/
```

步骤3 使用postgres登录数据库，并创建database: powa，以及安装相关插件。

须知

创建的database必须命名为powa，否则PoWA运行过程中会报错，部分功能失效。

```
[postgres@ecs-ad4d ~]$ psql -U postgres -d postgres
psql (12.6)
Type "help" for help.
postgres=# create database powa;
CREATE DATABASE
postgres=# \c powa
You are now connected to database "powa" as user "postgres".
powa=# create extension pg_stat_statements ;
CREATE EXTENSION
powa=# create extension btree_gist ;
CREATE EXTENSION
powa=# create extension powa;
CREATE EXTENSION
```

步骤4 配置需要采集性能指标的实例信息。

1. 执行SQL，添加目标实例信息。

```
powa=# select powa_register_server(
    hostname => '192.168.0.1',
    alias => 'myInstance',
    port => 5432,
    username => 'user1',
    password => '**********',
    frequency => 300);
     powa_register_server
    ----------------------
     t
    (1 row)
```

2. 通过查看“powa_servers”表来获取当前采集指标的实例信息。

```
powa=# select * from powa_servers;
id | hostname | alias | port | username |  password  | dbname | frequency | powa_coalesce | retention |
allow_ui_connection |version
----+---------------+------------+------+----------+------------+--------+-----------+---------------+-----------
+-----------------
0 |           | <local>|  0 |          |           |        |        -1 |   100         | 00:00:00  | t           |
1 | 192.168.0.1 | myInstance | user1 | 5432     | ********** | powa   |       300 |           100 | 1 day       |
t             |
(2 rows)
```

须知

在录入目标实例信息、查询目标实例都会涉及到目标实例的IP、root用户账户、明文密码等重要隐私信息，可能会面临信安全风险。

请谨慎评估该插件带来的安全风险后决定是否使用该插件。

----结束

PoWA-collector 配置

启动PoWA-collector：

```
cd /home/postgres/.local/bin
./powa-collector.py &
```

PoWA-collector启动时，将按以下顺序搜索配置文件作为其配置：

1. /etc/powa-collector.conf
2. ~/.config/powa-collector.conf
3. ~/.powa-collector.conf
4. ./powa-collector.conf

配置文件中需要包含以下选项：

- repository.dsn：URL，用于通知 powa-collector 如何连接专用存储数据库（ powa-repository ）。
- debug：Boolean类型，用于指定是否在调试模式下启动 powa-collector。

本次演示中将配置写入文件./powa-collector.conf

```
{
   "repository": {
   "dsn": "postgresql://postgres@localhost:5432/powa"
   },
   "debug": true
}
```

PoWA-collector 的配置中并没有密码的配置，所以powa-repository数据库的pg_hba.conf中需要配置对应的连接策略为trust免密连接。

PoWA-web 配置

启动PoWA-web：

```
cd /home/postgres/.local/bin
./powa-web &
```

PoWA-web启动时，将按以下顺序搜索配置文件作为其配置：

1. /etc/powa-web.conf
2. ~/.config/powa-web.conf
3. ~/.powa-web.conf
4. ./powa-web.conf

本次实例中需要将配置内容写入文件./powa-web.conf中。

```
# cd /home/postgres/.local/bin
# vim ./powa-web.conf
# 写入配置内容，并保存
servers={
    'main': {
      'host': 'localhost',
      'port': '5432',
      'database': 'powa',
      'username': 'postgres',
      'query': {'client_encoding': 'utf8'}
    }
}
cookie_secret="SECRET_STRING"
```

本章节中powa-repository数据库pg_hab.conf中配置为trust，免密连接，因此未配置password。

7.4 在 PoWA 上查看指标详情

PoWA部署完成并成功启动PoWA-collector、PoWA-web后，可以通过浏览器登录PoWA，查看监控实例的指标详情。

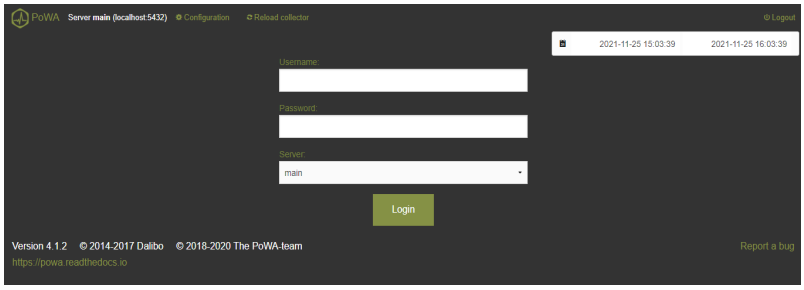
访问 PoWA

步骤1 使用浏览器访问PoWA。

说明

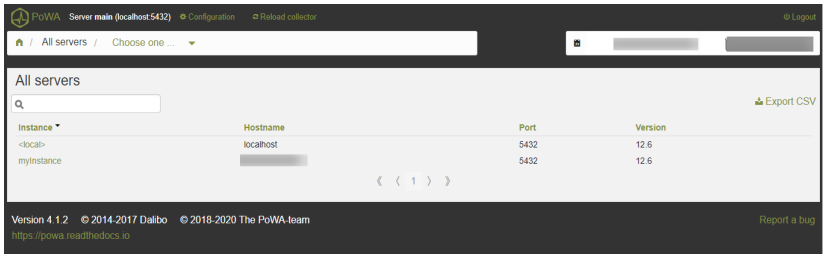
- powa-web.conf中未配置port选项，使用默认值8888
- 浏览器访问链接： http://*演示使用的ECS IP地址*:8888/

图 7-9 访问 PoWA



步骤2 输入用户名和密码，单击“Login”。

图 7-10 PoWA 首页



该PoWA中采集了两个PostgreSQL实例的信息：

- <local>：ECS自建PostgreSQL，作为powa-repository角色。
- myinstance：云数据库 RDS for PostgreSQL实例，作为性能数据采集目标。（ myinstance为在powa-repository中注册实例别名(alias字段) ）。

步骤3 单击对应的实例，即可查看实例具体的性能指标。

----结束

查看指标详情

PoWA可以采集、显示的性能指标非常丰富，下面的步骤以查看慢SQL指标为例。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域。

步骤3 单击页面左上角的，选择“数据库 > 云数据库 RDS”，进入RDS信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击操作列的“登录”，进入数据管理服务实例登录界面。

您也可以在“实例管理”页面，单击目标实例名称，在页面右上角，单击“登录”，进入数据管理服务实例登录界面。

步骤5 正确输入数据库用户名和密码，单击“登录”，即可进入您的数据库并进行管理。

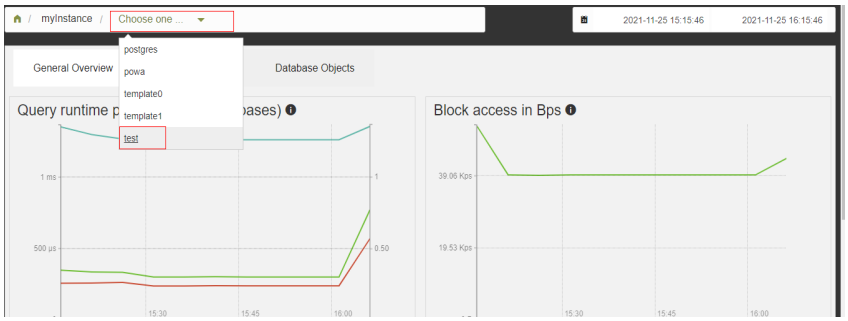
步骤6 在首页数据库列表栏单击“新建数据库”。

步骤7 在弹窗中填写数据库名称“test”、字符集等信息。

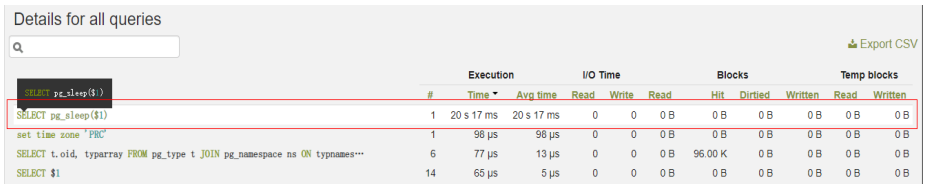
步骤8 单击“SQL查询”，在test数据库上执行慢SQL。

步骤9 等待大概5分钟后，在PoWA首页选中目标实例，选择test数据库，如下图所示：

图 7-11 PoWA 首页



在 Details for all queries中可以看到SELECT pg_sleep($1)语句，对应执行时间20s。



----结束

安装扩展插件并采集性能指标

下面步骤以pg_track_settings插件为例。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域。

步骤3 单击页面左上角的，选择“数据库 > 云数据库 RDS”，进入RDS信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击操作列的“登录”，进入数据管理服务实例登录界面。

您也可以在“实例管理”页面，单击目标实例名称，在页面右上角，单击“登录”，进入数据管理服务实例登录界面。

步骤5 正确输入数据库用户名和密码，单击“登录”，即可进入您的数据库并进行管理。

步骤6 选择powa数据库，执行SQL命令创建pg_track_settings插件。

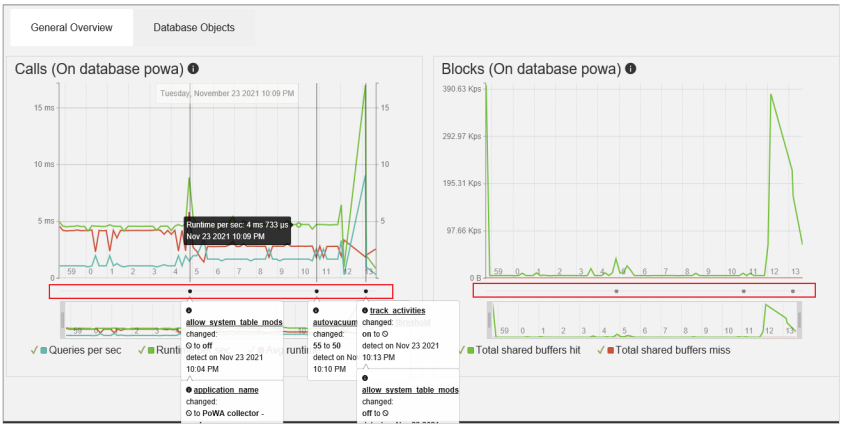
```
select control_extension('create', 'pg_track_settings');
```

步骤7 ECS上自建PostgreSQL(powa-repository)，安装pg_track_settings插件，并激活pg_track_settings插件采集性能指标。

```
# pg_track_settings
cd /home/postgres/env
wget https://github.com/rjuju/pg_track_settings/archive/refs/tags/2.0.1.tar.gz
mv 2.0.1.tar.gz pg_track_settings.2.0.1.tar.gz
tar -xzvf pg_track_settings.2.0.1.tar.gz
cd pg_track_settings-2.0.1
make && make install
# powa-repository
psql -d powa
powa=# create extension pg_track_settings ;
CREATE EXTENSION
# 激活目标实例 pg_track_settings 采集功功能
dbpowa=# select powa_activate_extension(1, 'pg_track_settings');
powa_activate_extension
-------------------------
t
(1 row)
```

步骤8 pg_track_settings插件扩展完成，进行验证。

在目标实例上修改参数“autovacuum_analyze_threshold”，原始默认值为50，修改后为55，等待大概5分钟，在PoWA页面上就可以看到对应参数修改的记录了，如下图所示：



上图中3个说明框中记录内容如下：

- 记录了pg_track_settings插件激活的时间点及当时数据库参数值。
- 记录了“autovacuum_analyze_threshold”参数被修改的时间以及原始值和修改后的值。
- 记录了pg_track_settings插件被取消的时间点及当时数据库参数值。

----结束

8 pg_dump 使用最佳实践

简介

pg_dump是PostgreSQL原生的备份工具。pg_dump生成的备份文件可以是一个SQL脚本文件，也可以是一个归档文件。详细信息请查看**pg_dump官方说明**。

- SQL脚本文件：纯文本格式的文件，其中包含将数据库重建到备份时状态的SQL命令。
- 归档格式的备份文件：必须与pg_restore一起使用来重建数据库，这种格式允许pg_restore选择恢复哪些数据。

注意事项

pg_dump适合单个库、schema级、表级导出，只会导出表及数据、函数等，数据库和用户需要提前在要恢复的库创建。

- --format=custom：备份为二进制格式，二进制格式的备份只能使用pg_restore来还原，并且可以指定还原的表。
- --format=plain：备份为文本，文本格式的备份还原，直接使用用户连接到对应的数据库执行备份文本即可。

限制条件

在使用pg_dump和pg_restore进行备份和恢复时，确保源数据库和目标库的版本一致，以避免出现兼容性问题，如果版本不一致可能会导致数据丢失或无法正确还原数据。

准备测试数据

```
# 创建数据库
create database dump_database;

# 登录dump_database数据库
\c dump_database

# 创建表1并插入数据
create table dump_table(id int primary key, content char(50));
insert into dump_table values(1,'aa');
insert into dump_table values(2,'bb');
```

```
# 创建表2并插入数据
create table dump_table2(id int primary key, content char(50));
insert into dump_table2 values(1,'aaaa');
insert into dump_table2 values(2,'bbbb');
```

使用 pg_dump 将数据库导出至 SQL 文件

语法

```
pg_dump --username=<DB_USER> --host=<DB_IPADDRESS> --port=<DB_PORT> --format=plain --
file=<BACKUP_FILE><DB_NAME>
```

- DB_USER为数据库用户。
- DB_IPADDRESS为数据库地址。
- DB_PORT为数据库端口。
- BACKUP_FILE为要导出的文件名称。
- DB_NAME为要导出的数据库名称。
- --format为导出的文件格式，plain为输出纯文本SQL脚本文件（默认）。其他选项详见**pg_dump官方说明**。

示例

- 导出数据库至SQL文件（INSERT语句）。
  ```
  $ pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --inserts --
  file=backup.sql dump_database
  Password for user root:
  ```
- 导出数据库中所有表结构至SQL文件。
  ```
  $ pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --schema-only --
  file=backup.sql dump_database
  Password for user root:
  ```
- 导出数据库中所有表数据至SQL文件。
  ```
  $ pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --data-only --
  file=backup.sql dump_database
  Password for user root:
  ```

命令执行完会生成“backup.sql”文件，如下：

```
[rds@localhost ~]$ ll backup.sql
-rw-r----- 1 rds rds 5657 May 24 09:21 backup.sql
```

使用 pg_dump 将数据库中的表导出至 SQL 文件

语法

```
pg_dump --username=<DB_USER> --host=<DB_ADDRESS> --port=<DB_PORT> --format=plain --
file=<BACKUP_FILE> <DB_NAME> --table=<TABLE_NAME>
```

- DB_USER为数据库用户。
- DB_ADDRESS为数据库地址。
- DB_PORT为数据库端口。
- BACKUP_FILE为要导出的文件名称。
- DB_NAME为要迁移的数据库名称。
- TABLE_NAME为要迁移的数据库中指定表名称。
- --format为导出的文件格式，plain为输出纯文本SQL脚本文件（默认）。其他选项详见**pg_dump官方说明**。

示例

- 导出数据库中指定的单表至SQL文件。
```
 $ pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --file=backup.sql
dump_database --table=dump_table
 Password for user root
```
- 导出数据库中指定的多表至SQL文件。
```
 $ pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --file=backup.sql
dump_database --table=dump_table --table=dump_table2
 Password for user root:
```
- 导出数据库中以ts_开头的所有表至SQL文件。
```
 $ pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --file=backup.sql
dump_database --table=ts_*
 Password for user root:
```
- 导出数据库中除ts_开头之外的所有表至SQL文件。
```
 $ pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --file=backup.sql
dump_database -T=ts_*
 Password for user root:
```

命令执行完会生成“backup.sql”文件，如下：

```
[rds@localhost ~]$ ll backup.sql
-rw-r----- 1 rds rds 5657 May 24 09:21 backup.sql
```

使用 pg_dump 导出特定 schema 下的数据

语法

```
pg_dump --username=<DB_USER> --host=<DB_IPADDRESS> --port=<DB_PORT> --format=plain --
schema=<SCHEMA> <DB_NAME> --table=<BACKUP_FILE>
```

- DB_USER为数据库用户。
- DB_IPADDRESS为数据库地址。
- DB_PORT为数据库端口。
- BACKUP_FILE为要导出的文件名称。
- DB_NAME为要导出的数据库名称。
- SCHEMA为要导出的schema名称。
- --format为导出的文件格式，plain为输出纯文本SQL脚本文件（默认）。其他选项详见**pg_dump官方说明**。

示例

- 导出指定库中public schema的所有数据。
```
pg_dump --username=root --host=192.168.61.143 --port=5432 --format=plain --schema=public
dump_database  > backup.sql
```
- 导出指定库中除public schema以外的所有数据，结果以自定义压缩格式导出。
```
pg_dump --username=root --host=192.168.61.143 --port=5432 --format=custom -b -v -N public
dump_database > all_sch_except_pub.backup
```

还原数据

使用纯文本SQL脚本文件导出的数据，还原时直接使用psql命令即可，比如：

```
# 还原特定数据库
psql --username=root --host=192.168.61.143 --port=5432  backup_database  < backup.sql

# 还原特定表
psql --username=root --host=192.168.61.143 --port=5432  backup_database --
table=dump_table  < backup.sql
```

```
# 还原特定schema
psql --username=root --host=192.168.61.143 --port=5432  backup_database --schema=public  < backup.sql
```

说明

在恢复之前在目标库中创建数据库backup_database。

使用其他格式导出的数据进行还原时，需要使用pg_restore。**pg_restore**用于恢复由pg_dump转储的任何非纯文本格式中的PostgreSQL数据库。

```
pg_restore --username=root --host=192.168.61.143 --port=5432 --dbname=backup_database --format=custom all_sch_except_pub.backup --verbose
```

常见问题

1. 如果在使用pg_dump导出时，报错用户权限不足。

   解决方法：

   检查是否使用的root用户导出，如果不是root用户则会报权限不足；如果使用root用户导出，仍然提示没有权限，请检查数据库版本，root用户执行pg_dump命令需要内核版本为支持root提权的版本，支持root提权版本情况见**root用户权限说明**。

2. 将备份的文件导入到RDS for PostgreSQL目标数据库时发现control_extension等几个函数报错。

   解决方法：

   由于目标库中自带这些函数，因此该报错可以忽略。

9 PgBouncer 使用最佳实践

PgBouncer 介绍

PgBouncer是为PostgreSQL提供的轻量级连接池工具，作用如下：

- 能够缓存和PostgreSQL的连接，当有连接请求进来的时候，直接分配空闲进程，而不需要PostgreSQL fork出新进程来建立连接，以节省创建新进程，创建连接的资源消耗。
- 能够有效提高连接的利用率，避免过多的无效连接，导致数据库消耗资源过大，CPU占用过高。
- 对客户端连接进行限制，预防过多或恶意的连接请求。

轻量级体现在：

- 使用libevent进行socket通信，通信效率高。
- C语言编写，效率高，每个连接仅消耗2KB内存。

PgBouncer目前支持三种连接池模型：

- session会话级连接。只有与当客户端的会话结束时，PgBouncer才会收回已分配的连接。
- transaction事务级连接。当事务完成后，PgBouncer会回收已分配的连接。也就是说客户端只是在事务中才能独占此连接，非事务请求没有独享的连接。
- statement语句级连接。任何数据库请求完成后，PgBouncer都会回收连接。此种模式下，客户端不能使用事务，否则会造成数据的不一致。

PgBouncer默认选项是session，建议修改为transaction。

安装配置

在云上部署PgBouncer连接池，需要先**购买弹性云服务器**。建议选择与后端RDS实例相同VPC、相同子网创建，降低网络通信时延。购买成功后登录ECS进行环境搭建。

1. 由于PgBouncer是基于libevent开发，需要安装libevent-devel openssl-devel两个依赖包。命令如下：
   ```
   yum install -y libevent-devel
   yum install -y openssl-devel
   ```
2. 依赖包安装后，在PgBouncer官网下载源码，以普通用户身份进行编译、安装。
   ```
   su - pgbouncer
   tar -zxvf pgbouncer-1.19.0.tar.gz
   ```



```
cd  pgbouncer-1.19.0
./configure --prefix=/usr/local
make
make install
```

3. 创建如下目录，用于保存PgBouncer的生成文件（日志、进程标识等）。

```
mkdir -p /etc/pgbouncer/
mkdir -p /var/log/pgbouncer/
mkdir -p /var/run/pgbouncer/
```

4. 启动PgBouncer前，需要构建配置文件“pgbouncer.ini”。

```
[databases]
* = host=127.0.0.1 port=5432
[pgbouncer]
logfile = /var/log/pgbouncer/pgbouncer.log
pidfile = /var/run/pgbouncer/pgbouncer.pid
listen_addr = *
listen_port = 6432
auth_type = md5
auth_file = /etc/pgbouncer/userlist.txt
admin_users = postgres
stats_users = stats, postgres
pool_mode = transaction
server_reset_query = DISCARD ALL
max_client_conn = 100
default_pool_size = 20
;; resolve: unsupported startup parameter: extra_float_digits
;;ignore_startup_parameters = extra_float_digits
```

配置文件中各参数含义可参考**PgBouncer官方文档**。

启动 pgbouncer

PgBouncer不能以root身份启动，需要基于普通用户身份进行启动。

```
pgbouncer -d /etc/pgbouncer/pgbouncer.ini
```

启动后，可通过**netstat -tunlp | grep pgbouncer**查看连接池的监听端口，而后进行连接：

```
psql -U root -d postgres -h 127.0.0.1 -p 6432
Password for user root:
psql (12.13)
Type "help" for help.
postgres=> \l
                                  List of databases
   Name    |   Owner   | Encoding |   Collate   |    Ctype    |    Access privileges
-----------+-----------+----------+-------------+-------------+-------------------------
 postgres  | pgbouncer | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | pgbouncer | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/pgbouncer
           |           |          |             |             | pgbouncer=CTc/pgbouncer
 template1 | pgbouncer | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/pgbouncer
```

停止 PgBouncer

可以直接通过Kill命令停止。

```
kill `cat /var/run/pgbouncer/pgbouncer.pid`
cat /var/run/pgbouncer/pgbouncer.pid | xargs kill -9
```

PgBouncer 管理

PgBouncer对外提供了一个虚拟数据库pgbouncer，之所以称为虚拟数据库，是因为它可以提供像PostgreSQL那样的数据库操作界面，但是这个数据库却并不是真实存在的，而是PgBouncer虚拟出来的一个命令行界面。登录虚拟数据库：

```
psql -p 6432 -d pgbouncer
```

如果修改了一些配置参数，可以不用重启PgBouncer而是**reload**使其生效。

```
pgbouncer=# reload;
RELOAD
```

登录后可以通过**show help**查看命令帮助，通过**show clients**查看客户端连接信息，通过**show pools**查看连接池信息。

实现读写分离示例

PgBouncer并不支持自动解析读写请求，并进行读写分离路由，需要业务侧对读写操作进行区分。

1. 首先，修改配置文件“pgbouncer.ini”配置的数据库信息，添加主库、只读库连接配置。本例中参数设置如下：

   ```
   [databases]
   ;; * = host=127.0.0.1 port=5432
   #配置只读库的连接信息
   mydb_read：host=10.7.131.69 port=5432 dbname=postgres user=root password=***
   # 配置主库的连接信息
   mydb_write:  host=10.8.115.171 port=5432 dbname=postgres user=root password=***
   [pgbouncer]
   logfile = /var/log/pgbouncer/pgbouncer.log
   pidfile = /var/run/pgbouncer/pgbouncer.pid
   listen_addr = *
   listen_port = 6432
   auth_type = md5
   auth_file = /etc/pgbouncer/userlist.txt
   admin_users = postgres
   stats_users = stats, postgres
   pool_mode = transaction
   server_reset_query = DISCARD ALL
   max_client_conn = 100
   default_pool_size = 20
   ;; resolve: unsupported startup parameter: extra_float_digits
   ;;ignore_startup_parameters = extra_float_digits
   ```

2. 验证是否能连接主库和只读数据库。通过psql成功连接只读、主库，可支持业务读写分离。

   ```
   psql -U root -d mydb_write -h 127.0.0.1 -p 6432
   Password for user root:
   psql  (14.6)
   mydb_write=>  SELECT pg_is_in_recovery();
    pg_is_in_recovery
   -----------------------
    f
   (1 row)
   psql -U root -d mydb_read -h 127.0.0.1 -p 6432
   Password for user root:
   psql  (14.6)
   mydb_read=>  SELECT pg_is_in_recovery();
   pg_is_in_recovery
   -----------------------
    t
   (1 row)
   ```

10 创建只读用户最佳实践

RDS for PostgreSQL中没有创建只读用户的语法，可以通过用户权限以及角色属性设置来达到只读用户的效果，本章节提供创建只读用户的方法。

操作步骤

1. 创建一个用户名为readonly，密码为readonlypasswd的用户。
   ```
   CREATE USER readonly WITH ENCRYPTED PASSWORD 'readonlypasswd';
   ```
2. 使用root用户执行以下SQL，设置readonly用户默认事务为只读。
   ```
   ALTER USER readonly SET default_transaction_read_only=on;
   ```
3. 使用root用户执行以下SQL，将schema（以public为例）的USAGE的权限赋予readonly用户。
   ```
   GRANT USAGE ON SCHEMA public to readonly;
   ```
4. 使用root用户执行以下SQL，将public下所有表的SELECT权限赋予readonly用户。
   ```
   GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;
   ```
5. 将public这个schema下新创建的表的SELECT权限默认赋给readonly用户。

   使用哪个用户执行，则该用户新创建的表的SELECT权限默认赋给readonly用户。
   ```
   ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO readonly ;
   ```
6. 如果需要在某个数据库下设置只读用户readonly，则需要对该数据库下所有的schema重复执行3~5。
7. 如果需要对某个实例设置全局的只读用户readonly，则需要分别连接每个数据库，执行步骤6。

11 基于 RDS for PostgreSQL 的表设计入门

RDS for PostgreSQL面向企业复杂SQL处理的OLTP在线事务处理场景，支持NoSQL数据类型（JSON/XML/hstore），支持GIS地理信息处理，在可靠性、数据完整性方面有良好声誉，适用于互联网网站、位置应用系统、复杂数据对象处理等应用场景。

本文将以某电商平台为例，介绍如何针对业务场景设计表结构，并在华为云RDS for PostgreSQL中演示相关库表的创建和应用请求处理。

场景说明

某电商平台针对其核心业务场景（用户在线购物、订单管理与商品统计分析）进行数据库表结构设计，以保障关键业务数据的一致性与完整性，同时提升在高并发场景下的事务处理效率。

表设计

- 用户表(t_customers)
  该表存储用户信息，每个用户在该表中有一条记录，并且有唯一的用户ID(cust_id)。

  表 11-1 用户表

字段名	字段类型	字段说明
cust_id	SERIAL	用户ID，主键。
cust_nickname	VARCHAR(50)	用户昵称，添加非空约束。
cust_gender	VARCHAR(10)	用户性别。
cust_birthday	DATE	用户生日。
cust_address	TEXT	用户收货地址，添加非空约束。

- 商品表(t_goods)

该表存储商品信息，每件商品在该表中有一条记录，并且有唯一的商品ID(item_id)。

表 11-2 商品表

字段名	字段类型	字段说明
item_id	SERIAL	商品ID，主键。
item_name	VARCHAR(100)	商品名称，添加非空约束。
item_category	VARCHAR(50)	商品类别，添加非空约束。
item_desc	TEXT	商品描述。
item_price	DECIMAL(10,2)	商品单价，添加非空约束。
stock_quantity	INTEGER	库存数量，添加非空约束。

- 订单表(t_orders)

  该表存储订单信息，用于关联用户和商品。每个订单在该表中有一条记录，并且有唯一的订单ID(trans_id)。

  表 11-3 订单表

字段名	字段类型	字段说明
trans_id	SERIAL	订单ID，主键。
cust_id	INTEGER	订单用户ID，关联用户表(t_customers)中的用户ID(cust_id)。
item_id	INTEGER	订单商品ID，关联商品表(t_goods)中的商品ID(item_id)。
purchase_quantity	INTEGER	商品购买数量，添加非空约束。
total_price	DECIMAL(10,2)	商品总价，添加非空约束。
order_time	TIMESTAMP	订单生成时间，添加DEFAULT约束。
order_status	VARCHAR(20)	订单状态，添加非空约束。

创建表

1. 在数据库中创建表。
   - 创建用户表(t_customers)。
     ```
     CREATE TABLE t_customers (
       cust_id SERIAL PRIMARY KEY,
       cust_nickname VARCHAR(50) NOT NULL,
       cust_gender VARCHAR(10),
       cust_birthday DATE,
       cust_address TEXT NOT NULL
     );
     ```
   - 创建商品表(t_goods)。
     ```
     CREATE TABLE t_goods(
       item_id SERIAL PRIMARY KEY,
       item_name VARCHAR(100) NOT NULL,
       item_category VARCHAR(50) NOT NULL,
       item_desc TEXT,
       item_price DECIMAL(10,2) NOT NULL,
       stock_quantity INTEGER NOT NULL
     );
     ```
   - 创建订单表(t_orders)。
     ```
     CREATE TABLE t_orders (
       trans_id SERIAL PRIMARY KEY,
       cust_id INTEGER REFERENCES t_customers(cust_id),
       item_id INTEGER REFERENCES t_goods(item_id),
       purchase_quantity INTEGER NOT NULL,
       total_price DECIMAL(10,2) NOT NULL,
       order_time TIMESTAMP NOT NULL DEFAULT NOW(),
       order_status VARCHAR(20) NOT NULL
     );
     ```
2. 假设商城上架了一些商品，用户已经在平台上注册了账号并且购买了部分商品，相关数据已传回系统数据库。在数据库中模拟插入如下测试数据：
   ```
   -- 用户数据
   INSERT INTO t_customers (cust_nickname, cust_gender, cust_birthday, cust_address) VALUES
     ('Rich Man', 'Female', '1995-08-12', '北京市海淀区'),
     ('Superman', 'Male', '1998-03-25', '上海市浦东新区');
   -- 商品数据
   INSERT INTO t_goods(item_name, item_category, item_desc, item_price, stock_quantity) VALUES
     ('智能手机X', '电子产品', '最新款智能手机', 5999.00, 22),
     ('无线耳机Pro', '电子产品', '爆款产品', 1299.00, 200),
     ('棉质T恤', '服装', '纯棉材质', 199.00, 300);
   -- 订单数据
   INSERT INTO t_orders(cust_id, item_id, purchase_quantity, total_price, order_status) VALUES
     (1, 1, 1, 5999.00, '已支付'),
     (1, 2, 2, 2598.00, '已发货'),
     (2, 3, 5, 995.00, '已完成');
   ```

场景示例

- Superman用户查询自己的所有订单（cust_id为2）。
  ```
  SELECT m.item_name, t.purchase_quantity, t.total_price, t.order_time, t.order_status
  FROM t_customers c
  JOIN t_orders t ON c.cust_id = t.cust_id
  JOIN t_goods m ON t.item_id = m.item_id
  WHERE c.cust_id = 2;
  ```
  返回结果如下：
  ```
  item_name | purchase_quantity | total_price |         order_time         | order_status
  ----------+-------------------+-------------+----------------------------+--------------
   棉质T恤   |                 5 |      995.00 | 2025-07-31 15:04:03.593379 | 已完成
  (1 row)
  ```
- 平台统计各商品销量。
  ```
  SELECT m.item_name, SUM(t.purchase_quantity) AS total_sold
  FROM t_goods m
  ```

```
LEFT JOIN t_orders t ON m.item_id = t.item_id
GROUP BY m.item_name
ORDER BY total_sold DESC;
```

返回结果如下：

```
item_name  | total_sold
-------------+------------
 棉质T恤     |          5
 无线耳机Pro |          2
 智能手机X   |          1
(3 rows)
```

- 查询库存数量低于50件的电子产品。

```
SELECT item_name, stock_quantity
FROM t_goods
WHERE stock_quantity < 50
AND item_category = '电子产品';
```

返回结果如下：

```
item_name | stock_quantity
-----------+----------------
 智能手机X |             22
(1 row)
```

12 RDS for PostgreSQL 权限管理最佳实践

基本概念

PostgreSQL 的用户权限管理基于角色（Role）和权限（Privilege）两个概念开展的，这两个概念具体解释如下：

- 角色（Role）

  角色是权限的集合载体，可理解为 "用户" 或 "用户组"：

  - 登录角色（Login Role）：具备登录数据库的权限（LOGIN属性），类似传统意义上的 "用户"。
  - 组角色（Group Role）：不直接登录（无LOGIN属性），用于批量管理权限，其他角色可加入该组继承权限。
  - 所有角色默认属于public角色（特殊组角色），需注意其默认权限。
- 权限（Privilege）

  权限是对数据库对象（表、函数、schema 等）的操作许可，常见类型包括：

  - 表 / 视图：SELECT（查询）、INSERT（插入）、UPDATE（更新）、DELETE（删除）、TRUNCATE（清空）等。
  - 数据库：CONNECT（连接）、CREATE（创建 schema）等。
  - Schema：CREATE（创建对象）、USAGE（访问对象）等。
  - 函数 / 存储过程：EXECUTE（执行）。
  - 序列：USAGE（使用）、UPDATE（修改）等。

权限管理基本原理

RDS for PostgreSQL 对用户User 权限管理的细分操作集中在Role上，需要注意的是，Role本身并不赋予登录权限，创建User后，赋予对应的Role，才能通过User登录数据库，并获得Role对应的权限，User本身的权限随Role的权限的变化而变化。

权限管理建议

- RDS for PostgreSQL默认提供一个高权限账号root，该账号具有高级别权限，建议由少量资深DBA掌握。
- 团队管理者可以创建1个资源用户账号，用于对Role进行管理，可以根据实际情况创建多个Role实现权限的细分管理。

- 在细分Role的基础上创建多个业务账号，可以使用业务账号登录和操作数据库。
- 如果团队prj项目涉及多个Schema，Role的权限划分设计尽量以Schema为单位，例如{prj}_{role}_{schema}_readonly / {prj}_{role}_{schema}_write 等。所以，这一需要注意的是，业务表请勿放到public中。因为PostgreSQL默认所有用户对public都有CREATE和USAGE权限。

权限规划样例

- 资深DBA拥有RDS for PostgreSQL实例的高权限账号，名称是 root。
- 项目管理者拥有1个资源账号，用于账号管理和Role管理，名称是db_prj_owner。
- 业务项目名称是db_prj，新建schema名称是db_prj、db_prj_1和db_prj2。

项目中新增的资源owner账号和Role规划如下：

表 12-1 权限说明

user/Role	schema中表权限	schema中存储过程权限
root，最高权限账号，默认创建完实例后拥有。	• DDL：CREATE、DROP、ALTER* • DQL：SELECT* • DML：UPDATE、INSERT、DELETE	• DDL：CREATE、DROP、ALTER* • DQL：SELECT，调用存储过程。
db_prj_owner，是唯一的项目资源owner账号。	• DDL：CREATE、DROP、ALTER* • DQL：SELECT* • DML：UPDATE、INSERT、DELETE	• DDL：CREATE、DROP、ALTER* • DQL：SELECT，调用存储过程。
db_prj_role1_readwrite (role)	• DQL：SELECT* • DML：UPDATE、INSERT、DELETE	DQL（SELECT，调用存储过程），若存储过程有DDL操作，会抛出权限相关错误。
db_prj_role2_readonly (role)	DQL(SELECT)	DQL（SELECT，调用存储过程），若存储过程有DDL或者DML操作，会抛出权限相关错误。

配置操作步骤

1. 创建项目资源owner账号db_prj_owner和项目Role。

   DBA使用 root 高权限账号执行如下操作。

```
--- db_prj_owner 是项目管理账号,此处密码仅为示例，请注意修改。
CREATE USER db_prj_owner WITH LOGIN PASSWORD 'XXXXXXX';

CREATE ROLE db_prj_role1_readwrite;
CREATE ROLE db_prj_role2_readonly;
```

```
--- 设置: 对于db_prj_owner 创建的表，db_prj_role1_readwrite 有 DQL（SELECT）、DML（UPDATE、INSERT、DELETE）权限。
ALTER DEFAULT PRIVILEGES FOR ROLE db_prj_owner GRANT ALL ON TABLES TO db_prj_role1_readwrite;

--- 设置: 对于db_prj_owner创建的SEQUENCES，db_prj_role1_readwrite 有 DQL（SELECT）、DML（UPDATE、INSERT、DELETE）权限。
ALTER DEFAULT PRIVILEGES FOR ROLE db_prj_owner GRANT ALL ON SEQUENCES TO db_prj_role1_readwrite;

--- 设置: 对于 db_prj_owner创建的表， db_prj_role2_readonly 只有 DQL（SELECT）权限。
ALTER DEFAULT PRIVILEGES FOR ROLE db_prj_owner GRANT SELECT ON TABLES TO db_prj_role2_readonly;
```

2. 创建db_prj_user_readwrite、db_prj_user_readonly业务账号。

   DBA使用root高权限账号执行如下操作。

```
--- db_prj_user_readwrite只有 DQL（SELECT）、DML（UPDATE、INSERT、DELETE）权限。
CREATE USER db_prj_user_readwrite WITH LOGIN PASSWORD 'XXXXXXX';
GRANT db_prj_role1_readwrite TO db_prj_user_readwrite;

--- db_prj_user_readonly只有 DQL（SELECT）权限。
CREATE USER db_prj_user_readonly WITH LOGIN PASSWORD 'XXXXXXXX';
GRANT db_prj_role2_readonly TO db_prj_user_readonly;
```

3. 创建schema，并授权给项目Role。

   DBA使用root高权限账号执行如下操作。

```
--- schema db_prj 的owner是 db_prj_owner 账号
CREATE SCHEMA db_prj  AUTHORIZATION db_prj_owner;

--- 授权ROLE相关SCHEMA访问权限。
GRANT USAGE ON SCHEMA db_prj TO db_prj_role1_readwrite;
GRANT USAGE ON SCHEMA db_prj TO db_prj_role2_readonly;
```

说明

db_prj_user_readwrite 和 db_prj_user_readonly 自动继承了相关Role的权限变更，不需要再额外操作。

开发应用场景

场景1：使用 db_prj_owner 账号：对schema db_prj 中的表进行DDL操作

```
CREATE TABLE db_prj.table1(col1 bigserial primary key, col2 int);
DROP TABLE db_prj.table1;
```

场景2：使用 db_prj_user_readwrite/db_prj_user_readonly 账号进行业务开发

业务开发遵循最小权限原则，在不需要改变现有数据的场景中，使用db_prj_user_readonly 账号，需要改变数据（DML操作，Insert，update，delete）的操作的地方才使用db_prj_user_readwrite 账号，这样便于处理好不同账号的数据操作安全隔离。

- 使用 db_prj_user_readwrite 账号，对schema db_prj中的表进行增删查改操作：

```
# 普通DML和DQL操作不受影响。
INSERT INTO db_prj.table1 (col2) VALUES(88),(99);
SELECT * FROM db_prj.table1;

--- db_prj_user_readwrite 没有 DDL（CREATE、DROP、ALTER）权限
CREATE TABLE db_prj.table2(id int);
ERROR:  permission denied for schema db_prj
LINE 1: create table db_prj.table2(id int);

DROP TABLE db_prj.table1;
ERROR:  must be owner of table test
```

```
ALTER TABLE db_prj.table1 ADD col3 int;
ERROR:  must be owner of table test

CREATE INDEX idx_xxxx on db_prj.table1(col1);
ERROR:  must be owner of table test
```

- 使用 db_prj_user_readonly 账号，对schema db_prj 中的表进行操作：

```
INSERT INTO db_prj.table1 (col2) VALUES(88),(99);
ERROR:  permission denied for table table1

SELECT id,name FROM db_prj.table1 limit 1;
 col1 | col2
------+-------
  1 | 88
(1 row)
```

场景3：给其他项目用户访问授权

如果有另外1个项目 db_prj1，需求为账号 db_prj1_user_readwrite 增加 db_prj 项目的表只读权限。DBA使用 root 账号做如下操作：

```
--- 给账号 db_prj1_user_readwrite 加上 db_prj_role2_readonly 权限集合。
GRANT db_prj_role2_readonly TO db_prj1_user_readwrite;
```

场景4：项目新增 schema db_prj1，并授权给项目Role

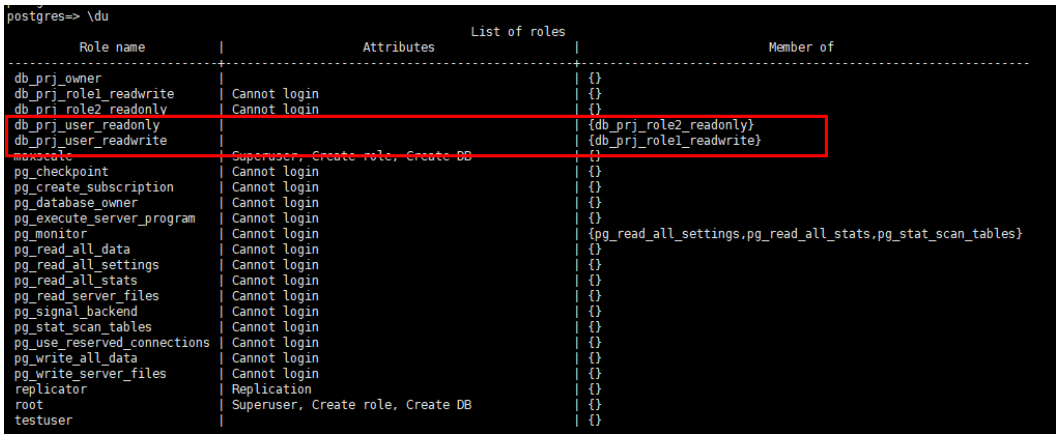
db_prj1_user_readwrite、db_prj1_user_readonly、db_prj1_user_readwrite 账号自动继承了相关Role的权限变更，不需要再额外操作。DBA使用 root 高权限账号做如下操作：

```
CREATE SCHEMA db_prj1 AUTHORIZATION db_prj_owner;

--- 授权ROLE相关SCHEMA访问权限。
--- CREATE 使得 db_prj_owner 对schema db_prj1 中的表有 DDL（CREATE、DROP、ALTER）权限。
GRANT USAGE ON SCHEMA db_prj1 TO db_prj_role1_readwrite;
GRANT USAGE ON SCHEMA db_prj1 TO db_prj_role2_readonly;
```

账号权限查询

- 使用PostgreSQL客户端命令行终端连接RDS for PostgreSQL数据库。然后使用\du命令查看所有用户和拥有的角色，例如：

图 12-1 使用\du 命令查询用户角色



```
postgres=> \du
                                   List of roles
         Role name          |                  Attributes                  |                          Member of
----------------------------+----------------------------------------------+-------------------------------------------------------------
 db_prj_owner               |                                              | {}
 db_prj_role1_readwrite     | Cannot login                                 | {}
 db_prj_role2_readonly      | Cannot login                                 | {}
 db_prj_user_readonly       |                                              | {db_prj_role2_readonly}
 db_prj_user_readwrite      |                                              | {db_prj_role1_readwrite}
 mamcsdc                    | Superuser, Create role, Create DB            | {}
 pg_checkpoint              | Cannot login                                 | {}
 pg_create_subscription     | Cannot login                                 | {}
 pg_database_owner          | Cannot login                                 | {}
 pg_execute_server_program  | Cannot login                                 | {}
 pg_monitor                 | Cannot login                                 | {pg_read_all_settings,pg_read_all_stats,pg_stat_scan_tables}
 pg_read_all_data           | Cannot login                                 | {}
 pg_read_all_settings       | Cannot login                                 | {}
 pg_read_all_stats          | Cannot login                                 | {}
 pg_read_server_files       | Cannot login                                 | {}
 pg_signal_backend          | Cannot login                                 | {}
 pg_stat_scan_tables        | Cannot login                                 | {}
 pg_use_reserved_connections | Cannot login                                 | {}
 pg_write_all_data          | Cannot login                                 | {}
 pg_write_server_files      | Cannot login                                 | {}
 replicator                 | Replication                                  | {}
 root                       | Superuser, Create role, Create DB            | {}
 testuser                   |                                              | {}
```

从查询结果中可以看出：db_prj_user_readwrite 账号的Member of列中，内容为db_prj_role2_readonly, db_prj1_role1_readwrite。

- 使用SQL查询用户角色：

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
        FROM pg_catalog.pg_auth_members m
        JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
        WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolbypassrls
FROM pg_catalog.pg_roles r
WHERE r.rolname !~ '^pg_'
ORDER BY 1;
```

图 12-2 使用 SQL 查询用户角色

```
postgres=> SELECT r.rolname, r.rolsuper, r.rolinherit,
    r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
    r.rolconnlimit, r.rolvaliduntil,
    ARRAY(SELECT b.rolname
          FROM pg_catalog.pg_auth_members m
          JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
          WHERE m.member = r.oid) as memberof
  , r.rolreplication
  , r.rolbypassrls
  FROM pg_catalog.pg_roles r
  WHERE r.rolname !~ '^pg_'
  ORDER BY 1;
        rolname         | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin | rolconnlimit | rolvaliduntil |         memberof         | rolreplication | rolbypassrls
------------------------+----------+------------+---------------+-------------+-------------+--------------+---------------+--------------------------+----------------+--------------
 db_prj_owner           | f        | t          | f             | f           | t           |           -1 |               | {}                       | f              | f
 db_prj_role1_readwrite | f        | t          | f             | f           | f           |           -1 |               | {}                       | f              | f
 db_prj_role2_readonly  | f        | t          | f             | f           | f           |           -1 |               | {}                       | f              | f
 db_prj_user_readonly   | f        | t          | f             | f           | t           |           -1 |               | {db_prj_role2_readonly}  | f              | f
 db_prj_user_readwrite  | f        | t          | f             | f           | t           |           -1 |               | {db_prj_role1_readwrite} | f              | f
 maxscale               | t        | t          | t             | t           | t           |           -1 |               | {}                       | f              | f
 replicator             | f        | t          | f             | f           | t           |           -1 |               | {}                       | t              | f
 root                   | t        | t          | t             | t           | t           |           -1 |               | {}                       | f              | f
 testuser               | f        | t          | f             | f           | t           |           -1 |               | {}                       | f              | f
 ytwork                 | t        | t          | t             | t           | t           |           -1 |               | {}                       | t              | t
(10 rows)
```

13 WAL 日志堆积问题定位及处理方法

指标异常说明

PostgreSQL使用WAL（Write-Ahead Logging）日志机制来保证数据的持久性和一致性。WAL 日志记录了对数据库的所有更改操作，在主备复制、崩溃恢复、逻辑复制等场景中都扮演关键角色。

数据库提供了以下与WAL日志相关的监控指标，建议重点关注以下指标：

- 事务日志使用量（WAL日志）：rds040_transaction_logs_usage
- 非活跃逻辑复制槽数量：inactive_logical_replication_slot
- 最滞后副本滞后量（因复制槽积压的WAL日志）：rds045_oldest_replication_slot_lag
- 事务日志生成速率：rds044_transaction_logs_generations

排查及解决方法

- **查看WAL日志大小是否异常并进行处理**

  首先通过查看监控rds040_transaction_logs_usage，观察WAL日志占用容量大小。观察WAL日志大小近期是否有明显上涨，或WAL日志占比磁盘容量较大。

  也可以通过下述SQL，查看WAL日志大小。如果发现WAL非常多，可以通过后续步骤依次排查。

  ```
  select pg_size_pretty(sum(size)) from pg_ls_waldir();
  ```

- **查看复制槽状态及延迟未清理的日志大小**

  复制槽会阻塞WAL的回收，查看监控指标inactive_logical_replication_slot如果不为0，且rds045_oldest_replication_slot_lag的堆积的WAL日志数据较大，甚至与rds040_transaction_logs_usage基本一致，表示复制槽阻塞WAL日志回收。

  或者通过下述SQL查询slot状态及WAL日志滞后量：

  ```
  select slot_name, active,
  pg_size_pretty(pg_wal_lsn_diff(b, a.restart_lsn)) as slot_latency
  from pg_replication_slots as a, pg_current_wal_lsn() as b;
  ```

  如果查询结果中，active字段存在'f' 状态的slot，并且slot_latency字段的大小很大，表示非活跃复制槽阻塞了WAL日志回收，导致WAL日志堆积无法清理。

  需要分析该复制槽是否还需要，如果不需要，建议执行删除slot命令的SQL：

  ```
  select pg_drop_replication_slot('slot_name');
  ```

- **查看WAL日志保留相关参数**

查看数据库参数“wal_keep_segments”、“wal_keep_size”、“max_wal_size”的值是否设置过大。

```
select name, setting from pg_settings where name in ('wal_keep_segments', 'wal_keep_size', 'max_wal_size');
```

- 对于RDS for PostgreSQL 12及以下版本，查看“wal_keep_segments”参数的值；对于12以上版本，查看“wal_keep_size”参数的值。
- WAL日志保留参数的值不宜太大，一般设置要小于磁盘总空间的10%；也不宜太小，一般要大于4GB，否则容易导致主库将备库需要的wal日志清理，进而导致备库异常。

- **查看写业务繁忙程度**

  可以通过rds044_transaction_logs_generations指标查看写业务繁忙程度，该指标表示平均每秒生成的事务日志（WAL日志）大小。

  如果该指标较大（平均超过50MB/s)，说明写业务较多，数据库内核会自动预留更多的WAL日志以便回收使用，WAL日志占用的磁盘空间会增加，建议通过磁盘扩容保证一定的磁盘冗余。

备份期间的 WAL 日志

如果WAL上升时，数据库处于备份状态，在备份完成后，WAL日志大小恢复正常，则表示备份期间WAL日志生成较快，导致备份速度赶不上生成速度。

可以调整备份时间，避免备份期间存在大量数据写入，或者扩大磁盘留足缓存容量。

14 批量更新、删除或插入数据

批量数据操作是一种优化手段，它能显著减少数据库与应用程序之间的频繁交互，从而降低系统开销并提升整体吞吐量。本文将结合具体代码示例，详细讲解如何高效实现数据的批量插入（Bulk Insert）、批量更新（Bulk Update）和批量删除（Bulk Delete），帮助开发者优化数据库性能。

创建示例表结构

```
CREATE TABLE product_inventory (
item_id INT PRIMARY KEY,
item_name VARCHAR(100),
stock_qty INT,
last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

批量数据插入方法

- **方法一：使用多行VALUES语法**

  一次性插入多条商品记录。

  ```
  INSERT INTO product_inventory (item_id, item_name, stock_qty)
  VALUES
      (101, '智能手表', 50),
      (102, '无线耳机', 120),
      (103, '蓝牙音箱', 75),
      (104, '平板电脑', 30);
  ```

- **方法二：结合SELECT生成序列数据**

  批量生成100条测试数据。

  ```
  INSERT INTO product_inventory (item_id, item_name, stock_qty)
  SELECT
      200 + n,
      '测试商品-' || n,
      (random() * 100)::INT
  FROM generate_series(1, 100) AS n;
  ```

- **方法三：事务包裹的多条INSERT**

  ```
  BEGIN;
  INSERT INTO product_inventory VALUES (301, '游戏手柄', 40);
  INSERT INTO product_inventory VALUES (302, '机械键盘', 60);
  INSERT INTO product_inventory VALUES (303, '电竞鼠标', 55);
  COMMIT;
  ```

- **方法四：使用COPY命令高效导入**

  COPY是PostgreSQL提供的一种高性能数据导入/导出协议，与传统的INSERT语句不同，它采用更精简的通信格式和传输机制，能够显著提升大批量数据的插入效率。具体优势包括：

- 协议优化：COPY使用二进制或文本流传输数据，避免了逐条解析SQL语句的开销。
- 性能对比：在插入数万行以上数据时，COPY通常比INSERT快10倍以上。
- 适用场景：适用于初始化数据、迁移或定期批量导入场景；支持从CSV、二进制文件或程序流中直接读取数据。

通过COPY命令从CSV格式导入的示例：

```
COPY product_inventory FROM STDIN WITH (FORMAT csv);
401, "4K显示器", 25
402, "曲面显示器", 18
403, "便携显示器", 32
\.
```

说明

不同的语言驱动对应的COPY接口不同，参考如下文档：

- **PostgreSQL JDBC Driver - API**
- **PostgreSQL 9.6.2 Documentation — COPY**

批量数据更新技术

- **使用临时表关联更新**

  创建临时更新数据表。

  ```
  WITH update_data(item_id, new_qty) AS (
      VALUES
          (101, 45),
          (102, 110),
          (104, 28)
  )
  -- 执行批量更新
  UPDATE product_inventory p
  SET stock_qty = u.new_qty
  FROM update_data u
  WHERE p.item_id = u.item_id;
  ```

- **基于CASE条件的批量更新**

  根据不同条件批量更新库存。

  ```
  UPDATE product_inventory
  SET stock_qty = CASE
      WHEN item_id = 201 THEN 90
      WHEN item_id = 202 THEN 65
      WHEN item_id = 203 THEN 40
      ELSE stock_qty
  END
  WHERE item_id IN (201, 202, 203);
  ```

批量数据删除方案

- **使用IN子句批量删除**

  删除指定ID的商品记录。

  ```
  DELETE FROM product_inventory
  WHERE item_id IN (301, 302, 303, 304);
  ```

- **通过子查询条件删除**

  删除库存量低于20的商品。

  ```
  DELETE FROM product_inventory
  WHERE stock_qty < 20;
  ```

- **使用TRUNCATE快速清空**

  清空整个表数据（不可回滚）。

```
TRUNCATE TABLE product_inventory;
```

性能优化建议

- **事务批处理**：将多个操作包裹在单个事务中。

```
BEGIN;
-- 批量操作1
-- 批量操作2
COMMIT;
```

- **批量大小控制**：建议每批处理1000~5000条记录。

15 使用事件触发器实现 DDL 回收站、防火墙、增量订阅同步

功能介绍

RDS for PostgreSQL开放事件触发器（Event Trigger），可以实现DDL回收站、DDL防火墙、DDL增量订阅同步等功能。灵活使用事件触发器可以减少维护成本，保护数据安全。

如果您对数据库安全有非常高的要求，可以通过PostgreSQL的事件触发器机制，实现DDL回收站和DDL防火墙功能，从多个维度保护数据安全。

本文提供一套基于PostgreSQL事件触发器实现DDL回收站、防火墙、增量订阅同步的最佳实践方案示例。

表 15-1 事件触发器功能说明

功能类型	作用	实现方式
事前防御	阻止危险DDL操作，例如：DROP TABLE、DROP INDEX、DROP DATABASE。	通过ddl_command_start事件触发器拦截未授权操作。
事后回档	在意外删除表后，支持从回收站恢复。	通过ddl_command_end和sql_drop事件触发器记录DDL操作。

DDL 回收站

通过使用pg_get_ddl_command和pg_get_ddl_drop这两个事件触发器收集并存入DDL语句到表ddl_recycle.ddl_log中，可以帮助您记录数据库DDL操作以便于事后进行回溯。

1. 创建专用模式和表。

```
CREATE SCHEMA ddl_recycle;

CREATE TABLE IF NOT EXISTS ddl_recycle.ddl_log (
    id SERIAL PRIMARY KEY,
    event_time TIMESTAMPTZ DEFAULT NOW(),
    username TEXT,
```

```
    database TEXT,
    client_addr INET,
    event TEXT,
    tag TEXT,
    object_type TEXT,
    schema_name TEXT,
    object_identity TEXT,
    command TEXT,
    is_dropped BOOLEAN DEFAULT FALSE
);
```

2. 创建事件触发器函数。

```
CREATE OR REPLACE FUNCTION ddl_recycle.log_ddl_command()
RETURNS event_trigger AS $$
DECLARE
    cmd TEXT;
    obj RECORD;
BEGIN
    SELECT query INTO cmd FROM pg_stat_activity WHERE pid = pg_backend_pid();

    IF TG_EVENT = 'ddl_command_end' THEN
        FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands() LOOP
            INSERT INTO ddl_recycle.ddl_log (
                username, database, client_addr, event, tag,
                object_type, schema_name, object_identity, command
            ) VALUES (
                current_user, current_database(), inet_client_addr(),
                TG_EVENT, TG_TAG, obj.object_type, obj.schema_name,
                obj.object_identity, cmd
            );
        END LOOP;
    ELSIF TG_EVENT = 'sql_drop' THEN
        FOR obj IN SELECT * FROM pg_event_trigger_dropped_objects() LOOP
            -- 标记原记录为已删除
            UPDATE ddl_recycle.ddl_log
            SET is_dropped = TRUE
            WHERE object_identity = obj.object_identity AND is_dropped = FALSE;

            -- 插入一条新记录，用来记录删除操作
            INSERT INTO ddl_recycle.ddl_log (
                username, database, client_addr, event, tag,
                object_type, schema_name, object_identity, command, is_dropped
            ) VALUES (
                current_user, current_database(), inet_client_addr(),
                TG_EVENT, TG_TAG, obj.object_type, obj.schema_name,
                obj.object_identity, cmd, TRUE
            );
        END LOOP;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

3. 注册事件触发器。

```
CREATE EVENT TRIGGER ddl_recycle_trigger
ON ddl_command_end
EXECUTE FUNCTION ddl_recycle.log_ddl_command();

CREATE EVENT TRIGGER ddl_drop_trigger
ON sql_drop
EXECUTE FUNCTION ddl_recycle.log_ddl_command();
```

执行完以上命令后，您的DDL语句就会记录在表test_ddl.tb_ddl_command中。

4. 执行一个DDL语句，查看是否能记录变更。

```
create table ddl_recycle.a(id int);
select * from ddl_recycle.ddl_log;
```

图 15-1 查看执行结果

```
test=> drop table ddl_recycle.a;
DROP TABLE
test=> select * from ddl_recycle.ddl_log;
 id |          event_time           | username | database | client_addr |      event      |     tag      | object_type |
 schema_name | object_identity |               command               | is_dropped
----+-------------------------------+----------+----------+-------------+-----------------+--------------+-------------+
-------------+-----------------+-------------------------------------+------------
  1 | 2025-07-31 11:46:22.078908+08 | test     | test     |             | ddl_command_end | CREATE TABLE | table       |
 ddl_recycle | ddl_recycle.a   | create table ddl_recycle.a(id int); | f
(1 row)
```

DDL 防火墙

您可以根据业务需求创建事件触发器，使用ddl_command_start事件类型可以阻止相应的DDL语句执行。

1. 创建防火墙规则表。

```
CREATE SCHEMA IF NOT EXISTS ddl_firewall;

CREATE TABLE IF NOT EXISTS ddl_firewall.rules (
    id SERIAL PRIMARY KEY,
    username TEXT,
    tag TEXT,
    pattern TEXT,
    action TEXT CHECK (action IN ('BLOCK', 'LOG')),
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

2. 创建防火墙触发器函数。

```
RETURNS event_trigger AS $$
DECLARE
    cmd TEXT;
    rule RECORD;
    is_blocked BOOLEAN := FALSE;
BEGIN
    SELECT query INTO cmd FROM pg_stat_activity WHERE pid = pg_backend_pid();

    FOR rule IN SELECT * FROM ddl_firewall.rules
                WHERE (username = current_user OR username = '*')
                  AND (tag = TG_TAG OR tag = '*')
                ORDER BY id
    LOOP
        IF cmd ~ rule.pattern THEN
            IF rule.action = 'BLOCK' THEN
                RAISE EXCEPTION 'DDL operation blocked by rule: %', rule.id;
            ELSIF rule.action = 'LOG' THEN
                INSERT INTO ddl_recycle.ddl_log (
                    username, database, client_addr, event, tag, command
                ) VALUES (
                    current_user, current_database(), inet_client_addr(),
                    'FIREWALL_LOG', TG_TAG, cmd
                );
            END IF;
        END IF;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

3. 注册事件触发器。

```
CREATE EVENT TRIGGER ddl_firewall_trigger
ON ddl_command_start
EXECUTE FUNCTION ddl_firewall.check_ddl();
```

4. 添加防火墙规则。

```
insert into ddl_firewall.rules values(1,'test', 'DROP TABLE', '', 'BLOCK');
```

5. 使用**test**用户执行删除表操作时会被拦截。

```
create table ddl_firewall.a(id int);
drop table ddl_firewall.a;
```

图 15-2 查看执行结果

```
test=> create table ddl_firewall.a(id int);
CREATE TABLE
test=> drop table ddl_firewall.a;
ERROR:  DDL operation blocked by rule: 1
CONTEXT:  PL/pgSQL function ddl_firewall.check_ddl() line 16 at RAISE
```

DDL 增量订阅同步

发布端将已经执行了的DDL语句存储在ddl_recycle.ddl_log中，订阅端可以读取记录进行数据同步。

1. 在发布端执行发布命令。
   ```
   CREATE PUBLICATION my_ddl_publication FOR TABLE ONLY ddl_recycle.ddl_log;
   ```
2. 在订阅端创建相同的表。
   ```
   CREATE SCHEMA ddl_recycle;
   CREATE TABLE IF NOT EXISTS ddl_recycle.ddl_log (
       id SERIAL PRIMARY KEY,
       event_time TIMESTAMPTZ DEFAULT NOW(),
       username TEXT,
       database TEXT,
       client_addr INET,
       event TEXT,
       tag TEXT,
       object_type TEXT,
       schema_name TEXT,
       object_identity TEXT,
       command TEXT,
       is_dropped BOOLEAN DEFAULT FALSE
   );
   ```
3. 在订阅端创建订阅。
   ```
   CREATE SUBSCRIPTION my_ddl_subscriptin CONNECTION 'host=*** port=*** user=*** password=***
   dbname=**' PUBLICATION my_ddl_publication;
   ```
4. 在订阅端针对ddl_recycle.ddl_log表创建相应的触发器，实现DDL增量同步。

16 使用 Replication Slot 创建数据订阅

功能介绍

PostgreSQL原生支持使用复制槽（Replication Slot）开启数据订阅（Change Data Capture，简称CDC）。Replication Slot是用于逻辑复制的核心机制，主要作用如下：

1. 确保主库的WAL日志（预写日志）不会被过早清理，直到从库或订阅者消费完日志。
2. 记录订阅者的消费位置（LSN），避免数据丢失或重复消费。
3. 支持通过逻辑解码插件（例如test_decoding和wal2json）将WAL日志解析为可读的变更数据（例如INSERT、UPDATE、DELETE操作），实现数据订阅。

本文介绍在RDS for PostgreSQL实例上开启数据订阅的配置方法。

前提条件

- 参考**自定义购买ECS**，已购买ECS。ECS选择与RDS for PostgreSQL实例相同的区域、VPC和安全组，便于RDS for PostgreSQL和ECS网络互通。
- 参考**购买并通过PostgreSQL客户端连接RDS for PostgreSQL实例**，已购买RDS for PostgreSQL实例并安装PostgreSQL客户端。
- 参考**通过内网连接RDS for PostgreSQL实例（Linux方式）**，已通过ECS连接到RDS for PostgreSQL实例。

注意事项

- 仅支持在主实例上开启数据订阅并消费，不支持只读实例。
- RDS for PostgreSQL支持Logical Replication Slot Failover，主备切换操作不会影响数据订阅。更多介绍，请参见**逻辑订阅故障转移（Failover Slot）**。
- 开启数据订阅前需要修改RDS for PostgreSQL实例参数，并重启RDS for PostgreSQL实例。请在业务低峰期进行修改，避免对业务造成影响。

开启数据订阅

步骤一：创建测试数据库

本示例以创建数据库**testdb**为例。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域。

步骤3 单击页面左上角的，选择“数据库 > 云数据库 RDS”，进入RDS信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击实例名称，进入实例的“概览”页。

步骤5 在左侧导航栏，单击“数据库管理”，单击“创建数据库”，在弹出框中填写数据库信息，单击“确定”。

图 16-1 创建数据库

----结束

步骤二：创建测试账号并配置权限

本示例创建的账号名仅为示例，您可以根据实际情况自定义名称。

1. 在账号管理页面，单击“创建账号”。
2. 在弹出框中，输入账号名称、密码、权限和备注，单击“确定”。
3. 创建数据订阅普通账号，例如**cdc_user**。
4. 使用**root**账号连接RDS for PostgreSQL实例。

```
psql -h <实例连接地址> -p 5432 -U root -d testdb
```

5. 执行如下命令，将**cdc_user**账号添加到Replication角色中，并查询修改结果。

```
ALTER USER cdc_user WITH REPLICATION;
SELECT rolreplication FROM  pg_roles WHERE rolname='cdc_user';
```

查询结果：

```
rolreplication
----------------
 t
(1 row)
```

6. 执行如下命令为**cdc_user**账号授权。

```
GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC to cdc_user;
```

步骤三：调整 RDS for PostgreSQL 实例参数

1. 执行如下命令，查询实例参数设置。

```
SELECT name,
       setting,
       short_desc,
       source
FROM pg_settings
WHERE name ='wal_level';
```

查询结果：

```
name                  | setting |                          short_desc                          |       source
----------------------+---------+--------------------------------------------------------------+--------------------
 wal_level            | replica | Sets the level of information written to the WAL.            |
configuration file
(1 rows)
```

wal_level参数控制WAL日志中记录的信息详细程度，其取值决定了WAL日志的用途（例如崩溃恢复、物理复制、逻辑复制等）。wal_level有如下三个主要取值，从低到高记录的信息依次增加：

- minimal：仅记录崩溃恢复必需的最基础信息，不支持任何复制（几乎不用于生产环境）。
- replica（默认值）：记录物理复制所需的完整信息，支持物理复制（例如主备同步，备实例通过重做WAL日志与主实例保持物理存储一致）和只读副本（Hot Standby）。
- logical：在replica级别的基础上，额外记录逻辑复制所需的元数据（例如行级变更的具体内容、表结构信息等），支持逻辑复制和变更数据捕获（CDC）。

2. 在实例管理页面，单击实例名称，进入实例的“概览”页。
3. 在左侧导航栏，选择“参数设置”。
4. 将wal_level参数取值修改为logical。

   修改实例参数的具体方法，请参见**修改RDS for PostgreSQL实例参数**。

   请在业务低峰期修改实例参数，避免对业务造成影响。
5. 修改实例参数并提交后，您需要重启RDS for PostgreSQL实例。

步骤四：创建 Logical Replication Slot

步骤三：调整RDS for PostgreSQL实例参数中修改了实例参数，请在RDS for PostgreSQL实例重启完成后，实例状态变为“正常”时再进行以下操作。

1. 使用**root**账号连接RDS for PostgreSQL实例。

```
psql -h <实例连接地址> -p 5432 -U root -d testdb
```

2. 执行如下命令，指定解码插件为test_decoding，创建一个名为cdc_replication_slot的Replication Slot。

```
SELECT * FROM pg_create_logical_replication_slot(
    'cdc_replication_slot',
    'test_decoding'
);
```

执行成功后，返回结果包含复制槽名称和初始LSN（日志序列号），如下所示：

```
    slot_name    |    lsn
 ----------------+-----------
 cdc_replication_slot | 0/16A00000
(1 row)
```

3. 执行以下命令查看所有复制槽，验证复制槽创建成功。

```
SELECT slot_name, plugin, active FROM pg_replication_slots;
```

若slot_name为cdc_replication_slot且plugin为test_decoding，表示复制槽创建成功。

```
slot_name            |     plugin     | slot_type | datoid | database | temporary | active | active_pid | xmin |
catalog_xmin | restart_lsn | confirmed_flush_lsn | wal_status | safe_wal_size  | two_phase
---------------------+---------------+-----------+--------+----------+-----------+--------+------------+------
+--------------+-------------+---------------------+------------+----------------+-----------
 cdc_replication_slot | test_decoding | logical   |  31721 | testdb   | f         | f      |            |      |
25523| 2/2500A1B0  | 2/25003882          | reserved   |               | f
(1 row)
```

步骤五：创建测试数据

执行以下命令，创建测试数据，模拟生产环境。

1. 创建测试表，用于后续生成变更数据。

```
CREATE TABLE public.tb_tests (
    id SERIAL PRIMARY KEY,
    product_name VARCHAR(50) NOT NULL,
    price NUMERIC(10,2) NOT NULL,
    create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. 插入数据。

```
INSERT INTO public.tb_tests (product_name, price) VALUES ('笔记本电脑', 5999.99);
INSERT INTO public.tb_tests (product_name, price) VALUES ('机械键盘', 299.99);
```

3. 更新数据。

```
UPDATE public.tb_tests SET price = 5899.99 WHERE id = 1;
```

4. 删除数据。

```
DELETE FROM public.tb_tests  WHERE id = 2;
```

步骤六：客户端读取数据

1. 使用cdc_user账号连接RDS for PostgreSQL实例。

```
psql -h <实例连接地址> -p 5432 -U cdc_user -d testdb
```

2. 执行以下命令，从Replication Slot中读取数据。

```
SELECT * FROM pg_logical_slot_peek_changes(
    'cdc_replication_slot',  -- 复制槽名称
    NULL,                    -- 起始LSN（NULL表示从当前位置开始）
    NULL,                    -- 最大返回条目数（NULL表示无限制）
  );
```

查询结果示例：

```
lsn          |  xid  |                                data
-------------+-------+-------------------------------------------------------------------------------
 0/16A01200 | 1234 | BEGIN 1234
 0/16A01200 | 1234 | table public.tb_tests: INSERT: id[integer]:1 product_name[character varying]:'笔记
本电脑' price[numeric(10,2)]:5999.99 create_time[timestamp with time zone]:'2024-05-20
10:00:00+08'
```

```
 0/16A01350 | 1234 | table public.tb_tests: INSERT: id[integer]:2 product_name[character varying]:'机械
键盘' price[numeric(10,2)]:299.99 create_time[timestamp with time zone]:'2024-05-20 10:01:00+08'
 0/16A014A0 | 1234 | COMMIT 1234
 0/16A015F0 | 1235 | BEGIN 1235
 0/16A015F0 | 1235 | table public.tb_tests: UPDATE: id[integer]:1 price[numeric(10,2)]:5899.99
(old:5999.99)
 0/16A016E0 | 1235 | COMMIT 1235
 0/16A017D0 | 1236 | BEGIN 1236
 0/16A017D0 | 1236 | table public.tb_tests: DELETE: id[integer]:2
 0/16A01860 | 1236 | COMMIT 1236
 (8 rows)
```

步骤七：客户端消费订阅数据

在客户端终端执行以下命令，连接数据库并从复制槽cdc_replication_slot消费数据。

```
pg_recvlogical \
  -h <PostgreSQL主机地址> \
  -p 5432 \
  -U cdc_user \
  -d testdb \
  --slot=cdc_replication_slot \
  --start \
  -f -
```

参数说明如下，请替换为实际值。

- -h：RDS for PostgreSQL主机地址
- -p：端口号，默认5432
- -U：订阅用户，需要具备REPLICATION权限。
- -d：数据库名
- --slot：复制槽名称
- --start：从当前位置开始消费，首次消费可省略，默认从起始位置。
- -f：输出到控制台，- 表示标准输出。

结果示例：

```
BEGIN 1234
table public.tb_tests: INSERT: id[integer]:1 product_name[character varying]:'笔记本电脑'
price[numeric(10,2)]:5999.99 create_time[timestamp with time zone]:'2024-05-20 10:00:00+08'
table public.tb_tests: INSERT: id[integer]:2 product_name[character varying]:'机械键盘'
price[numeric(10,2)]:299.99 create_time[timestamp with time zone]:'2024-05-20 10:01:00+08'
COMMIT 1234
BEGIN 1235
table public.tb_tests: UPDATE: id[integer]:1 price[numeric(10,2)]:5899.99 (old:5999.99)
COMMIT 1235
BEGIN 1236
table public.tb_tests: DELETE: id[integer]:2
COMMIT 1236
```

说明

- 消费特性：pg_recvlogical消费后，数据会被标记为“已处理”，复制槽的restart_lsn会更新到最新位置，后续无论是pg_recvlogical还是pg_logical_slot_peek_changes都不会再返回这些数据（避免重复消费）。
- 持续监听：命令会保持连接状态，若后续有新的变更（例如再次执行INSERT或UPDATE），会实时输出新的事务数据。
- 终止消费：按Ctrl+C可终止命令，下次执行时会从上次终止的位置继续消费（依赖复制槽记录的restart_lsn）。

关闭数据订阅

开启了数据订阅的RDS for PostgreSQL实例，需要使用更多的WAL日志存储空间，当不再需要数据订阅时，需通过以下步骤彻底关闭，避免复制槽持续占用WAL日志空间：

1. 使用**root**账号连接RDS for PostgreSQL实例。
   ```
   psql -h <实例连接地址> -p 5432 -U root -d testdb
   ```
2. （可选）确认复制槽存在。
   ```
   SELECT slot_name FROM pg_replication_slots WHERE slot_name = 'cdc_replication_slot';
   ```
3. 删除复制槽。该操作不可逆，请谨慎操作。
   ```
   SELECT pg_drop_replication_slot('cdc_replication_slot');
   ```
4. 验证数据订阅已关闭。
   ```
   SELECT slot_name FROM pg_replication_slots;
   ```
   如果查询结果中没有cdc_replication_slot，表示已关闭。

17 基于 Pgpool 实现读写分离

功能介绍

Pgpool是一个专为PostgreSQL设计的中间件，部署在数据库服务器与客户端之间，并且在前后台之间传递消息。 因此Pgpool对于服务器和客户端来说是透明的。通过提供连接池、复制、负载均衡、限制超过限度连接，显著提升数据库集群的性能、可用性和可扩展性。

本文介绍RDS for PostgreSQL实例及只读实例如何结合Pgpool实现读写分离。

前提条件

- 参考**自定义购买ECS**，已购买ECS。ECS选择与RDS for PostgreSQL实例相同的区域、VPC和安全组，便于RDS for PostgreSQL和ECS网络互通。
- 参考**购买并通过PostgreSQL客户端连接RDS for PostgreSQL实例**，已购买RDS for PostgreSQL实例并安装PostgreSQL客户端。
- 参考**创建只读实例**，已在RDS for PostgreSQL主实例上创建只读实例。
- 参考**通过内网连接RDS for PostgreSQL实例（Linux方式）**，已通过ECS连接到RDS for PostgreSQL实例。

操作步骤

步骤 1：安装 Pgpool

本文中ECS以CentOS 7为例，PostgreSQL版本以12版本为例。其他系统和数据库版本参考以下步骤替换相应命令。

登录到ECS上，执行以下命令安装PostgreSQL 12和Pgpool工具软件。

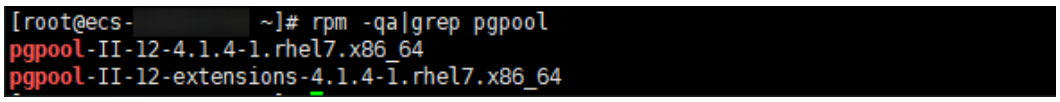
```
# 配置当前环境yum仓库
sudo yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
# 搜索postgresql包
sudo yum search all postgresql
# 搜索pgpool包
sudo yum search all pgpool
# 安装postgresql-12
sudo yum install -y postgresql12-server
# 安装pgpool-12
sudo yum install -y pgpool-II-12-extensions
```

步骤 2：配置 Pgpool

使用Pgpool实现负载均衡访问，所有认证发生在客户端和Pgpool之间，同时客户端仍然需要继续通过PostgreSQL的认证过程。

1. 查询Pgpool安装路径，命令如下：
```
rpm -qa | grep pgpool
```

图 17-1 执行结果



2. 在购买的RDS for PostgreSQL主实例上执行以下SQL，创建业务用户。

   创建业务用户。
```
create role digoal login encrypted password 'xxxxxxx';
create database digoal owner digoal;
```
   创建Pgpool数据库健康心跳用户，检查只读实例回放延迟（wal replay）。

   只要能登录postgres数据库或指定的库即可，配合Pgpool参数使用。
```
create role nobody login encrypted password 'xxxxxxx';
```
   xxxxxxx：新建数据库用户密码。

3. 修改配置文件pgpool.conf。
```
cd /etc/pgpool-II-12/

cp pgpool.conf.sample-stream pgpool.conf

vi pgpool.conf
```
   - /etc/pgpool-II-12/：安装Pgpool的目录。
   - pgpool.conf.sample-stream：Pgpool针对流复制模式提供的默认配置文件模板。
   - pgpool.conf：Pgpool配置文件。

4. 按 i 键进入编辑模式，主要结合日志对进行以下项进行修改，其他配置信息可以根据pgpool.conf文件注释选择。
```
listen_addresses = '*'

backend_hostname0 = '主实例IP地址'
backend_port0 = 5432
backend_flag0 = 'ALWAYS_MASTER'

backend_hostname1 = '只读实例IP地址'
backend_port1 = 5432
backend_flag1 = 'ALLOW_TO_FAILOVER'

enable_pool_hba = on

pid_file_name = '/var/run/pgpool-II-12/pgpool.pid'
```
   - listen_addresses：Pgpool服务监听的网络IP地址，这里选择的是监听所有IP。
   - backend_hostname：连接到后端PostgreSQL数据库节点的网络IP地址。
   - backend_port：连接到后端PostgreSQL数据库节点的网络端口号。
   - backend_flag：Pgpool控制节点行为。

5. 使用md5模式，配置pool_passwd密码文件。
```
pg_md5 --md5auth --username=digoal "xxxxxxx"
pg_md5 --md5auth --username=nobody "xxxxxxx"
```
   生成digoal和nobody用户密码，并自动写入pool_passwd文件。

6. 检查自动生成的pool_passwd文件。

```
cd /etc/pgpool-II-12

cat pool_passwd
```

pool_passwd：Pgpool存储PostgreSQL用户的加密密码的配置文件。

图 17-2 执行结果

```
[root@ecs-          pgpool-II-12]# cat pool_passwd
digoal:md5e8318d48f545394c4e67941aae8df253
nobody:md5ca14b3976cc05f036a450f0bd864b050
```

7. 配置pgpool_hba文件。

```
cd /etc/pgpool-II-12
cp pool_hba.conf.sample pool_hba.conf
vi pool_hba.conf

# 按i进入编辑模式
# 在pool_hba.conf文件中添加以下内容
host all all 0.0.0.0/0 md5
```

- pool_hba.conf.sample：客户端连接认证规则的配置文件模板。
- pool_hba.conf：客户端连接Pgpool时的认证规则文件。

图 17-3 执行结果

```
# "local" is for Unix domain socket connections only
local   all         all                               trust
# IPv4 local connections:
host    all         all         127.0.0.1/32          trust
host    all         all         ::1/128               trust
host all all 0.0.0.0/0 md5
```

8. 配置PCP管理密码文件。

该操作是用来管理Pgpool的密码和用户，不是数据库的用户和密码。

```
cd /etc/pgpool-II-12

pg_md5 abc  # 例如密码是abc
900150983cd24fb0d6963f7d28e17f72

cp pcp.conf.sample pcp.conf

vi pcp.conf

# 按i进入编辑模式
# 在pcp.conf文件中添加以下内容
USERID:MD5PASSWD
manage:900150983cd24fb0d6963f7d28e17f72  #表示使用manage用户来管理PCP密码文件
```

- pcp.conf.sample：配置PCP管理接口认证的模板文件。
- pcp.conf：PCP管理接口认证的文件，定义管理员远程管理Pgpool的用户名和加密密码。

步骤 3：启动 Pgpool 并验证读写分离

1. 启动Pgpool。

```
cd /etc/pgpool-II-12

pgpool -f ./pgpool.conf -a ./pool_hba.conf -F ./pcp.conf
```

2. 通过Pgpool连接数据库。
```
psql -h 127.0.0.1 -p 9999 -U digoal -d postgres
```
   - 127.0.0.1：使用pgpool.conf配置文件中listen_addresses配置的IP地址。
   - 9999：pgpool.conf中配置默认监听端口号。

图 17-4 执行结果

```
[root@ecs-          pgpool-II-12]# psql -h 127.0.0.1 -p 9999 -U digoal -d postgres
psql (12.22, server 16.8)
WARNING: psql major version 12, server major version 16.
         Some psql features might not work.
Type "help" for help.

postgres=>
```

3. 查看当前数据库Pgpool集群状态信息。
```
show pool_nodes;
```

图 17-5 执行结果

```
postgres=> show pool_nodes;
 node_id |   hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node | replication_delay | replication_state | replication_sync_state | last_status_change
---------+---------------+------+--------+-----------+---------+------------+-------------------+-------------------+-------------------+------------------------+---------------------
 0       | 192.168.0.46  | 5432 | up     | 0.500000  | primary | 1          | false             | 0                 |                   |                        | 2025-07-09 11:27:57
 1       | 192.168.0.117 | 5432 | up     | 0.500000  | standby | 1          | true              | 0                 |                   |                        | 2025-07-09 11:27:57
(2 rows)
```

4. 在数据库上执行以下命令，然后断开重连，再次查询pg_is_in_recovery()，如果交替返回t和f，说明是交替将请求发送了给主库和只读库，即读写分离成功。
```
SELECT pg_is_in_recovery();
```
   pg_is_in_recovery()：检测数据库实例当前是否处于恢复模式。返回t，表示当前是备库；返回f，表示当前是主库。

图 17-6 执行结果

```
postgres=> SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 row)

postgres=> \q
[root@ecs-          pgpool-II-12]# psql -h 127.0.0.1 -p 9999 -U digoal -d postgres
psql (12.22, server 16.8)
WARNING: psql major version 12, server major version 16.
         Some psql features might not work.
Type "help" for help.

postgres=> SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 f
(1 row)

postgres=> \q
[root@ecs-          pgpool-II-12]# psql -h 127.0.0.1 -p 9999 -U digoal -d postgres
psql (12.22, server 16.8)
WARNING: psql major version 12, server major version 16.
         Some psql features might not work.
Type "help" for help.

postgres=> SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 f
(1 row)
```

常见问题

如何停止、重新加载 Pgpool 配置?

您可以使用pgpool --help查询帮助命令，例如：

```
cd /etc/pgpool-II-12

pgpool -f ./pgpool.conf -m fast stop
```

如果有多个只读实例，应该如何配置 pgpool.conf 文件?

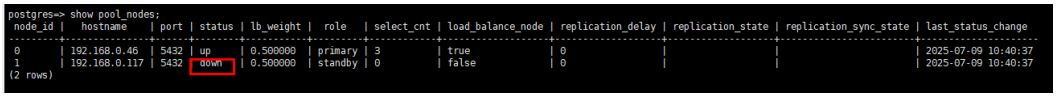
根据pgpool.conf文件配置字段样式，持续增加新的配置信息，例如：

```
backend_hostname1 = 'xx.xx.xxx.xx'
backend_port1 = 5432
backend_weight1 = 1
backend_flag1 = 'ALLOW_TO_FAILOVER'
backend_application_name1 = 'server1'

backend_hostname2 = 'xx.xx.xx.xx'
backend_port2 = 5432
backend_weight2 = 1
backend_flag2 = 'ALLOW_TO_FAILOVER'
backend_application_name1 = 'server2'
```

查询 pg_is_in_recovery()时，一直返回 t 或 f 怎么办?

如果一直返回t，表示Pgpool只连接到了从库；如果一直返回f，表示Pgpool只连接到了主库。同时，执行**show pool_nodes;**始终存在异常节点。

图 17-7 执行结果



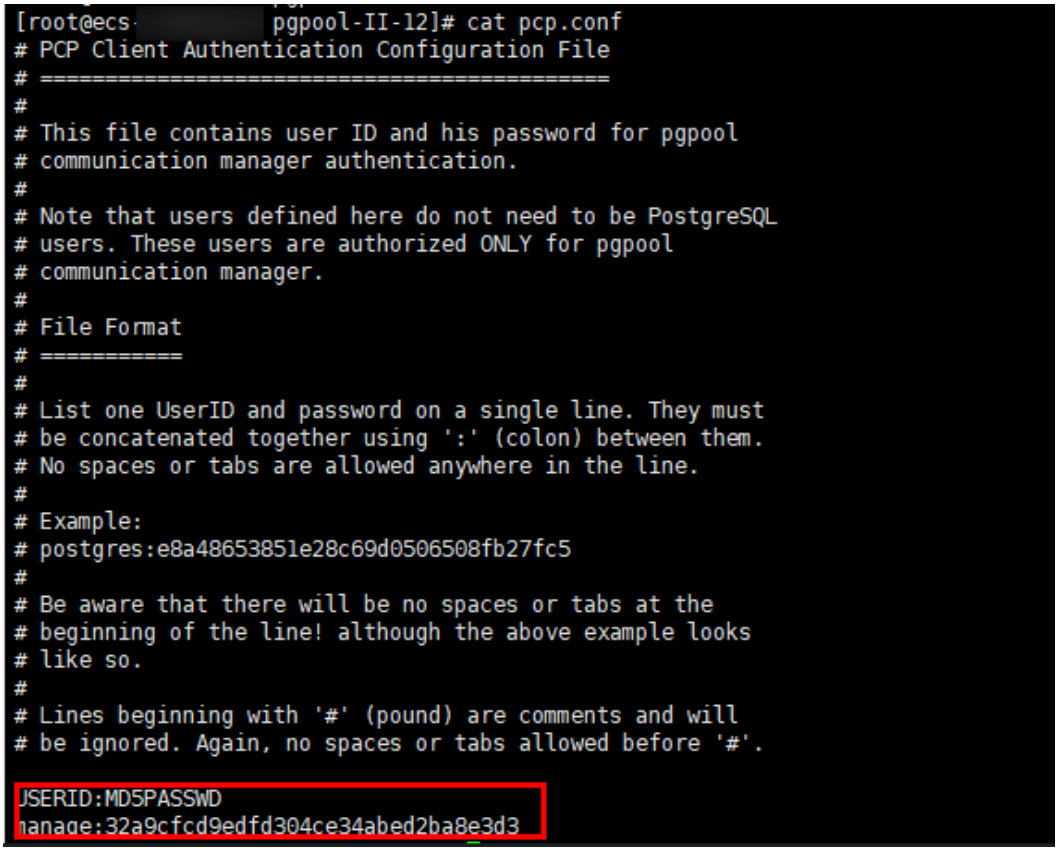
可以通过清理pgpool_status配置文件并重启尝试，例如：

```
cd /etc/pgpool-II-12

# 暂停服务
pgpool -f ./pgpool.conf -m fast stop

# 删除配置文件
rm /tmp/pgpool_status

# 重启服务
pgpool -f ./pgpool.conf -a ./pool_hba.conf -F ./pcp.conf
```

如果还是存在异常，可以尝试pcp_attach_node重新注册主库或从库。
pcp_attach_node工具是Pgpool自带的集群管理工具，pcp_attach_node工具可以将集群中的节点重新注册到集群中。

- 查看PCP工具的连接用户。
  ```
  cd /etc/pgpool-II-12

  cat pcp.conf
  ```

图 17-8 执行结果



- 使用pcp_attach_node重新注册。

  ```
  pcp_attach_node -h 本地IP -U pcp_user node_id
  ```

  例如：

  ```
  pcp_attach_node –h 127.0.0.1 –U manage 0
  ```

18 用户喜好推荐系统

使用场景

推荐系统广泛应用于各类互联网平台和传统行业，通过分析用户行为、偏好和上下文信息，为用户提供个性化内容推荐。常见的场景有：

- 电子商务与零售：根据用户浏览、购买历史推荐相关商品（如“猜你喜欢”）。
- 流媒体与内容平台：基于观看记录、评分推荐内容（如电影、短视频等）。
- 旅游服务：基于用户偏好推荐目的地。
- 应用推荐：根据用户下载和使用应用习惯，推荐应用。

本文以电影为例，介绍如何设计推荐系统数据库。

前提条件

已安装**pg_trgm插件**。

设计实现

PostgreSQL的pg_trgm扩展提供了基于三元组的字符串相似度计算功能，可以结合该插件的相似度计算功能来搭建简易的用户推荐系统。

1. 连接RDS for PostgreSQL实例。
2. 在实例上创建测试表。

```
CREATE TABLE movies (
    id INT,                    -- 电影ID
    title VARCHAR(255),        -- 电影名称
    description TEXT,          -- 电影描述
    genres VARCHAR(255)[],     -- 电影类型
    year INTEGER               -- 电影年份
);
```

3. 查找与特定电影标题相似的电影。

```
-- 创建GIN索引加速相似度查询
CREATE INDEX movies_title_gin_idx ON movies USING gin(title gin_trgm_ops);

SELECT
    m2.id,
    m2.title,
    similarity(m1.title, m2.title) AS similarity_score
FROM
    movies m1,
    movies m2
```

```
WHERE
    m1.id = 123 AND                    -- 目标电影ID
    m1.id != m2.id AND
    similarity(m1.title, m2.title) > 0.3  -- 相似度阈值
ORDER BY
    similarity_score DESC
LIMIT 10;
```

4. 基于电影描述相似度搜索电影。

```
-- 为描述创建GIN索引
CREATE INDEX movies_description_gin_idx ON movies USING gin(description gin_trgm_ops);

SELECT
    m2.id,
    m2.title,
    similarity(m1.description, m2.description) AS similarity_score
FROM
    movies m1,
    movies m2
WHERE
    m1.id = 123 AND
    m1.id != m2.id AND
    similarity(m1.description, m2.description) > 0.1  -- 描述相似度阈值可以设置较低
ORDER BY
    similarity_score DESC
LIMIT 10;
```

5. 结合标题、描述以及类型等多个特征搜索电影。

```
SELECT
    m2.id,
    m2.title,
    (
        0.5 * similarity(m1.title, m2.title) +
        0.3 * similarity(m1.description, m2.description) +
        0.2 * (SELECT COUNT(*) FROM unnest(m1.genres) AS g1
                JOIN unnest(m2.genres) AS g2 ON g1 = g2)::FLOAT /
              GREATEST(array_length(m1.genres, 1), array_length(m2.genres, 1))
    ) AS combined_similarity
FROM
    movies m1,
    movies m2
WHERE
    m1.id = 123 AND
    m1.id != m2.id
ORDER BY
    combined_similarity DESC
LIMIT 10;
```

6. 可以根据用户配置来限制搜索范围。例如执行以下SQL只搜索同类型的电影。

```
SELECT m2.id, m2.title, similarity(m1.title, m2.title) AS similarity_score
FROM movies m1, movies m2
WHERE m1.id = 123 AND
      m1.id != m2.id AND
      m1.genres && m2.genres AND  -- 至少有一个共同类型
      similarity(m1.title, m2.title) > 0.3
ORDER BY similarity_score DESC
LIMIT 10;
```


19 RDS for PostgreSQL 指标告警配置建议

通过在云监控服务界面设置告警规则，用户可自定义监控目标与通知策略，及时了解实例的运行状况，从而起到预警作用。本章节介绍了设置RDS for PostgreSQL指标告警规则的配置及建议。

创建指标告警规则

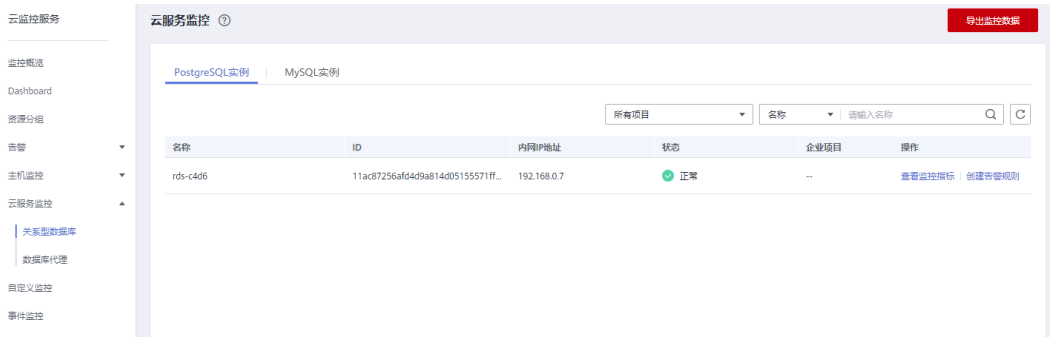
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在“服务列表”中，选择“管理与监管 > 云监控服务”，进入“云监控”服务信息页面。

步骤4 在左侧导航栏选择“云服务监控 > 关系型数据库”。

图 19-1 选择监控对象



步骤5 选择需要添加告警规则的实例，单击操作列的“创建告警规则”。

图 19-2 创建告警规则

步骤6 在“创建告警规则”页面，填选相关信息。

表 19-1 告警规则信息

参数	参数说明
名称	系统会随机产生一个名称，用户也可以进行修改。
描述	告警规则描述。
触发规则	根据需要可选择关联模板、导入已有模板或自定义创建。 **说明** 选择关联模板后，所关联模板内容修改后，该告警规则中所包含策略也会跟随修改。 建议选择导入已有模板，模板中已经包含CPU使用率、内存使用率、磁盘利用率三个常用告警指标。
模板	选择需要导入的模板。 您可以选择系统预置的默认告警模板，或者选择自定义模板。
告警策略	触发告警规则的告警策略。 是否触发告警取决于连续周期的数据是否达到阈值。例如CPU使用率监控周期为5分钟，连续三个周期平均值≥80%，则触发告警。 **说明** 告警规则内最多可添加50条告警策略，若其中一条告警策略达到条件都会触发告警。
告警级别	根据告警的严重程度不同等级，可选择紧急、重要、次要、提示。

图 19-3 设置告警通知

表 19-2 告警通知

参数	参数说明
发送通知	配置是否发送邮件、短信、HTTP和HTTPS通知用户。
通知方式	根据需要可选择通知组或主题订阅两种方式。
通知组	需要发送告警通知的通知组。
通知对象	选择主题订阅时设置需要发送告警通知的对象，可选择云账号联系人或主题名称。 ● 云账号联系人为注册时的手机和邮箱。 ● 主题是消息发布或客户端订阅通知的特定事件类型。
生效时间	该告警仅在生效时间段发送通知消息，非生效时段则在隔日生效时段发送通知消息。 如生效时间为08:00-20:00，则该告警规则仅在08:00-20:00发送通知消息。
触发条件	可以选择“出现告警”、“恢复正常”两种状态，作为触发告警通知的条件。
归属企业项目	告警规则所属的企业项目。只有拥有该企业项目权限的用户才可以查看和管理该告警规则。
标签	标签由键值对组成，用于标识云资源，可对云资源进行分类和搜索。

步骤7 单击“立即创建”，告警规则创建完成。

关于告警参数的配置，请参见《**云监控用户指南**》。

----结束

指标告警配置建议

表 19-3 RDS for PostgreSQL 指标告警配置建议

指标ID	指标名称	指标含义	最佳实践阈值	最佳实践告警级别	告警后的处理建议
rds001_cpu_util	CPU使用率	该指标用于统计测量对象的CPU使用率，以比率为单位。	连续3个周期原始值 > 80 %	重要	1. 建议参考**CPU使用率高问题定位及处理方法**排查及处理。 2. 因为业务增量导致CPU持续保持高位，建议升配规格，参考**变更实例的CPU和内存规格**。
rds002_mem_util	内存使用率	该指标用于统计测量对象的内存使用率，以比率为单位。	连续3个周期原始值 > 90 %	重要	1. 建议参考**内存使用率高问题定位及处理方法**排查及处理。 2. 因为业务增量导致内存持续保持高位，建议升配规格，参考**变更实例的CPU和内存规格**。
rds039_disk_util	磁盘利用率	该指标用于统计测量对象的磁盘利用率，以比率为单位。	连续3个周期原始值 > 80 %	重要	1. 建议参考**磁盘使用率高问题定位及处理方法**排查及处理。 2. 因为业务增量导致磁盘使用率持续保持高位，建议根据业务使用情况进行扩容，参考**手动变更磁盘容量**。

指标ID	指标名称	指标含义	最佳实践阈值	最佳实践告警级别	告警后的处理建议
rds045_oldest_replication_slot_lag	最滞后副本滞后量	多个副本中最滞后副本（依据接收到的WAL数据）滞后量。	连续1个周期原始值 > 20480 MB	重要	建议参考**最滞后副本滞后量和复制时延高问题定位及处理方法**排查及处理。
rds046_replication_lag	复制时延	副本滞后时延。	连续3个周期原始值 > 600 s	重要	
rds083_conn_usage	连接数使用率	该指标用于统计当前已用的PgSQL连接数占总连接数的百分比。	连续3个周期原始值 > 80 %	重要	1. 建议排查连接数增长的业务影响，及时排查业务侧连接是否有效，优化实例连接，释放不必要的连接。参考**RDS for PostgreSQL数据库连接数满的排查思路**。 2. 关于连接数设置建议，参考**RDS for PostgreSQL数据库实例支持的最大数据连接数是多少**。
active_connections	活跃连接数	该指标为统计数据库当前活跃连接数。	连续1个周期原始值 > [当前CPU核数*2] Counts	重要	建议参考**连接数和活跃连接数异常情况定位及处理方法**排查及处理。
oldest_transaction_duration	最长事务存活时长	该指标为统计当前数据库中存在的最长事务存活时长。	根据业务情况来配置。参考值：连续1个周期原始值 > 7200000 ms	重要	建议参考**长事务问题定位及处理方法**排查及处理。

指标ID	指标名称	指标含义	最佳实践阈值	最佳实践告警级别	告警后的处理建议
oldest_transaction_duration_2pc	最长未决事务存活时长	该指标为统计当前数据库存在的最长未决事务存活时长。	根据业务情况来配置。参考值：连续1个周期原始值 > 7200000 ms	重要	
db_max_age	最大数据库年龄	该指标为统计当前数据库的最大数据库年龄(获取表pg_database中max(age(datfrozenxid))值)。	连续1个周期原始值 > 1000000000	重要	建议参考**数据库年龄增长问题定位及处理方法**排查及处理。
slow_sql_three_second	已执行3s的SQL数	该指标为统计数据库执行时长3秒以上的慢SQL个数。 该指标为采集时刻的瞬时值，并不是一分钟内的累计值。	根据业务情况来配置。参考值：连续1个周期原始值 > [当前CPU核数*2] Counts	重要	建议参考**已执行3s或5s SQL数问题定位及处理方法**排查及处理。
slow_sql_five_second	已执行5s的SQL数	该指标为统计数据库执行时长5秒以上的慢SQL个数。 该指标为采集时刻的瞬时值，并不是一分钟内的累计值。	根据业务情况来配置。参考值：连续1个周期原始值 > [当前CPU核数*2] Counts	重要	
inactive_logical_replication_slot	非活跃逻辑复制槽数量	该指标用于统计当前数据库中存在的非活跃逻辑复制槽数量。	连续3个周期原始值 > 1 Counts	重要	建议参考**存在非活跃逻辑复制槽问题定位及处理方法**排查及处理。

20 RDS for PostgreSQL 安全最佳实践

PostgreSQL数据库在可靠性、稳定性、数据一致性等获得了业内极高的声誉，已成为许多企业的首选开源关系数据库，业界简称PG。RDS for PostgreSQL是一种基于云计算平台的即开即用、稳定可靠、弹性伸缩、便捷管理的在线云数据库服务。

为加强RDS for PostgreSQL数据库安全性，本文将从以下几个维度给出建议，您可以根据业务需要在本指导的基础上进行安全配置。

- **配置数据库的最大连接数**
- **配置客户端认证超时时间**
- **配置SSL连接和加密算法**
- **配置密码加密功能**
- **配置合理的pg_hba规则**
- **配置服务器拒绝带反斜杠转义的引号**
- **定期检查并删除业务不再使用的角色**
- **建议回收public模式的所有权限**
- **设置合理的用户角色密码有效期**
- **配置日志级别记录发生错误的SQL语句**
- **确保数据库账号的最低权限**
- **开启备份功能**
- **开启数据库审计功能**
- **避免绑定EIP直接通过公网访问RDS for PostgreSQL**
- **数据库版本更新到最新版本**
- **配置账号认证失败延迟时间**

配置数据库的最大连接数

max_connections 决定了数据库的最大并发连接数。增加这个参数值可能引起RDS for PostgreSQL请求更多的System V共享内存或者信号量，会导致超出操作系统默认配置允许的值。请根据业务的复杂度，合理配置max_connections，具体可参考**实例使用规范**。

配置客户端认证超时时间

authentication_timeout控制完成客户端认证的时间上限，单位是秒。该参数可以防止客户端长时间占用连接通道，默认是60s。如果在指定的时间内没有完成认证，连接将被强制关闭。该超时时间的配置是为了增强PostgreSQL的安全性。

配置 SSL 连接和加密算法

尽可能利用SSL进行TCP/IP连接，使用SSL加密通信可确保客户端和服务器之间的所有通信都经过加密，防止数据被泄露和篡改，确保数据的完整性。在设置SSL加密时，服务端需要配置安全的TLS协议和加密算法，推荐使用TLSv1.2协议，加密算法推荐使用EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EDH+aRSA+AESGCM:EDH+aDSS+AESGCM:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!SRP:!RC4，具体请参考**SSL连接**。

可通过修改参数“ssl_min_protocol_version”配置TLS协议，修改参数“ssl_ciphers”配置加密算法。

配置密码加密功能

密码必须要加密。使用**CREATE USER**或者**ALTER ROLE**修改密码时候，默认使用加密的方式，推荐使用scram-sha-256，可通过修改参数“password_encryption”进行配置。

MD5选项仅供与低版本兼容场景使用，新建数据库实例默认使用scram-sha-256。

须知

参数“password_encryption”修改后需要重置密码后才能生效。

配置合理的 pg_hba 规则

为了确保PostgreSQL数据库的安全性，配置合理的pg_hba规则至关重要。建议仅允许必要的用户和主机访问数据库，限制特定IP地址或子网的访问，定期审查和更新pg_hba文件以确保规则符合当前的业务需要。具体配置方法请参考**修改pg_hba配置**。

配置服务器拒绝带反斜杠转义的引号

参数“backslash_quote”控制字符串里面引号是否可以被\'代替。推荐使用SQL标准方法，表示一个引号是写两遍 ('')，如果客户端代码不能正确的转义，可能发生SQL注入攻击。建议配置参数“backslash_quote”值为“safe_encoding”，拒绝带反斜杠转义的引号的查询，可以避免SQL注入的风险。

定期检查并删除业务不再使用的角色

对于每个查询出来的角色，检查是否必须存在，任何未知的角色都需要被审视，确保每个角色都是正常使用的，否则删除这些角色。可通过如下命令进行查询：

SELECT rolname FROM pg_roles;

建议回收 public 模式的所有权限

public模式是默认的模式，所有用户都可以访问其中的对象，包括表、函数、视图等。如果public模式中的对象被授予了权限，那么所有用户都可以访问这些对象，这可能会导致安全漏洞。root用户可通过如下命令回收权限：

revoke all on schema public from public;

设置合理的用户角色密码有效期

当创建角色时候，使用**VALID UNTIL**关键字设置过多长时间后角色的密码不再有效。如果这个关键字被忽略，密码会长期有效。建议定期更改密码，例如每三个月更改一次密码。可通过如下命令进行设置：

CREATE ROLE name WITH PASSWORD *'password'* **VALID UNTIL 'timestamp';**

检查是否设置密码有效期：

SELECT rolname,rolvaliduntil FROM pg_roles WHERE rolsuper = false AND rolvaliduntil IS NULL;

配置日志级别记录发生错误的 SQL 语句

参数“log_min_error_statement”控制哪些引起错误的SQL语句记录到服务器日志中。对于大于等于当前配置等级的SQL语句消息，会记录到日志里。有效的值包括debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal, panic。“log_min_error_statement”至少配置为“error”，具体可参考**日志配置管理**。

确保数据库账号的最低权限

RDS for PostgreSQL支持“基于角色”的方法授予账号对数据和命令的访问权限。建议管理员结合业务需要，遵从最低授权原则，创建合适的**数据库账号**，对账号进行授权。如果发现存在不符合该角色的账号权限，请结合业务需要，对账号权限进行更新或者**删除**。由于PostgreSQL存在一些**内置账号**，用于给数据库实例提供完善的后台运维管理服务，禁止用户使用和删除。

开启备份功能

创建云数据库RDS实例时，系统默认开启自动备份策略，默认自动备份保留7天，可根据业务需要调整备份保留时长。RDS for PostgreSQL实例支持**自动备份**和**手动备份**，您可以定期对数据库进行备份，当数据库故障或数据损坏时，可以**通过备份文件恢复数据库**，从而保证数据可靠性，详情请参见**数据备份**。

开启数据库审计功能

通过将PostgreSQL审计扩展（pgAudit）与RDS for PostgreSQL数据库实例一起使用，可以捕获审计员通常需要或满足法规要求的详细记录。例如，您可以设置pgAudit扩展来跟踪对特定数据库和表所做的更改、记录进行更改的用户以及许多其他详细信息。pgAudit默认不开启，根据业务需要开启插件。具体配置可参考**使用pgAudit插件**。

避免绑定 EIP 直接通过公网访问 RDS for PostgreSQL

避免RDS for PostgreSQL部署在公网或者DMZ里，应该将RDS for PostgreSQL部署在华为云内部网络，使用路由器或者防火墙技术把RDS for PostgreSQL保护起来，避免

直接绑定EIP方式从公网访问RDS for PostgreSQL。通过这种方式防止未授权的访问及DDos攻击等。建议解绑弹性公网IP，如果您的业务必须绑定EIP，请务必通过设置**安全组规则**限制访问数据库的源IP。

数据库版本更新到最新版本

对于PostgreSQL社区停止维护的版本，云上RDS for PostgreSQL服务也会相应发布**产品生命周期**。使用较老的版本可能存在安全风险，运行最新版本的软件可以避免受到某些攻击。如果业务需要，可通过**升级内核小版本**或者**使用转储与还原升级大版本**。

配置账号认证失败延迟时间

PostgreSQL数据库默认内置了auth_delay插件，auth_delay会使服务器在返回认证失败之前短暂停止，使得暴力破解数据库密码更难。可通过**修改RDS for PostgreSQL实例参数**设置auth_delay.milliseconds参数（该参数为返回认证失败之前等待的毫秒数）延迟账号登录认证失败的等待时间，缺省值是3000。