ALIBABA CLOUD

阿里云

云原生分布式数据库 PolarDB-X

用户指南1.0

文档版本：20210809

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.实例管理 08
1.1. 实例介绍 08
1.2. 创建实例 08
1.3. 实例变配 10
1.4. 升级版本 10
1.5. 参数设置 12
1.6. 释放实例 15
2.私有RDS管理 16
2.1. 概述 16
2.2. 查看私有RDS实例信息 16
2.3. 申请或释放外网地址 17
2.4. 设置可维护时间段 19
2.5. 创建高权限账号 20
2.6. 参数设置 21
2.7. 主备切换 22
2.8. 切换可用区 23
2.9. 设置白名单 24
3.只读实例 27
3.1. 实例介绍 27
3.2. 创建实例 28
3.3. 管理实例 29
4.账号管理 32
5.监控与报警 36
5.1. 实例监控 36
5.2. 数据库监控 37
5.3. 存储监控 38

5.4. 管理报警规则 40
6.切换可用区 42
7.SQL审计与分析 43
7.1. 简介 43
7.2. 开启 SQL 日志审计 46
7.3. 日志字段 48
7.4. 日志分析 49
7.5. 日志报表 56
7.6. 高级管理 61
7.7. 为子账号授予SQL审计权限 63
8.数据库管理 76
8.1. 概述 76
8.2. 查看数据库信息 76
8.3. 创建数据库 77
9.网络与连接 83
9.1. 连接数据库 83
9.2. 修复分库连接 85
9.3. 外网访问 85
10.扩容管理 88
10.1. 平滑扩容 88
10.2. 热点扩容 88
11.存储管理 89
12.数据表管理 91
12.1. 调整拆分键 91
12.2. 关闭全表扫描 92
12.3. 查看表结构 93
12.4. 删除数据表 94
13.设置白名单 95

14.分布式事务 96
14.1. 基于 MySQL 5.7 的分布式事务 96
14.2. 基于 MySQL 5.6 的分布式事务 96
15.数据导入导出 98
15.1. 数据迁移或同步方案概览 98
15.2. 使用mysqldump导入导出数据 99
15.3. 使用程序进行大数据导入 101
15.4. 通过数据集成导入导出数据 103
16.自定义控制指令 117
16.1. 概览 117
16.2. 帮助语句 117
16.3. 查看规则和节点拓扑类语句 118
16.4. SQL 调优类语句 124
16.5. 统计信息查询类语句 130
16.6. SHOW PROCESSLIST指令与KILL指令 134
16.7. 老版本 SHOW PROCESSLIST 指令与 KILL 指令 137
16.8. 回收站管理 139
17.自定义HINT 140
17.1. HINT简介 140
17.2. 自定义SQL超时时间 142
17.3. 扫描全部或部分分库分表 143
17.4. 指定分库执行 SQL 144
17.5. 读写分离 147
18.SQL调优指南 149
18.1. 查询慢SQL明细 149
18.2. SQL调优基础概念 150
18.3. SQL调优方法与实战 151
18.4. SQL调优进阶 154

18.4.1. 查询优化器介绍 154
18.4.2. 查询改写与下推 157
18.4.3. 查询执行器介绍 161
18.4.4. 执行计划和基本算子 165
18.4.5. JOIN与子查询的优化和执行 169
18.4.6. 优化聚合与排序 174
18.4.7. 执行计划管理 178
18.5. 智能索引推荐 186
19.系统运维 190
19.1. 高危类SQL自动保护 190
19.2. 待处理事件 190
20.全局二级索引 192
20.1. 使用全局二级索引 192
20.2. 使用全局二级索引时的注意事项 194
21.访问控制 197
21.1. 注册和登录阿里云账号 197
21.2. RAM在PolarDB-X中的应用 198
21.3. 激活PolarDB-X访问RDS服务授权 199
21.4. 通过自定义策略授权RAM用户管理PolarDB-XDRDS 201
21.5. RAM资源授权 205

1.实例管理

1.1. 实例介绍

PolarDB-X 1.0由计算层与存储层资源构成，计算层在物理资源上由多个计算节点所组成的分布式集群，业务数据存储在下挂的多个私有定制RDS实例中。目前售卖的PolarDB-X 1.0计算资源实例皆为专享实例，其中包括主实例和只读实例。

主实例

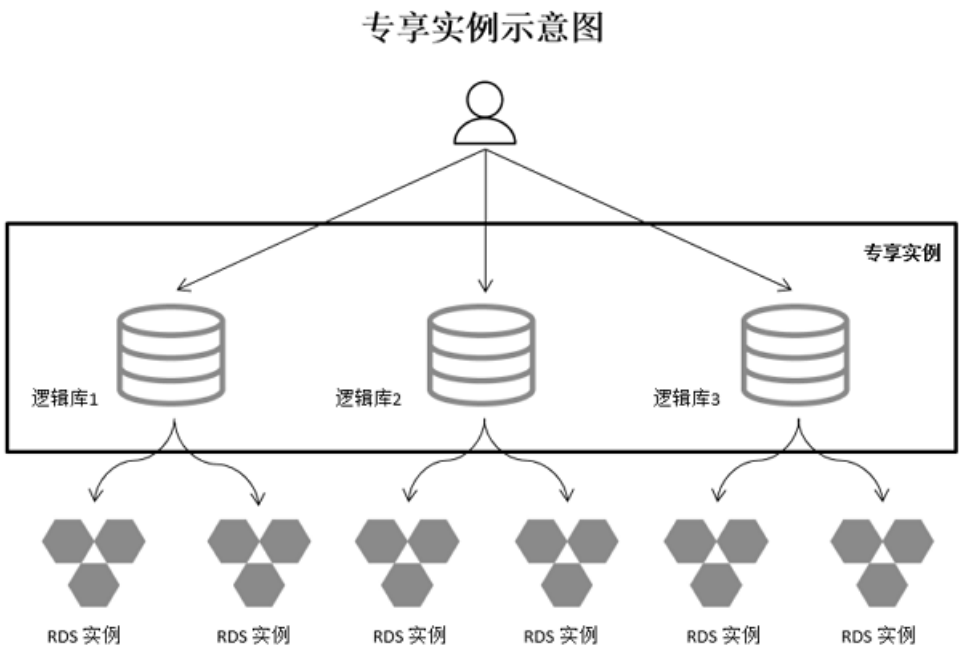
PolarDB-X 1.0主实例专注于解决单机关系型数据库扩展性问题，用于处理实时高并发的在线事务业务。

只读实例

PolarDB-X 1.0只读实例是对主实例的补充，用户可以将复杂SQL计算或有并发诉求的普通查询在只读实例上执行。通过物理资源隔离方式，有效降低主实例的负载。只读实例与主实例可共享同一份数据，也可利用PolarDB-X 1.0存储资源只读实例数据，使数据库诉求更加贴合业务，降低在线数据架构的复杂度，节省运维和资源持有成本。

专享实例

专享实例可以独享物理资源。



1.2. 创建实例

本文将介绍如何创建PolarDB-X 1.0计算资源主实例。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。

4. 单击页面右上角**创建实例**。
5. 在购买页面，设置如下参数。

选项	说明
商品类型	您可以选择商品类型为**包年包月**或**按量付费**。 **说明** ○ 按量付费为后付费模式，通常用于测试或学习了解。 ○ 包年包月为预付费模式，用于长期生产环境稳定使用，单位价格比按量付费便宜。
地域和可用区	通过选择地域和可用区配置实例所在的物理位置。
实例类型	固定为**专享实例**，不可变更。
网络类型	您可以选择网络类型为**经典网络**或**专有网络**。 **说明** 若您选择配置**专有网络**，您需要提前准备相应地域和可用区内的专有网络和虚拟交换机，详情请参见使用专有网络。
MySQL版本	您可以选择**MySQL 8**版本（与 MySQL 8.0 完全兼容）或**MySQL 5**版本（与MySQL 5.x完全兼容）。 **说明** 实例创建后，如需变更MySQL版本，可提交工单联系售后服务进行变更。
存储类型	您可以选择存储类型为**RDS MySQL**（即私有定制RDS MySQL）或**POLARDB**（即租户侧PolarDB MySQL）。
系列	您可以选择不同系列对应不同业务场景，生产环境在线业务建议选择标准版或企业版，详情请参见实例规格。
规格	通过选择规格配置实例物理规格CPU核数和内存大小，详情请参见实例规格。
购买数量	选择需要购买的实例数量。
购买时长	选择实例购买的时长。 **说明** 仅当商品类型为**包年包月**时支持该参数。

6. 单击**立即购买**。
7. 在**确认订单**页面，确认实例配置参数和服务协议后，单击**去支付**。
8. 在**支付**页面确认待支付订单，单击**订购**。
9. 支付成功后，一般需要1~5分钟开通服务。之后您可以返回控制台的**实例列表**页查看新创建的实例。

1.3. 实例变配

一个PolarDB-X是由多个计算节点分组成的分布式集群。单个节点承担了SQL 路由、数据合并、聚合等功能。通过实例变配，您可以变更PolarDB-X的节点规格，在业务繁忙时进行升配，承载更多的业务流量QPS，在业务空闲时进行降配，避免资源浪费。

注意事项

- 降配过程中由于应用与实例连接会中断，可能在短时间内会产生闪断 ，请确保应用程序具备重连机制。
- 使用长连接时，升配后新增节点无法立即接收到流量，建议灰度重启ECS实例（即先重启拥有少量业务服务的ECS，观察业务情况，确认没问题后，再重启其他所有ECS）。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标实力，在目标实例右侧的**操作**栏中，单击**升配**或**降配**。
4. 在变配页面中，选择待变配的规格，阅读并选中服务协议，单击**立即购买**。
5. 在**支付**页面，确认待支付订单和支付方式，单击**支付**。
6. 购买后，请耐心等待约5分钟开通服务，之后您可以在**实例列表**中查看变更后的实例配置。

1.4. 升级版本

您可以升级更新PolarDB-X 1.0实例到最新版本，快速体验实例新特性，本文将介绍如何在控制台上升级PolarDB-X 1.0实例版本。

注意事项

- 升级前请先在推荐版本实例上进行全面验证，避免SQL兼容性问题，推荐使用变更评估进行验证。
- 升级时，不要进行其它操作（如建库、平滑扩容等）。
- 升级后24小时内可以回滚，但不能变更配置。若超过回滚时效，需要回滚请提交工单。
- 升级会有闪断和少量报错，请在业务低峰期执行。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。
5. 在**常用操作**区域，单击**升级版本**。
6. 在弹出的对话框中设置如下参数。

参数	说明

参数	说明
性能评估	您可以选择**开启**或**关闭**性能评估。 **说明** 性能评估时会对实例的性能产生一定影响，因此建议您在低峰期执行性能评估操作。
版本系列	请在**版本系列**下拉框中选择目标版本进行升级。 **说明** 关于PolarDB-X实例各版本的详系信息，请参见版本说明。

7. 单击**变更评估**。

   说明 若您在控制台上没有看到**变更评估**按钮，说明当前实例所在地域不支持该功能，您可以跳过如下步骤直接单击**一键变更**进行版本升级。

8. 您可以在实例**基本信息**页面右上角，单击图标查看评估任务进度。

9. 评估完成后，单击任务列表**操作**栏中的**查看**获取评估报告详情。

10. 在弹出的对话框中，您可以查看评估报告结果，也可以单击**版本变更**进行版本升级。

1.5. 参数设置

您可以在控制台上查看PolarDB-X实例和数据库相关的参数说明，还可以自定义修改这些参数值。

注意事项

- 只读实例仅支持设置**数据库级**参数，不支持**实例级**的参数设置。
- 部分参数仅支持特定的实例版本，详请请参见参数说明。

参数说明

参数级别	参数变量	参数说明	支持版本
实例级	SLOW_SQL_TIME	慢SQL阈值，超过该阈值的SQL会被记录到逻辑慢SQL日志中。	实例版本需为5.3.4-15378085或以上才支持该参数，关于实例版本，详情请参见实例版本。
	LOGIC_IDLE_TIMEOUT	Client端与PolarDB-X 1.0实例间的连接空闲时间，单位为毫秒（ms）。	实例版本需为5.3.4-15378085或以上才支持该参数，关于实例版本，详情请参见实例版本。
	MAX_ALLOWED_PACKET	Client端与PolarDB-X 1.0实例交互的最大网络包大小，单位为Byte。	实例版本需为5.3.4-15378085或以上才支持该参数，关于实例版本，详情请参见实例版本。
数据库级	MAX_POOL_SIZE	PolarDB-X计算层节点和单个存储层物理分片的最大链接数。	所有版本
	MIN_POOL_SIZE	PolarDB-X计算层节点和单个存储层物理分片的最小链接数。	所有版本
	IDLE_TIMEOUT	PolarDB-X计算层节点和存储层节点的连接空闲时间。	所有版本
	SOCKET_TIMEOUT	PolarDB-X计算层节点和存储层节点交互时的SQL超时时间。	所有版本
	FORBID_EXECUTE_DML_ALL	是否禁止删除或更新全表。	所有版本
	TEMP_TABLE_MAX_ROWS	PolarDB-X分布式查询中临时表的大小。	实例版本需为5.3.4-15378085或以上才支持该参数，关于实例版本，详情请参见实例版本。
	MERGE_UNION_SIZE	当查询一个数据库上多个分表时最多支持使用UNION合并的分表数量。	实例版本需为5.3.4-15378085或以上才支持该参数，关于实例版本，详情请参见实例版本。

参数级别	参数变量	参数说明	支持版本
	PURE_ASYNC_DDL_MODE	是否启用异步DDL引擎的纯异步模式。 • 若启用纯异步模式，客户端连接PolarDB-X实例执行DDL后会立即返回任务提交结果，您只能通过异步DDL管理语句查看DDL的执行状态。 • 若禁用纯异步模式（即仍为同步模式），客户端会执行DDL阻塞直到DDL执行完毕才返回执行结果。	实例版本需为5.3.4-15378085或以上才支持该参数，关于实例版本，详情请参见实例版本。
	ENABLE_RANDOM_PHY_TABLE_NAME	是否启用为物理表使用随机表名。	实例版本需为5.3.4-15378085或以上才支持该参数，关于实例版本，详情请参见实例版本。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中，单击**诊断与优化 > 参数设置**，您可以选择修改**实例级**或**数据库级**的参数。
   - 修改**实例级**参数
     a. 在**实例级**页签下找到目标参数，单击**运行参数值**栏的。

> **说明** 您可以单击**参数描述**栏的图标查看目标参数的详细说明。

b. 在弹出的对话框中，输入新的参数值，单击**确定**。

> **说明** 请输入**可修改**参数取值范围内的参数值，否则当您**批量提交**时会出现错误提示。

- 修改**数据库级**参数

a. 单击**数据库级**页签后，找到目标数据库名称并单击目标数据库页签。

b. 找到目标参数，单击**运行参数值**栏的✎。

> **说明** 您可以单击**参数描述**栏的?图标查看目标参数的详细说明。

c. 在弹出的对话框中，输入新的参数值，单击**确定**。

> **说明** 请输入**可修改**参数取值范围内的参数值，否则当您**批量提交**时会出现错误提示。

6. 修改**运行参数值**后，单击右上角**批量提交**。

> **说明** 对于**是否重启**栏显示为**是**的参数，**批量提交**新的**运行参数值**后，您需要重启实例新参数值才会生效。

1.6. 释放实例

您可以根据业务需求手动释放按量付费（即后付费）的实例。

注意事项

- 包年包月（即预付费）实例不支持手动释放，实例到期后会自动被释放。
- 只有在运行状态下的实例才能被手动释放。实例被释放后，数据将无法找回，请谨慎操作。
- 实例释放操作本身不会主动删除PolarDB-X实例上的数据，但实例释放后不可恢复，请谨慎操作。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标实例，在**操作**列单击**释放**。
4. 在弹出的对话框中，单击**确定**。

2.私有RDS管理

2.1. 概述

PolarDB-X 1.0由计算层DRDS实例与存储层私有定制RDS实例组成，通过挂载多个MySQL进行分库分表水平拆分。PolarDB-X 1.0存储层私有定制RDS实例与计算层DRDS实例配合更加深度紧密，性能表现更优秀，服务链路更稳定，售卖价格更优惠。

注意事项

为便于集中运维管理，RDS控制台不展示私有定制RDS实例信息。但您可以在云原生分布式控制台对私有定制RDS实例进行集中运维管理。

费用

关于PolarDB-X 1.0存储层私有定制RDS实例的费用，请参见PolarDB-X 1.0定价。

规格族

PolarDB-X 1.0存储层私有定制RDS实例包括通用型和独享型两种规格族。

规格族	描述	适用场景
通用型	独享被分配的内存，与同一物理机上的其他通用型实例共享CPU和存储资源。通过资源复用享受规模红利，性价比较高。存储大小不与CPU和内存绑定，可以灵活选配。	对性能稳定性要求较低的应用场景。
独享型	具有完全独享的CPU和内存，性能长期稳定，不会因物理机上其它实例的行为而受到影响。独享型的顶配是独占物理机型，也称为独占主机型，完全独占一台物理机的所有资源。	以数据库为核心系统的业务场景，例如金融、电商、政务、大中型互联网业务等。

运维管理

您可以在云原生分布式控制台私有定制RDS实例进行如下操作：

- 查看私有RDS实例信息
- 申请或释放外网地址
- 设置可维护时间段
- 创建高权限账号
- 参数设置
- 主备切换
- 切换可用区
- 设置白名单

2.2. 查看私有RDS实例信息

本文介绍如何查看私有定制RDS的基本信息和配置信息。

前提条件

进行私有RDS管理之前，您需要先购买私有定制RDS实例，详情请参见步骤二：在私有定制RDS实例之上构建PolarDB-X 1.0数据库。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标PolarDB-X 1.0实例，单击实例ID。
4. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
5. 找到目标私有定制RDS实例，单击实例ID。

6. 在弹出的**基本信息**对话框中您可以查看如下信息：

- 您可以在**基本信息**区域查看目标私有定制RDS实例的ID、实例类型及系列以及内外网地址和端口等信息。
- 您可以在**配置信息**区域查看目标私有定制RDS实例的规格、存储空间以及最大IOPS和最大连接数等信息。

2.3. 申请或释放外网地址

私有定制RDS支持内网地址和外网地址两种地址类型，默认提供内网地址供您内部访问私有定制RDS实例，如果需要通过外网访问，您需要申请外网地址。

内网地址和外网地址介绍

地址类型	说明

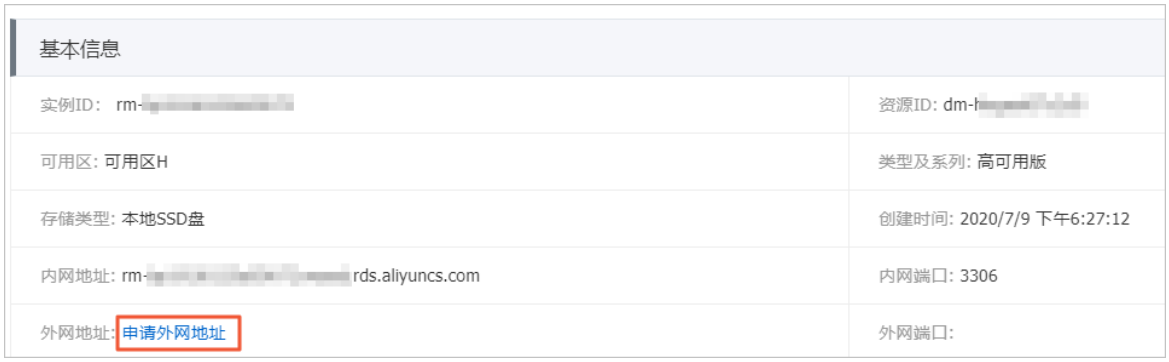
地址类型	说明
内网地址	• 默认提供内网地址，无需申请，无法释放。 • 通过内网访问私有定制RDS实例时，安全性高，而且可以实现私有定制RDS的最佳性能。

外网地址	• 外网地址需要手动申请，不需要时也可以释放。
	• 无法通过内网访问私有定制RDS实例时，您需要申请外网地址。具体场景如下：
	  ○ ECS实例访问私有定制RDS实例，且ECS实例与私有定制RDS实例位于不同地域，或者ECS实例与私有定制RDS实例的网络类型不同。
	  ○ 阿里云以外的设备访问私有定制RDS实例。
	说明
	• 外网地址会降低实例的安全性，请谨慎使用。
	• 为了获得更快的传输速率和更高的安全性，建议您将应用迁移到与您的私有定制RDS实例在同一地域且网络类型相同的ECS实例，然后使用内网地址。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标PolarDB-X 1.0实例，单击实例ID。
4. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
5. 找到目标私有定制RDS实例，单击实例ID。

6. 在弹出的**基本信息**对话框中，单击**申请外网地址**或**释放外网地址**。



> **说明** **释放外网地址**将会导致您无法从外网连接至目标私有定制RDS实例，且已使用外网地址连接的应用会无法访问目标实例，因此在释放前请确保您的应用已更换至通过内网地址连接实例。

2.4. 设置可维护时间段

为保障私有定制RDS实例的稳定性，后端系统会不定期对实例进行维护操作。您可以根据业务规律，将可维护时间段设置在业务低峰期，以免维护过程中可能对业务造成的影响。

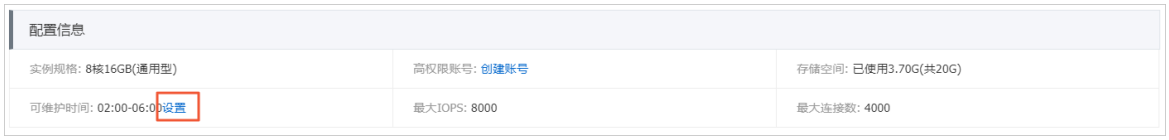
注意事项

- 实例维护当天，为保障整个维护过程的稳定性，实例会在可维护时间段之前进入实例维护中的状态。当实例处于该状态时，对数据库的访问以及查询类操作（如性能监控）不会受到任何影响，但除了账号管理、数据库管理和IP白名单设置外的变更操作（如升降级、重启等）均暂时无法使用。
- 在可维护时间段内，实例会发生1到2次连接闪断，请确保应用程序具有重连机制。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标PolarDB-X 1.0实例，单击实例ID。
4. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
5. 找到目标私有定制RDS实例，单击实例ID。

6. 在**配置信息**区域，单击**可维护时间**后的**设置**。



7. 在弹出的对话框中，选择一个可维护时间段，单击**确定**。

> 说明
> - 默认可维护时间段为02:00~06:00。
> - 自定义的可维护时间段仅支持一天中的任意一个小时（如00:00~01:00）。

2.5. 创建高权限账号

本文介绍如何在控制台上为私有定制RDS实例创建高权限账号。

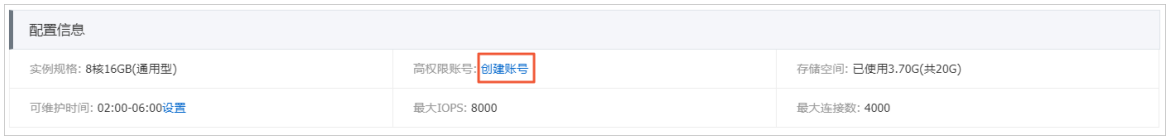
注意事项

- 只能通过控制台或API创建和管理高权限账号。
- 一个实例只支持创建一个高权限账号，高权限账号可以管理所有普通账号和数据库，并拥有实例下所有数据库的所有权限。
- 高权限账号可以断开任意账号的连接，您可以通过高权限账号实现个性化和精细化的权限管理（如按用户分配不同表的查询权限）。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标PolarDB-X 1.0实例，单击实例ID。
4. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
5. 找到目标私有定制RDS实例，单击实例ID。

6. 在**配置信息**区域，单击**创建账号**。



7. 在弹出的对话框中，设置如下参数：

参数	说明
数据库账号	填写账号名称，名称需满足如下要求： ○ 账号名称长度为4~20位。 ○ 以字母开头，由大写字母、小写字母、数字、下划线组成。

参数	说明
账号类型	固定为**高权限账号**，无需选择。 **说明** 若需要创建普通账号，您可以使用高权限账号登录目标数据库后通过相关SQL语句创建普通账号，详情请参见执行SQL基本操作。
新密码	设置账号密码，密码需满足如下要求： ○ 长度为8~32个字符。 ○ 由大写字母、小写字母、数字、特殊字符中的至少三种组成。 ○ 特殊字符为： !@#$%^&*()_+-=
确认密码	再次输入密码。

8. 单击**确定**。

2.6. 参数设置

您可以在控制台上查看私有定制RDS实例详细的参数说明或自定义这些参数值。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标PolarDB-X 1.0实例，单击实例ID。
5. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
6. 找到目标私有定制RDS实例，单击左侧**操作**栏中的**更多 > 参数设置**。
7. 在弹出的对话框中找到目标参数，单击**运行参数值**栏的图标。

说明 您可以单击参数**描述**栏的图标查看目标参数的详细说明。

8. 在弹出的对话框中，输入新的参数值，单击**确定**。

> **说明** 请输入**可修改参数**取值范围内的参数值，否则当您**批量提交**时会出现错误提示。

9. 修改**运行参数值**后，单击右上角**批量提交**。

> **说明** 对于**是否重启**栏显示为**是**的参数，**批量提交**新的**运行参数值**后，您需要重启实例新参数值才会生效。

2.7. 主备切换

您可以设置主备实例切换，切换后原来的主实例会变成备实例。

注意事项

- 主备实例切换过程中可能会有闪断，请确保您的应用程序具有自动重连机制。
- 如果实例下挂载有只读实例，那么主备实例切换后，只读实例的数据会有几分钟的延迟，因为需要重建复制链路、同步增量数据等。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标实例，单击实例ID。
4. 找到目标实例，单击右侧**操作**栏中的**更多 > 主备切换**。
5. 您可以根据业务需要在实例主备切换页面选择**切换时间**。

- 若您需要立刻执行主备切换，请单击**立即切换**。
- 若您需要在其他时间进行切换，请单击**可维护时间内切换**。

> 说明
> - 默认的可维护时间段为18:00Z~22:00Z。
> - 您可以单击**修改**自定义可维护时间段，自定义的可维护时间段仅支持一天中的任意一个小时（如00:00~01:00）。

2.8. 切换可用区

在使用私有RDS实例时，您可以切换实例到其他可用区，并需同时切换数据库资源。

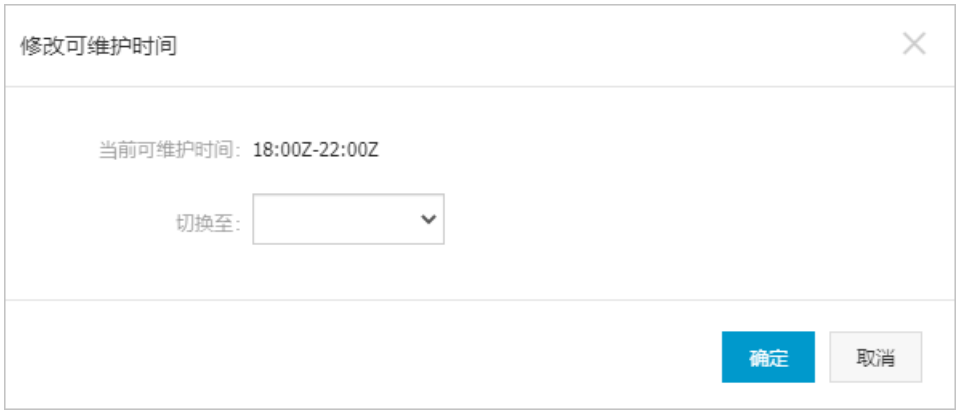
注意事项

- 切换时会有约30秒的闪断，请确保应用具有自动重连机制 。
- 切换专有网络VPC和虚拟交换机会造成虚拟IP（VIP）的变更，请您在应用程序中尽量使用内外网连接地址进行连接，不要使用IP地址。
- VIP的变更会短暂影响到PolarDB-X的可用性，请及时在PolarDB-X控制台刷新并查看连接信息。
- 如果有正在执行的DTS，可用区迁移后，需要重启相应的DTS任务。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
5. 单击右侧**操作**栏中的**更多 > 切换可用区**。
6. 在弹出的界面选择要迁移到的目标可用区和**切换时间**，然后单击确定。

- 单击**立即切换**。
- 在**可维护时间内切换**，单击**修改**可以修改可维护时间段。



2.9. 设置白名单

私有定制RDS实例创建后，您还需要设置私有定制RDS实例的IP白名单，以允许外部设备访问该实例。设置白名单并不会影响实例的正常运行。您可以通过定期维护白名单，保证实例得到高级别的访问安全保护。本文将介绍白名单的相关操作。

注意事项

- 默认情况下，IP白名单只包含IP地址127.0.0.1，表示任何IP地址均无法访问该私有定制RDS实例。只有已添加到IP白名单中的IP地址才可以访问该实例。
- 若将IP白名单设置为%或者0.0.0.0/0，表示允许任何IP地址访问该实例。该设置将极大降低数据库的安全性，如非必要请勿使用。

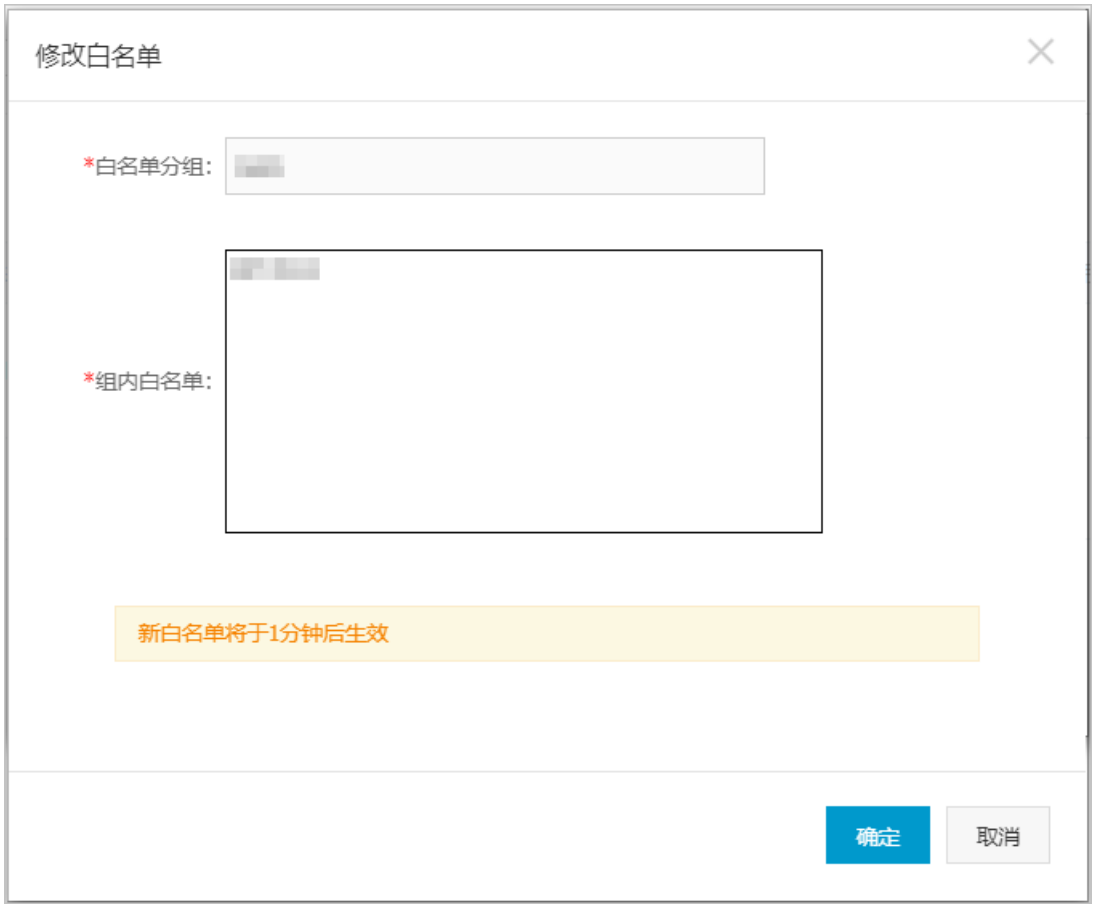
修改组内白名单

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标PolarDB-X 1.0实例，单击实例ID。
4. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
5. 找到目标私有定制RDS实例，单击右侧操作栏中的**更多 > 设置白名单**。

6. 在弹出的对话框中，找到目标白名单分组，单击右侧的**修改**。

> **说明** 系统只有默认的白名单分组，您也可以根据业务需求新增白名单分组。单击**添加白名单分组**，新增一个自定义分组。填写名称以及IP地址或IP段，单击**确定**。

7. 对组内白名单进行修改，单击**确认**。



> **说明**
> - 可以在组内白名单中填写IP地址（如192.168.0.1，表示允许192.168.0.1的IP地址访问私有RDS）或IP段（如192.168.0.0/24，表示允许从192.168.0.1到192.168.0.255的IP地址访问私有RDS）。
> - 多个IP地址或IP段间用英文逗号（,）隔开（如192.168.0.1,192.168.0.0/24）。

删除白名单分组

> **说明** 不支持删除默认的白名单分组。

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标PolarDB-X 1.0实例，单击实例ID。

5. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
6. 找到目标私有定制RDS实例，单击右侧操作栏中的**更多 > 设置白名单**。
7. 在弹出的对话框中，找到目标白名单分组，单击右侧的**删除**。
8. 单击**确认**。

清空默认白名单

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标PolarDB-X 1.0实例，单击实例ID。
5. 在左侧导航栏中单击**配置与管理 > 私有RDS管理**。
6. 找到目标私有定制RDS实例，单击右侧操作栏中的**更多 > 设置白名单**。
7. 在弹出的对话框中，找到默认白名单，单击右侧**清空**。
8. 单击**确认**。

> **说明** 清空默认白名单时，不能删除IP地址127.0.0.1。

3.只读实例

3.1. 实例介绍

PolarDB-X 1.0计算资源只读实例是对PolarDB-X 1.0计算资源主实例的一种扩展补充，兼容主实例的SQL查询语法。

背景信息

PolarDB-X 1.0计算资源只读实例与主实例可以共享同一份数据，通过物理资源隔离的方式，缓解主实例的负载压力，降低业务架构的链路复杂度，无需进行额外的数据同步操作，节省运维及预算成本。可直接在RDS只读实例或RDS主实例上进行复杂数据查询分析。提供多种实例类型，可抵御超高并发的访问压力、减少复杂查询的响应时间。

注意事项

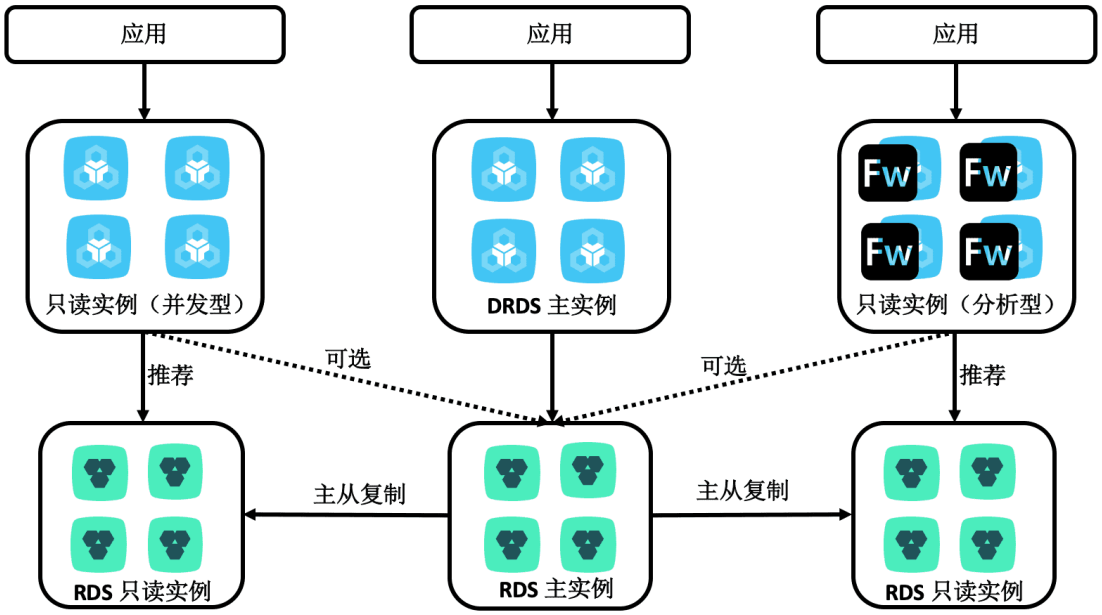
- 只读实例与主实例必须属于同一个地域，但是可以在不同的可用区。
- 只读实例无法单独存在，必须隶属于某个主实例。若您在主实例上创建了数据库，该数据库也会被复制到只读实例上。同理，若您在主实例上删除数据库时，只读实例上对应的数据库也会被删除。
- 目前，PolarDB-X 1.0计算资源只读实例仅支持专有网络VPC实例，只读实例上的RDS实例必须是VPC实例。

使用限制

- 不支持将数据迁移至只读实例。
- 不支持在只读实例上创建或删除数据库。
- 不支持克隆只读实例。
- 只读实例不支持使用数据变更DML语句。

只读实例类型

PolarDB-X 1.0计算资源只读实例分为并发型只读实例和分析型只读实例两种类型，PolarDB-X 1.0计算资源只读实例的拓扑如下：



- 并发型只读实例

  通过物理资源隔离的方式，可抵御超高流量的并发查询，以确保在线业务链路的稳定性。并发型只读实例主要应用于如下业务场景：

  - 存在高并发、大流量简单查询的业务场景。
  - 离线抽取数据的业务场景。

- 分析型只读实例

  在并发型只读实例的能力基础上，增加PolarDB-X 1.0团队自研的分布式MPP内存计算执行引擎实现对复杂算子并行计算。用于加速海量数据下的复杂分析SQL的执行效率，显著减少复杂分析SQL的响应时间，可快速返回在海量数据量下的多表JOIN、聚合、排序等操作的处理结果。分析型只读实例主要应用于如下业务场景：

  - 需要在海量数据下进行多表JOIN、聚合、排序、子查询等复杂SQL查询操作的场景。
  - 运营、BI相关后台系统的报表、大盘等分析型业务。

计费方式

PolarDB-X 1.0计算资源只读实例提供多种规格配置的按量付费，计费周期为1小时，不足1小时按1小时计算，计费详情请参见只读实例定价说明。

> **说明** 关于只读实例欠费后的服务可用性，请参见欠费预警/停机策略。

3.2. 创建实例

本文主要介绍如何创建PolarDB-X 1.0计算资源主实例只读实例。

注意事项

- 如果主实例存在数据库，只读实例创建成功之后，需要在只读实例管理中选择需要使用的RDS，详情请参见管理实例。

- 若主实例的存储类型为RDS MySQL，则仅当网络类型为专有网络时才支持创建只读实例。
- 存储类型为PolarDB MySQL的主实例在经典网络和专有网络下均可创建只读实例。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。
5. 在**相关实例**区域，单击**创建DRDS只读实例**。

6. 在弹出的对话框中，设置如下参数。

参数	说明
实例类型	您可以选择**并发型**或**分析型**，关于两种只读实例类型的差别，请参见只读实例类型。
可用区	在主实例地域下，为只读实例选择可用区。主实例和只读实例可以在不同可用区。
实例网络类型	固定为**专有网络**，不可变更。
VPC	选择专有网络名称。
虚拟交换机	选择虚拟交换机名称。
实例系列	您可以选择不同系列对应不同业务场景，生产环境在线业务建议选择标准版或企业版，详情请参见实例规格。
实例规格	通过选择规格配置实例物理规格CPU核数和内存大小，详情请参见实例规格。

7. 单击**立即购买**。
8. 在**确认订单**页面确认参数配置详情并选中服务协议，单击**去支付**。

> **说明** 支付后完成后需等待1~5分钟开通服务，创建成功后您可以在控制台的**实例列表**页中查看该只读实例。

3.3. 管理实例

本文主要介绍如何管理PolarDB-X 1.0计算资源只读实例。

查看只读实例

- 通过实例列表页查看只读实例。

  在实例列表页，选择只读实例的ID，进入目标只读实例的基本信息页。

> **说明** 实例ID前带有R标识的为只读实例。

- 通过主实例的基本信息页查看只读实例。
  i. 在实例列表页面，单击主实例ID，进入到主实例的基本信息页。
  ii. 在**相关实例**区域，将鼠标放置在只读实例的数字上面，即会出现只读实例的ID。

  iii. 单击实例ID，进入目标只读实例的基本信息页。

查看只读实例的数据库列表

只读实例的数据库是从主实例复制而来，只读实例数据库下的PolarDB-X 1.0存储资源实例与主实例下数据库的PolarDB-X 1.0存储资源实例是同一套实例。为了不影响主实例数据库的业务，建议为只读实例选择不在主实例数据库中使用的PolarDB-X 1.0存储资源只读实例。

如果主实例已经存在数据库了，创建只读实例后，进入到只读实例数据库列表，数据库的状态都是**待选RDS**状态。

单击**待选RDS**即可进入到选择私有定制RDS实例的界面。

> **说明** 不支持在PolarDB-X 1.0计算资源只读实例的管理界面创建和删除数据库，只读实例的数据库随着主实例的数据库创建而创建，随着主实例的数据库删除而删除。

选择私有定制RDS实例

PolarDB-X 1.0计算资源只读实例的数据存储在PolarDB-X 1.0存储资源实例（包括PolarDB-X 1.0存储资源主实例和PolarDB-X 1.0存储资源只读实例）上。 在**存储节点信息**页面，您可以查看当前数据库下可供选择的所有PolarDB-X 1.0存储资源实例。

列表中的PolarDB-X 1.0存储资源主实例是PolarDB-X 1.0计算资源主实例对应的数据库正在使用的主实例，列表以PolarDB-X 1.0存储资源主实例为单位进行分组。单击列表右侧**操作**栏中的**选用**，即可选择目标PolarDB-X 1.0存储资源实例在当前PolarDB-X 1.0计算资源只读实例中使用。

说明

- 建议选择新的PolarDB-X 1.0存储资源只读实例，避免在PolarDB-X 1.0计算资源主实例与PolarDB-X 1.0计算资源只读实例中使用相同的PolarDB-X 1.0存储资源只读实例，从而影响PolarDB-X 1.0计算资源主实例的业务稳定性。
- PolarDB-X 1.0计算资源主实例的数据库完成平滑库扩容以后，默认会选择目标PolarDB-X 1.0存储资源主实例作为PolarDB-X 1.0计算资源只读实例上对应的存储实例，请尽快将该PolarDB-X 1.0存储资源主实例切换为对应的PolarDB-X 1.0存储资源只读实例。

释放只读实例

您可以通过实例列表页或只读实例基本信息页来释放PolarDB-X 1.0计算资源只读实例。只读实例释放后，目标只读实例下的数据库也会被删除，该操作对于PolarDB-X 1.0计算资源主实例的数据库没有任何影响。

4.账号管理

PolarDB-X 提供账号管理服务，本文介绍如何在控制台上创建账号、修改账号权限和重置账号密码。

注意事项

- 仅实例版本为5.3.6 10460044及以上版本才支持账号管理功能。
- 目前控制台只支持对普通账号授予DML、DDL、只读、读写四种权限，如需授予更精细化的权限，请使用SQL语句进行操作，详情请参见执行SQL基本操作。

账号类型和权限

- PolarDB-X 1.0实例支持如下两种数据库账号类型：

账号类型	说明
高权限账号	○ 只能通过SQL语句进行创建和管理，详情请参见执行SQL基本操作。 ○ 一个实例中只能创建一个高权限账号，可以管理所有普通账号和数据库。 ○ 拥有更多权限，可满足个性化和精细化的权限管理需求，例如可按用户分配不同表的查询权限。 ○ 拥有实例下所有数据库的所有权限，可以断开任意账号的连接。
普通账号	○ 可以通过控制台、API或SQL语句进行创建和管理。 ○ 一个实例可以创建多个普通账号，具体的数量与实例内核有关。 ○ 需要手动给普通账号授予特定数据库的权限。 ○ 普通账号不能创建和管理其他账号，也不能断开其他账号的连接。

- 不同类型的账号在不同权限下支持的SQL操作如下表所示：

账号类型	权限	SELECT	INSERT	UPDATE	DELETE	INDEX	ALTER	CREATE	DROP	GRANT
普通账号	DDL	不支持	不支持	不支持	不支持	支持	支持	支持	支持	不支持
	DML	支持	支持	支持	支持	不支持	不支持	不支持	不支持	不支持
	只读	支持	不支持	不支持	不支持	不支持	不支持	不支持	不支持	不支持
	读写	支持	支持	支持	支持	支持	支持	支持	支持	不支持
高权限账号	Root	支持	支持	支持	支持	支持	支持	支持	支持	支持

创建账号

1. 登录云原生分布式数据库控制台。

2. 在页面左上角选择目标实例所在地域。
3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中，单击**账号管理**。
5. 单击页面右上角**创建账号**。

> **说明** 仅支持通过控制台创建普通账号，若您需要创建高权限账号，请使用SQL语句进行创建，详情请参见执行SQL基本操作。

6. 在弹出的对话框中，设置如下参数：

参数	说明
数据库账号	填写账号名称。 **说明** 账号名称需满足如下要求： ○ 由小写字母、数字或特殊字符组成，长度需为2~16个字符。 ○ 需以字母开头，以字母或数字结尾。 ○ 不能和已有的账号名重复。 ○ 账号创建完成后，完整的账号名称由您在此填写的名称和host（默认为 % ）组成，表示允许该账号从任意主机登录数据库。
新密码	设置账号密码。 **说明** 账号密码需满足如下要求： ○ 密码长度需为8~32个字符。 ○ 需由大写字母、小写字母、数字、特殊字符中的任意三种组成。 ○ 特殊字符为： !@#$%^&*()_+-=
确认新密码	再次输入密码。
授权数据库	为该账号授予一个或多个数据库的权限。 i. 选中一个或多个数据库，单击**授权>**将其从左侧**数据库列表**添加到右侧**已授权数据库**。 ii. 在右侧**已授权数据库**中，为目标数据库选择权限。 **说明** ○ 默认权限为**只读**，您也可以修改权限为**读写**、**仅DDL**或**仅DML**。 ○ 如果要为多个数据库批量设置相同的权限，可以单击**已授权数据库**框右上角的权限对应按钮，如**全部读写**。

7. 单击**确定**。

修改普通账号权限

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中，单击**账号管理**。
5. 在目标账号右侧的**操作**栏中，单击**修改权限**。

> **说明** 高权限账号拥有所有数据库的所有权限，因此不需要**修改权限**。

6. 在弹出的对话框中，设置如下参数：

参数	说明
授权数据库	为该账号修改一个或多个数据库的权限。 i. 选中一个或多个数据库，单击**授权>**或**<移除**将其添加到目标账号或从目标账号中删除。 ii. 在右侧**已授权数据库**中，为目标数据库选择权限。 **说明** ○ 您也可以将权限修改为**只读**、**读写**、**仅DDL**或**仅DML**。 ○ 如果要为多个数据库批量设置相同的权限，可以单击**已授权数据库**框右上角的权限对应按钮，如**全部读写**。

7. 单击**确定**。

重置账号密码

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中，单击**账号管理**。
6. 在目标账号右侧的**操作**栏中，单击**重置密码**。

7. 在弹出的对话框中，设置如下参数：

参数	说明
新密码	设置账号密码。 **说明** 账号密码需满足如下要求： ○ 密码长度需为8~32个字符。 ○ 需由大写字母、小写字母、数字、特殊字符中的任意三种组成。 ○ 特殊字符为： !@#$%^&*()_+-=
确认新密码	再次输入密码。

8. 单击**确定**。

删除账号

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中，单击**账号管理**。
6. 在目标账号右侧的**操作**栏中，单击**删除**。

说明 控制台不支持删除高**权限账号**操作，但您可以通过SQL语句删除高权限账号，详情请参见执行SQL基本操作。

7. 在弹出的对话框中，单击**确定**

5.监控与报警

5.1. 实例监控

为方便您掌握实例的运行状态，PolarDB-X提供了监控查询功能。您可以在控制台上查看目标实例、数据库、或存储节点的各项监控数据详情。本文将介绍如何在**实例监控**页查看PolarDB-X实例的各项性能监控数据。

实例监控指标表

监控类别	监控指标	参数	说明
资源监控	CPU	cpu	PolarDB-X实例计算节点CPU使用率的平均值。
	内存	old	PolarDB-X实例计算节点的内存使用率。内存使用率波动属于正常现象。
	网络	netin（输入流量）	PolarDB-X实例计算节点的网络输入流量的总和，单位为Kbps。存储节点返回数据到计算节点，会产生网络输入流量。
		netout（输出流量）	PolarDB-X实例计算节点的网络输出流量的总和，单位为Kbps。计算节点发送物理SQL到存储节点，计算节点返回数据到应用，均会产生网络输出流量。
引擎监控	逻辑QPS	lqps	PolarDB-X实例每秒处理的SQL语句数目的总和。
	物理QPS	pqps	PolarDB-X计算节点每秒发送到存储节点的SQL操作数总和。
	逻辑RT	lrt	PolarDB-X实例对于每条SQL的平均响应时间。
	物理RT	prt	PolarDB-X计算节点发送到存储节点的SQL的平均响应时间。
	连接数	con	应用到PolarDB-X实例的连接总数。
	活跃线程数	thread	PolarDB-X实例中用来执行SQL的线程数。

> 说明
> - 上述各**资源监控**项的数据采集周期均为1分钟，各**引擎监控**项的数据采集周期均为5秒钟。
> - 最多支持查看7天内的监控数据。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在**实例列表**页，找到目标实例并单击实例ID。
4. 在左侧导航栏中，单击**监控与报警 > 实例监控**。
5. 在**实例监控**页，设置如下参数：

参数	说明
监控指标	您可以选择**资源监控**或**引擎监控**。
监控指标	选择需要查看的实例监控指标，关于各监控指标详情，请参见实例监控指标表。
查询时间	您可以指定查询时间为1小时、6小时、12小时、1天、1周。您也可以自定义查询时间范围，自定义查询时间范围为1分钟~1周。

相关API

API	说明
DescribeDrdsInstanceMonitor	查看实例监控详情。

5.2. 数据库监控

为方便您掌握实例的运行状态，PolarDB-X提供了监控查询功能。您可以在控制台上查看目标实例、数据库、或存储节点的各项监控数据详情。本文将介绍如何在**数据库监控**页查看PolarDB-X实例中单个数据库的各项性能监控数据。

数据库监控指标表

监控指标	参数	说明
QPS	qps	PolarDB-X实例所选数据库每秒处理的SQL语句数目的总和。
连接数	con	应用到PolarDB-X实例所选数据库的连接总数。
活跃线程	thread	PolarDB-X实例中目标数据库用来执行SQL的线程数。

> **说明**
>
> - 上述各监控项的数据采集周期均为1分钟。
> - 最多支持查看7天内的监控数据。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在**实例列表**页，找到目标实例并单击实例ID。
4. 在左侧导航栏中，单击**监控与报警 > 数据库监控**。
5. 在**数据库监控**页，设置如下参数：

参数	说明
数据库	从**数据库**列表中选中目标数据库。

参数	说明
数据指标	选择需要查看的数据库监控指标，关于各监控指标详情，请参见数据库监控指标表。
查询时间	您可以指定查询时间为1小时、6小时、12小时、1天、1周。您也可以自定义查询时间范围，自定义查询时间范围为1分钟~1周。

相关API

API	说明
DescribeDrdsInstanceDbMonitor	查看数据库监控详情。

5.3. 存储监控

为方便您掌握实例的运行状态，PolarDB-X提供了监控查询功能。您可以在控制台上查看目标实例、数据库、或存储节点的各项监控数据详情。本文将介绍如何在**存储监控**页查看PolarDB-X实例中存储节点的各项性能监控数据。

存储监控指标表

监控项	监控指标	参数	说明
资源监控	**CPU和内存**	MySQL_MemCpuUsage	PolarDB-X实例存储节点的CPU使用率和内存使用率。
	磁盘空间	MySQL_DetailedSpaceUsage	PolarDB-X实例存储节点的总空间、数据空间、日志空间、临时空间和系统空间的使用量，单位为MByte。
	IOPS	MySQL_IOPS	PolarDB-X实例存储节点的IOPS（每秒IO请求次数）。
	连接数	MySQL_Sessions	PolarDB-X实例存储节点的当前活跃连接数和总连接数。
	网络流量	MySQL_NetworkTraffic	PolarDB-X实例存储节点的每秒输入和输出流量，单位为KB。
	TPS/QPS	MySQL_QPSTPS	PolarDB-X实例存储节点的平均每秒SQL语句执行次数和平均每秒事务数。
	InnoDB缓存	MySQL_InnoDBBufferRatio	PolarDB-X实例存储节点中InnoDB缓冲池的读命中率、InnoDB缓冲池的使用率和InnoDB缓冲池脏块的百分率。

监控项	监控指标	参数	说明
引擎监控	InnoDB读写	MySQL_InnoDBDataReadWriten	PolarDB-X实例存储节点中InnoDB平均每秒钟读取的数据量和InnoDB平均每秒钟写入的数据量，单位为KB。
	缓存请求次数	MySQL_InnoDBLogRequests	PolarDB-X实例存储节点平均每秒向InnoDB缓冲池的读次数和平均每秒向InnoDB缓冲池的写次数。
	InnoDB日志	MySQL_InnoDBLogWrites	PolarDB-X实例存储节点的平均每秒日志写请求数、平均每秒向日志文件的物理写次数和平均每秒向日志文件完成的fsync()写数量。
	临时表	MySQL_TmpDiskTableCreates	PolarDB-X实例存储节点执行语句时在硬盘上自动创建的临时表的数量。
	COMDML	MySQL_COMDML	PolarDB-X实例存储节点的平均每秒Delete语句执行次数、平均每秒Insert语句执行次数、 平均每秒Insert_Select语句执行次数、平均每秒Replace语句执行次数、平均每秒Replace_Select语句执行次数、平均每秒Select语句执行次数和平均每秒Update语句执行次数。
	RowDML	MySQL_RowDML	PolarDB-X实例存储节点的平均每秒从InnoDB表读取的行数、平均每秒从InnoDB表更新的行数、平均每秒从InnoDB表删除的行数、平均每秒从InnoDB表插入的行数和平均每秒向日志文件的物理写次数。
	MyISAM读写	MySQL_MyISAMKeyReadWrites	PolarDB-X实例存储节点中MyISAM平均每秒钟从缓冲池中的读取次数、MyISAM平均每秒钟从缓冲池中的写入次数、MyISAM平均每秒钟从硬盘上读取的次数和MyISAM平均每秒钟从硬盘上写入的次数。
	MyISAM索引	MySQL_MyISAMKeyBufferRatio	PolarDB-X实例存储节点中MyISAM平均每秒Key Buffer使用率、MyISAM平均每秒Key Buffer读命中率和MyISAM平均每秒Key Buffer写命中率。

> 说明
> - 上述各**资源监控**项的数据采集周期均为1分钟，各**引擎监控**项的数据采集周期均为5秒钟。
> - 最多支持查看7天内的监控数据。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在**实例列表**页，找到目标实例并单击实例ID。
4. 在左侧导航栏中，单击**监控与报警 > 存储监控**。
5. 在**存储监控**页面，设置如下参数：

参数	说明
数据库	从**数据库**列表中选中目标数据库。
监控指标	您可以选择**资源监控**或**引擎监控**。
监控指标	选择需要查看的存储层资源监控指标，关于各监控指标详情，请参见存储监控指标表。
查询时间	您可以指定查询时间为1小时、6小时、12小时、1天、1周。您也可以自定义查询时间范围，自定义查询时间范围为1分钟~1周。

相关API

API	说明
DescribeRDSPerformance	查看存储层RDS的监控详情。

5.4. 管理报警规则

您可以通过PolarDB-X控制台创建和管理阈值报警规则，帮助您及时了解PolarDB-X实例、数据库和存储资源的各项监控数据是否出现异常并快速进行处理。本文将以实例监控为例介绍如何管理报警规则。

创建报警规则

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在**实例列表**页，找到目标实例并单击实例ID。
4. 在左侧导航栏中，选择**监控与报警 > 实例监控**。
5. 在**实例监控**页，单击**设置报警规则**。

6. 在**报警规则列表**页，单击**创建报警规则**。

7. 在弹出的页面中，根据业务需要设置各项参数。

> 说明 在**关联资源**区域，选择**产品**为**云原生分布式数据库PolarDB-X 1.0**。关于其它参数的设置，详情请参见创建阈值报警规则。

8. 单击**确认**即可。

报警规则管理

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在**实例列表**页，找到目标实例并单击实例ID。
4. 在左侧导航栏中，选择**监控与报警 > 实例监控**。
5. 在**实例监控**页，单击**设置报警规则**。

6. 在**报警规则列表**页的**阈值报警**页签下，您可以对已有的报警规则执行以下操作。

- 单击**操作**栏中的**查看**，可查看报警规则的基本信息。
- 单击**操作**栏中的**报警历史**，可查看报警历史。
- 单击**操作**栏中的**修改**，可修改报警规则。
- 单击**操作**栏中的**禁用**，可禁用报警规则。
- 单击**操作**栏中的**删除**，可删除报警规则。
- 单击**通知对象**栏中的**查看**，可查看报警联系组、报警联系人及报警通知方式。

6.切换可用区

PolarDB-X 1.0支持切换可用区服务，若出现错误选择可用区、原可用去库存不足等情况，您可以将原PolarDB-X 1.0实例迁移到其它可用区。

注意事项

- 切换可用区过程一般持续约1~2分钟，建议在流量低峰期操作避免影响正常业务。
- 可用区切换时会发生秒级的连接闪断，请确保您的应用客户端具备自动重连机制。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。
5. 在**常用操作**区域，单击**切换可用区**。

6. 在弹出的对话框中，从下拉框中选中目标可用区，单击**确定**。

7.SQL审计与分析

7.1. 简介

分布式关系型数据库PolarDB-X 1.0与日志服务联合推出 SQL 审计与分析功能，不仅支持历史 SQL 记录的审计，而且提供对 SQL 执行状况、性能指标、安全问题的实时诊断分析能力。

背景信息

您可以在 DRDS 控制台，开通 SQL 审计与分析功能，进行实时日志分析。同时，DRDS 依托日志服务，提供开箱即用的报表中心，使您对数据库执行状况、性能、潜在安全问题了如指掌。

- **操作简单**：轻松配置即可开启，实时进行 SQL 日志的审计与分析。
- **性能无损**：实时拉取 DRDS 节点 SQL 日志文件并上传至日志服务，对实例本身性能无影响。
- **历史问题追溯**：支持导入历史 SQL 日志，追溯问题。
- **实时分析**：依托日志服务，提供 SQL 实时分析、开箱即用的报表中心并支持自定义报表和下钻分析，对数据库执行状况、性能、安全问题了如指掌。
- **实时告警**：支持基于特定指标，定制准实时的告警，确保关键业务异常时可及时响应。
- **价格优势**：存储费用比传统方案更低， 每 GB 日志存储费用低至0.58元/月，详情参考计费方式。

限制与说明

- 使用 SQL 审计与分析功能，需要开通阿里云日志服务。
- 当前支持的区域：华东1（杭州）、华东2（上海）、华北1（青岛）、华北2（北京）、华南1（深圳）、新加坡。其它地区陆续开放中。
- 共享实例不支持 SQL 审计与分析功能。
- SQL 审计日志默认保存45天，您也可以根据需要修改日志保存时间。
- 请勿随意删除或修改日志服务为您默认创建的 Project、Logstore、索引和仪表盘设置，日志服务会不定期更新与升级 SQL 日志审计功能，专属日志库的索引与默认报表也会自动更新。
- 子账号使用 SQL 审计与分析，需要为其授予日志服务相关权限，详细说明请参考为子账号授予 DRDS SQL 审计权限。
- SQL 审计功能默认为关闭状态。开启该功能后，会产生额外费用，详细收费标准请参见计费方式。如果您需要关闭该功能，请在 DRDS 控制台关闭。关闭后不再写入数据，可以查看历史数据，这部分历史数据会产生存储和索引费用。您可以在日志服务控制台删除 Logstore 以删除历史数据，删除后日志服务不再对该部分数据计费。
- 单条 SQL 最大支持长度为5M。

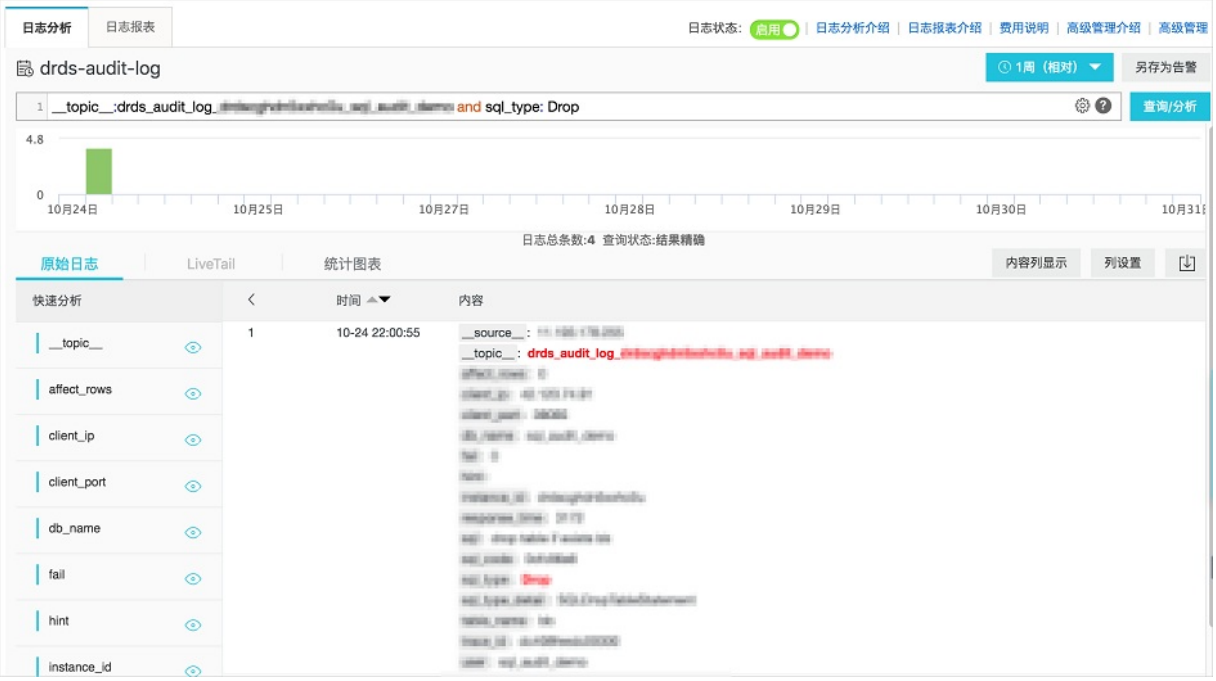
应用场景

问题 SQL 排查

开启 DRDS SQL 审计与分析功能后，您可以对 SQL 日志进行快速检索，排查定位问题。例如，需要查询是否有人执行 DROP 操作，可以执行如下查询：

```
sql_type: Drop
```

如下图所示，查询结果中包括 SQL 的执行时间、用户、执行该 SQL 的客户端 IP 地址等。



高代价SQL模板分析

大多数应用中，SQL 通常是基于若干模板动态生成的，只是参数不同。通过日志服务的实时分析能力，您可以获取当前数据库中的高代价 SQL 列表。

例如执行以下查询：

```
| SELECT sql_code as "SQL模板ID", round(total_time * 1.0 /sum(total_time) over() * 100, 2) as "总体耗时比例(%)" ,execute_times as "执行次数", round(avg_time) as "平均执行时间",round(avg_rows) as "平均影响行数", CASE WHEN length(sql) > 200 THEN  concat(substr(sql, 1, 200), '.....') ELSE trim(lpad(sql, 200, ' ')) end as "样例SQL" FROM  (SELECT sql_code, count(1) as execute_times, sum(response_time) as total_time, avg(response_time) as avg_time, avg(affect_rows) as avg_rows, arbitrary(sql) as sql FROM log GROUP BY sql_code) ORDER BY "总体耗时比例(%)" desc limit 10
```

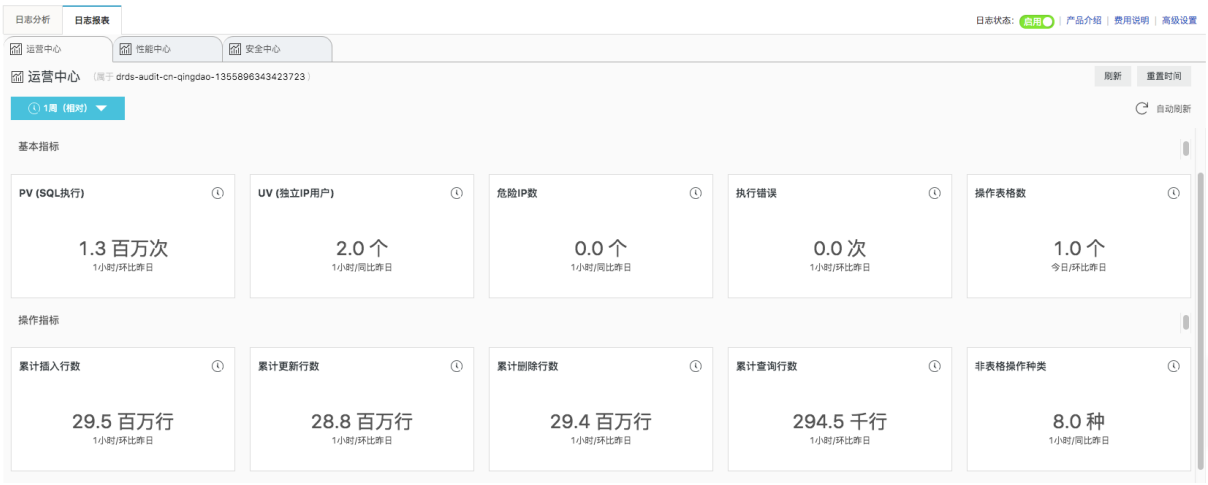
查询结果如下图所示。包括 SQL 模板的 ID、该模板 SQL 占 SQL 的耗时比例、执行次数、平均执行时间、平均影响行数以及样例 SQL。您可以基于该分析结果，找到应用中代价最高的 SQL 模板，对其进行优化。

SQL模板ID	总体耗时比例(%)	执行次数	平均执行时间	平均影响行数	样例SQL
ac0c0ca2	34.97	38116	10343.0	1.0	
4f3b8c1d	27.22	30212	10157.0	8.0	

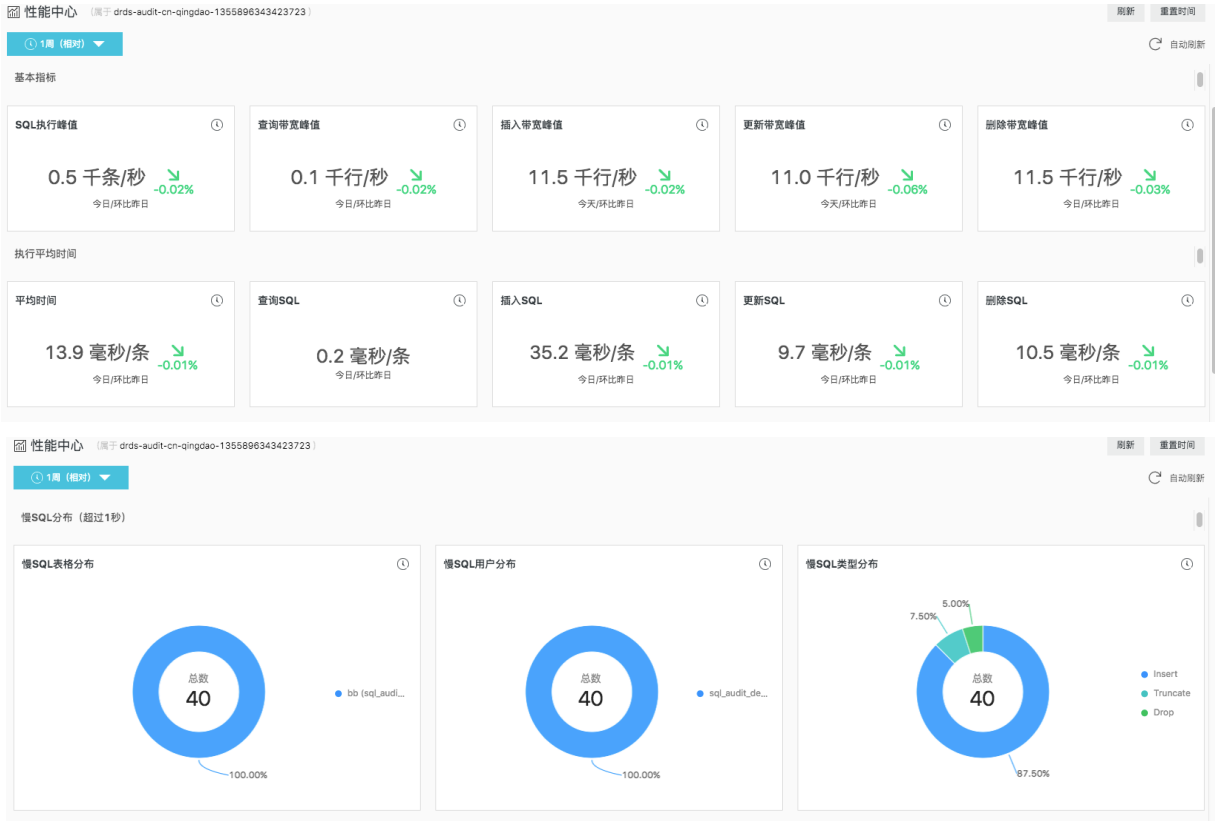
日志报表统计

为了便于分析问题，DRDS SQL 审计分析结合日志服务，为您提供了开箱即用的报表，让您对数据库的执行状况、性能和潜在安全问题进行实时诊断分析，报表页面如下图所示。

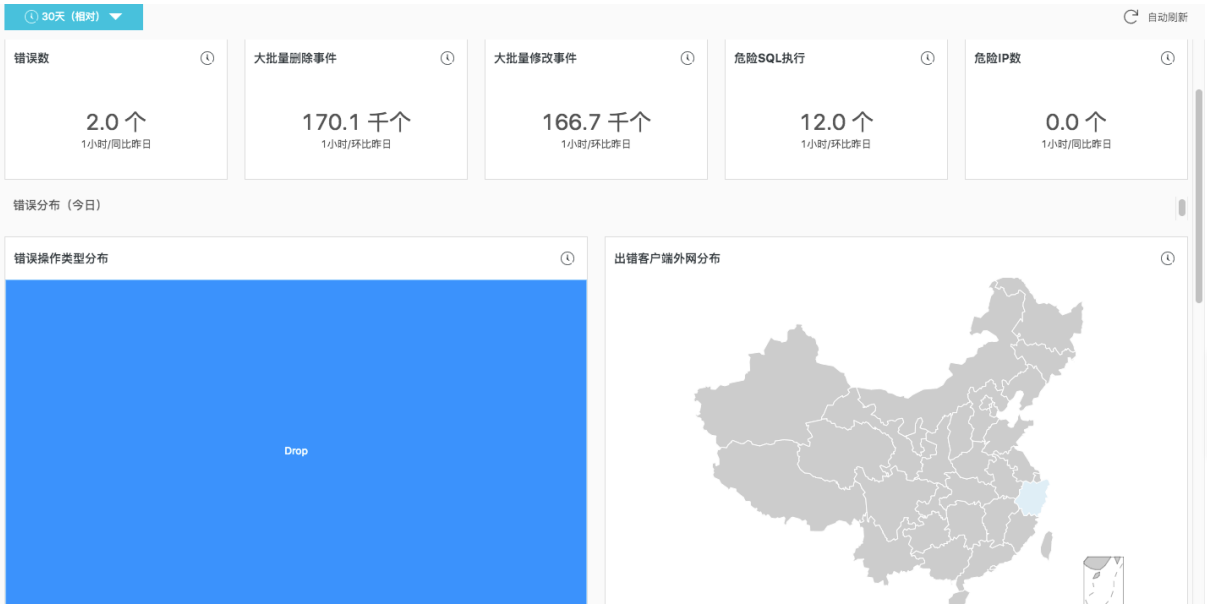
- 运营中心



- 性能中心



- 安全中心



7.2. 开启 SQL 日志审计

SQL 日志审计功能默认关闭，您可以在 DRDS 控制台上手动开启。默认情况下，只对开启 SQL 日志审计功能之后产生的日志数据进行审计分析，您也可以导入部分历史数据。

背景信息

- 开通阿里云日志服务。
- 已购买 DRDS 专享实例，并创建了数据库。
- 默认情况下，只有主账号可以开启并操作 SQL 日志审计功能。子账号开启和操作 SQL 日志审计功能前，需要先授予日志服务相关权限。详细步骤请参考子账号授权。

背景信息

SQL 审计功能默认为关闭状态。您可以参考本文档开启该功能。开启后会产生额外费用，详细收费标准请参见计费方式。如果您需要关闭该功能，请在 DRDS 控制台关闭。关闭后不再写入数据，可以查看历史数据，这部分历史数据会产生存储和索引费用。您可以在日志服务控制台删除 Logstore 以删除历史数据，删除后日志服务不再对该部分数据计费。

当前支持的区域：华东1（杭州）、华东2（上海）、华北1（青岛）、华北2（北京）、华南1（深圳）、新加坡。其它地区陆续开放中。

操作步骤

1. 登录 DRDS 控制台，并选择地域。
2. 在实例列表页面单击实例名称。
3. 在左侧导航栏中单击 SQL 审计与分析。
4. 展开下拉框，选择需要开启 SQL 审计与分析的数据库。
5. 请根据页面提示进行授权。授权后，DRDS 有权限将日志数据写入日志服务。
6. 在日志状态一栏或下拉框中状态栏打开功能开关。

7. 确认是否导入历史数据。

   默认情况下，只对开启 SQL 日志审计功能之后产生的日志数据进行审计分析。

   如果您发现 DRDS 数据库的数据被修改，但是并未开启 SQL 审计功能，可以在开启功能时导入历史数据，将过去发生的日志加入审计分析范围，追溯数据篡改者。 DRDS 会根据您 DRDS 实例节点上的日志存储情况，动态检测支持导入的历史数据范围，目前最多支持导入七天内的数据。

   - 如果需要导入历史数据，请打开功能开关，并指定回溯开始时间和回溯结束时间，并单击启用。 导入历史数据功能基于 DRDS 节点保存的日志时间动态检测，最大支持导入七天的历史数据。 导入过程涉及文件操作，需要一定的时间，请耐心等待。您可以在控制台右上角的任务列表中查看日志回溯的进度。
   - 如果不需要导入历史数据，请直接单击启用。

执行结果

执行结果

SQL 审计功能已成功开启，相同 Region 下的 DRDS 数据库的审计日志会写入同一个日志服务的 Logstore 中。同时，日志服务为您创建以下默认配置。

说明 请勿随意删除或修改日志服务为您默认创建的 Project、Logstore、索引和仪表盘设置，日志服务会不定期更新与升级 SQL 日志审计功能，专属日志库的索引与默认报表也会自动更新。

表 1. 默认配置

默认配置项	配置内容

Project	默认为您创建 Project `drds-audit-区域名-阿里云账户ID`，例如 `drds-audit-cn-qingdao-12345678`。
Logstore	默认为您创建 Logstore `drds-audit-log`。
地域	相同 Region 下的 DRDS 数据库的审计日志会写入同一个日志服务的 Logstore 中。 例如，华东1（杭州）地域中所有可用区的 DRDS 数据库审计日志都会写入日志服务的 Logstore `drds-log-cn-hangzhou-阿里云账户ID` 中。
Shard	默认为您创建5个 Shard，并开启 自动分裂 Shard 功能。
索引	日志服务自动为您创建索引。您的索引设置会随着功能升级自动更新，请勿手动修改或删除索引。
日志存储时间	默认保存45天。45天以上的日志自动删除。 若您需要更长的存储时间，可以自定义修改。详细步骤请参考 高级管理。
仪表盘	默认创建三个仪表盘，分别为： • 运营中心 • 性能中心 • 安全中心 关于仪表盘的更多信息，请参考 日志报表**说明** 请勿修改默认的仪表盘配置，默认仪表盘会随着功能升级自动更新。您可以参考 仪表盘另外创建新的仪表盘。

7.3. 日志字段

本文将介绍PolarDB-X 1.0 SQL审计日志中包含的日志字段详情。

字段名称	字段说明	支持的实例版本
`__topic__`	日志主题，格式为 `drds_audit_log_{instance_id}_{db_name}`，例如 `drds_audit_log_drdsxyzabcd_demo_drds_db`。	所有版本
`instance_id`	PolarDB-X 1.0实例ID。	所有版本
`db_name`	PolarDB-X 1.0数据库名称。	所有版本
`user`	执行SQL的用户名。	所有版本
`client_ip`	访问PolarDB-X 1.0实例的客户端IP。	所有版本
`client_port`	访问PolarDB-X 1.0实例的客户端端口。	所有版本
`sql`	被执行的SQL语句。	所有版本

字段名称	字段说明	支持的实例版本
trace_id	SQL执行的TRACE ID。 如果是事务的话，会以TRACE ID加 - 作为前缀+数字表示，例如 drdsabcdxyz-1 ， drdsabcdxyz-2 等。	所有版本
sql_code	模板SQL的HASH值。	所有版本
hint	SQL执行的HINT。	所有版本
table_name	查询涉及的表名，多个表之间用英文逗号（,）分隔。	所有版本
sql_type	SQL类型，包括SELECT、INSERT、DELECT、SET、ALTER、CREATE、DROP、TRUNCATE、REPLACE和OTHER类型。	所有版本
sql_type_detail	SQL解析器的名称。	所有版本
sql_time	SQL开始执行的时间，格式为 yyyy-MM-dd HH:mm:ss.SSS 。	所有版本
response_time	响应时间，单位为毫秒（ms）。	实例版本需为5.3.4-15378085或以上
affect_rows	SQL执行返回行数。 • 若执行的是增删改等语句时，表示的时影响的行数。 • 若执行的是查询语句时，表示的是返回的行数。	实例版本需为5.3.4-15378085或以上
fail	SQL执行是否出错，取值范围如下： • 0：成功 • 1：失败	实例版本需为5.3.4-15378085或以上

7.4. 日志分析

PolarDB-X支持SQL审计与分析功能，依托日志服务产品，提供强大的日志分析能力。本文档将介绍常见场景的SQL日志分析语句及示例。

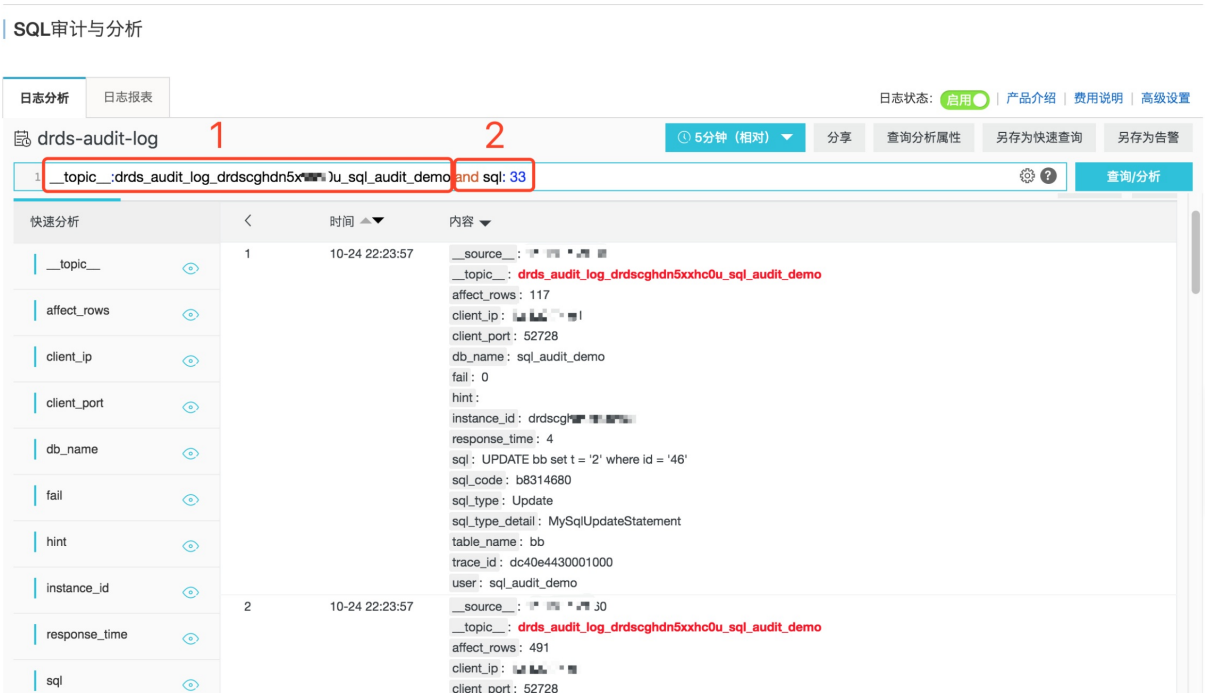
背景信息

开启PolarDB-X SQL审计与分析功能之后，您可以在当前页面通过日志服务的查询分析语法进行SQL 审计与分析。结合日志服务的查询分析语法，在日志分析页面，您可以快速定位问题 SQL，并针对PolarDB-X数据库的SQL执行状况、性能指标，安全问题进行分析。关于日志服务的查询分析语法，详情请参见查询语法分析概述。

注意事项

相同Region下，PolarDB-X数据库的审计日志都是写入同一个日志服务的Logstore里，所以PolarDB-X SQL审计与分析的搜索页面会默认为您带上按照__topic__的过滤条件，保证您搜索到的SQL日志是PolarDB-X数据库的。因此本文提供的所有的查询语句，都需要在已有的过滤条件后追加使用。

例如下图中序号1部分的语句为默认过滤条件，序号2部分的语句为追加的过滤条件。



快速定位问题SQL

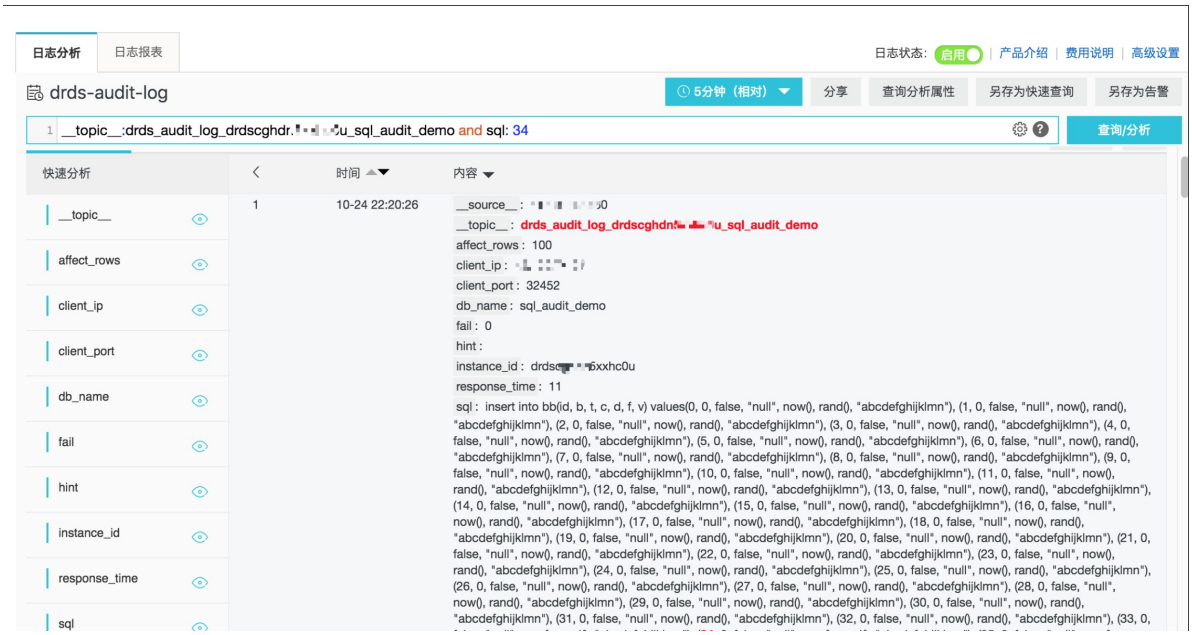
您可以使用以下命令快速定位问题SQL。

- 模糊搜索

  例如，您可以使用如下命令查询包含关键字为 `34` 的SQL语句：

  ```
  and sql: 34
  ```

  查询结果如下图所示：



- 字段搜索

  依赖预置的索引字段，PolarDB-X SQL审计还支持根据字段搜索。例如您可以使用如下命令查询 `Drop` 类

型的SQL：

```
and sql_type:Drop
```

查询结果如下图所示：

日志服务支持鼠标点击自动生成查询语句，如下图所示。

- 多条件搜索

  您可以通过 `and` 或 `or` 等关键字实现多条件的搜索。例如您可以使用如下命令查询对 `id=34` 行的删除操作：

  ```
  and sql:34 and sql_type: Delete
  ```

- 数值比较搜索

  索引字段中的 `affect_rows` 和 `response_time` 是数值类型，支持比较操作符。例如您可以使用如下命令查询 `response_time` 大于1s的Insert SQL：

```
and response_time > 1507 and sql_type: Insert
```

或者使用如下命令查询删除100行以上数据的SQL:

```
and affect_rows > 100 and sql_type: Delete
```

SQL执行状况分析

您可以使用以下命令查看SQL执行状况。

- SQL执行失败率

  您可以使用以下命令查询SQL执行的失败率:

  ```
  | SELECT sum(case when fail = 1 then 1 else 0 end) * 1.0 / count(1) as fail_ratio
  ```

  查询结果如下图所示:

  如果您的业务对SQL错误率敏感，可以在此查询结果的基础上，定制报警信息，单击下图中1所示的另存为告警。以下报警设置表示每隔15分钟，检查15分钟内SQL执行的错误率大于0.01的日志数量。您也可以根据业务需要定制告警。

- SQL累计查询行数

  您可以使用如下命令查询Select语句累计查询的行数：

  ```
  and sql_type: Select | SELECT sum(affect_rows)
  ```

- SQL类型分布

  您可以使用如下命令查询SQL类型分步：

  ```
  | SELECT  sql_type, count(sql) as times GROUP BY sql_type
  ```

- SQL独立用户 IP 分布

  您可以使用如下命令查询SQL独立用户的IP地址分布：

  ```
  | SELECT  user, client_ip, count(sql) as times GROUP BY user, client_ip
  ```

SQL性能分析

您可以使用以下命令查看SQL性能分析详情。

- SELECT平均耗时

  您可以使用以下命令查询SELECT语句的平均耗时：

  ```
  and sql_type: Select | SELECT avg(response_time)
  ```

- SQL执行耗时分布

  您可以使用以下命令查询SQL执行耗时分布。

  ```
  and response_time > 0 | select  case  when response_time <= 10 then '<=10毫秒'  when response_time > 10 and response_time <= 100 then '10~100毫秒'  when response_time > 100 and response_time <= 1000 then '100毫秒~1秒'  when response_time > 1000 and response_time <= 10000  then '1秒~10秒'  when response_time > 10000 and response_time <= 60000  then '10秒~1分钟'  else '>1分钟' end as latency_type,  count(1) as cnt group by latency_type order by latency_type DESC
  ```

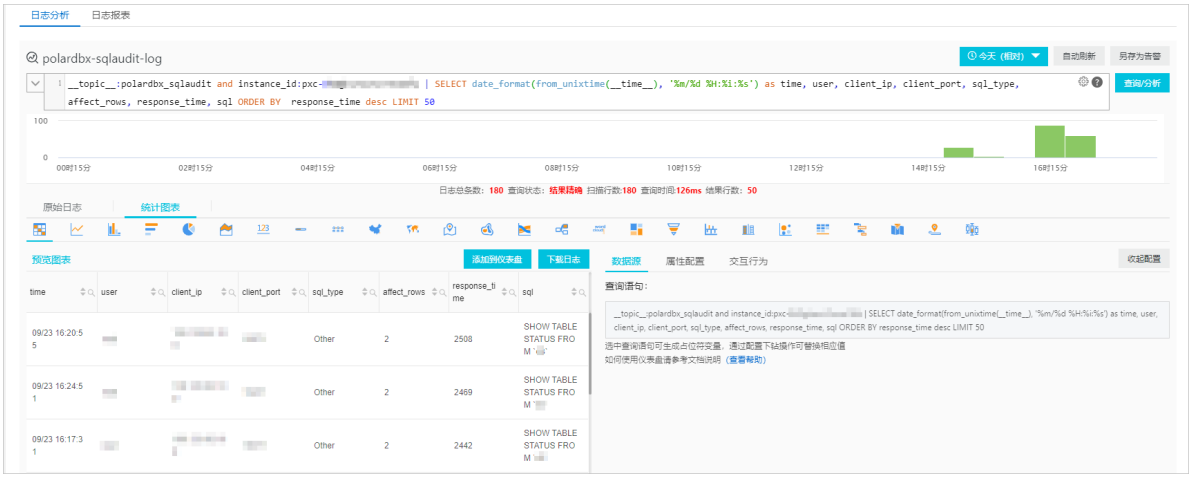
  > **说明** 上述查询给出了按照给定时间段的SQL执行时间分布，您也可以调整时间段的范围，获取更加精细的结果。

- 慢SQL Top 50

  您可以使用以下命令查询系统排名前50的慢SQL

  ```
  | SELECT date_format(from_unixtime(__time__), '%m/%d %H:%i:%s') as time, user, client_ip, client_port, sql_type, affect_rows, response_time, sql ORDER BY  response_time desc LIMIT 50
  ```

  查询结果如下图所示，结果中包含SQL执行时间、执行的用户名、IP 地址、端口号、SQL 类型、影响行数、执行时间以及SQL的文本。

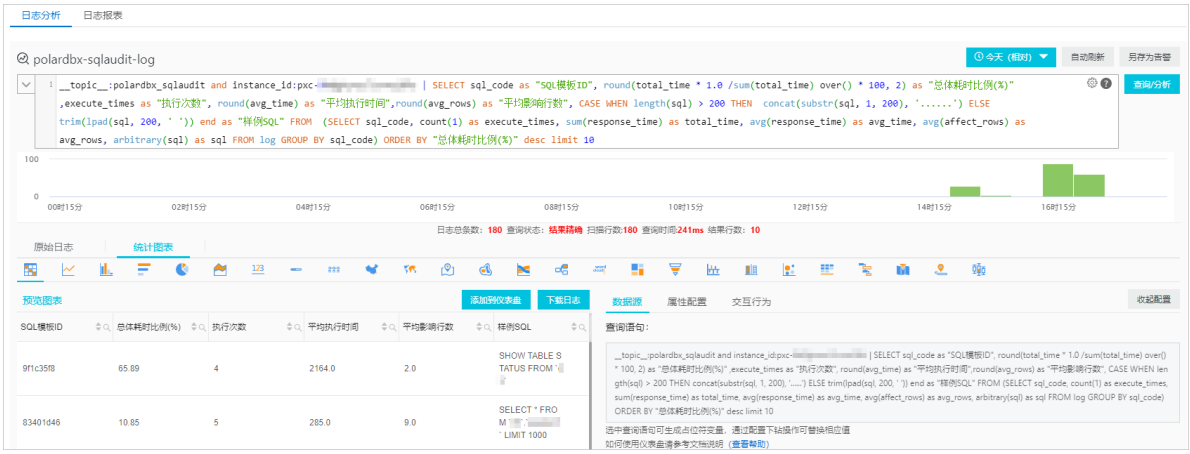
  

- 高代价SQL模板Top 10

  在大多数应用中，SQL通常基于若干模板动态生成的，只是参数不同。您可以使用以下命令通过模板ID找到应用中高代价的SQL模板进行分析优化：

  ```
  | SELECT sql_code as "SQL模板ID", round(total_time * 1.0 /sum(total_time) over() * 100, 2) as "总体耗时比例(%)" ,execute_times as "执行次数", round(avg_time) as "平均执行时间",round(avg_rows) as "平均影响行数", CASE WHEN length(sql) > 200 THEN  concat(substr(sql, 1, 200), '.....') ELSE trim(lpad(sql, 200, ' ')) end as "样例SQL" FROM  (SELECT sql_code, count(1) as execute_times, sum(response_time) as total_time, avg(response_time) as avg_time, avg(affect_rows) as avg_rows, arbitrary(sql) as sql FROM log GROUP BY sql_code) ORDER BY "总体耗时比例(%)" desc limit 10
  ```

统计结果包括SQL模板ID，该模板SQL占总体SQL的耗时比例、执行次数、平均执行时间、平均影响行数以及样例 SQL（为了显示效果，该列按照200的长度截断）。



> **说明** 上述查询是按照总体耗时比例排序，当然您也可以根据平均执行时间，执行次数进行排序排查问题。

- 事务平均执行时长

  对于相同事务内的SQL，预置的 `trace_id` 字段前缀相同，后缀为 `'-'+序号` ；非事务的SQL的 `trace_id` 中则不包含 `'-'` 。因此，您可以使用如下命令对事务SQL的性能进行相关分析。

  > **说明** 由于事务分析涉及前缀匹配操作，查询效率会低于其它类型的查询操作。

  - 查询事务的平均执行耗时：

    ```
    | SELECT  sum(response_time) / COUNT(DISTINCT substr(trace_id, 1, strpos(trace_id, '-') - 1)) where strpos(trace_id, '-') > 0
    ```

  - 慢事务Top 10

    可以按照事务的执行时间排序查询慢事务的列表：

    ```
    | SELECT substr(trace_id, 1, strpos(trace_id, '-') - 1) as "事务ID" , sum(response_time) as "事务耗时" where strpos(trace_id, '-') > 0 GROUP BY substr(trace_id, 1, strpos(trace_id, '-') - 1) ORDER BY "事务耗时" DESC LIMIT 10
    ```

    在此基础上，可以根据查到的慢事务ID，搜索该事务下的所有SQL，分析执行慢的具体原因，查询语句如下：

    ```
    and trace_id: db3226a20402000*
    ```

  - 大批量操作事务Top 10

    按照事务内SQL影响的行数，可以获取大批量操作的事务列表，查询语句如下：

    ```
    | SELECT substr(trace_id, 1, strpos(trace_id, '-') - 1) as  "事务ID" , sum(affect_rows) as "影响行数" where strpos(trace_id, '-') > 0 GROUP BY substr(trace_id, 1, strpos(trace_id, '-') - 1) ORDER BY "影响行数" DESC LIMIT 10
    ```

SQL安全性分析

您可以使用以下命令查看SQL安全性分析情况。

- 错误SQL类型分布

  您可以使用如下命令查看错误SQL类型分布：

  ```
  and fail > 0 | select sql_type, count(1) as "错误次数" group by sql_type
  ```

- 高危SQL列表

  高危SQL是指DROP或RUNCATE类型的SQL，您也可以根据需求增加更多条件。

  ```
  and sql_type: Drop OR  sql_type: Truncate
  ```

- 大批量删除SQL列表

  您可以使用如下命令大批量删除SQL列表：

  ```
  and affect_rows > 100 and sql_type: Delete | SELECT date_format(from_unixtime(__time__), '%m/%d %H:%i:%s') as time, user, client_ip, client_port, affect_rows, sql ORDER BY  affect_rows desc LIMIT 50
  ```

7.5. 日志报表

PolarDB-X SQL审计与分析，依托日志服务，为您提供开箱即用的报表，包括运营中心、性能中心、安全中心，让您对PolarDB-X数据库的执行状况、性能指标、潜在安全问题了如指掌。

开启SQL日志审计之后，在当前页面单击进入日志报表页签，即可查看日志服务提供的报表页面，包括运营中心、性能中心和安全中心。

说明

- 由于相同地区的 PolarDB-X 数据库的审计日志均写入日志服务同一个 Logstore 中，查看当前 PolarDB-X 数据库的报表数据时，默认为您添加基于 `__topic__:drds_audit_log_实例ID_数据库名` 的过滤条件，表示查看当前数据库的数据。例如，`__topic__:drds_audit_log_drdsxyzabcd_demo_drds_db` 。
- PolarDB-X 实例版本如果早于版本5.3.4-15378085，SQL 日志中会缺少相关字段（关于日志字段说明请参见日志字段），日志报表页面只提供简化版的运营中心。如需使用完整版的报表，请升级到最新版本。

图 1. 查看报表

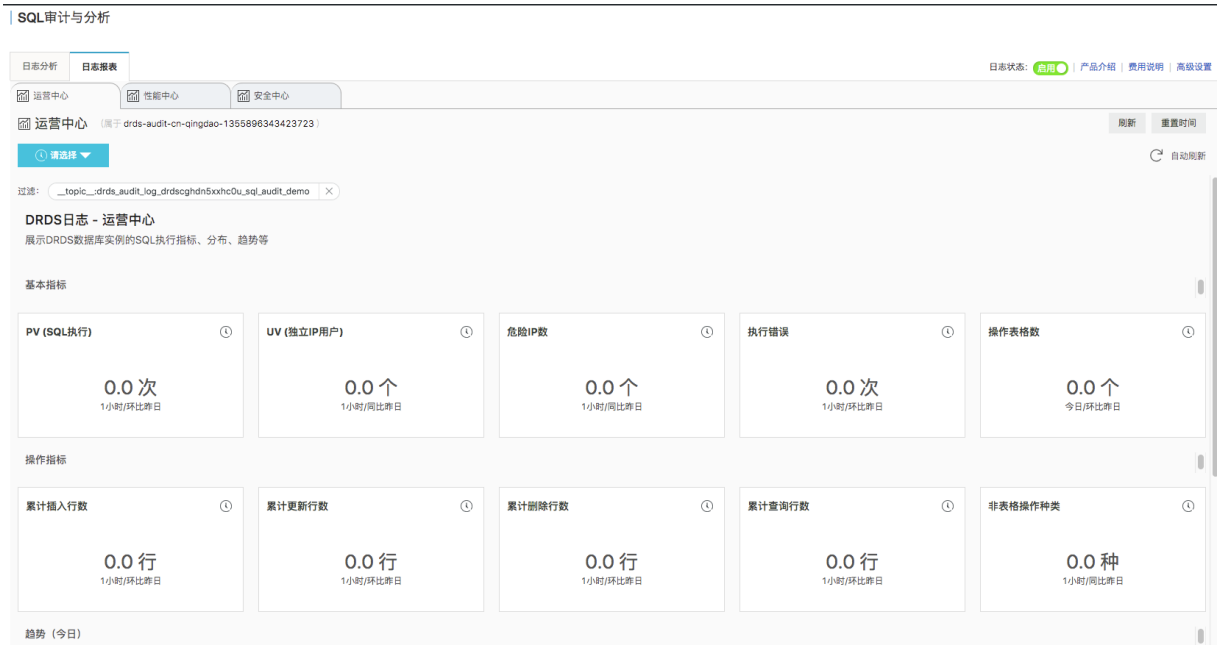


image.png | left | 827x437

日志报表页面的所有图表都是基于不同时间段的数据统计结果，您可以根据需求修改时间范围页。时间范围的修改既可面向所有图表，也可以针对单一图表。

- 单击时间选择器（图中1位置），可以在弹出的时间选择控件中修改当前页面所有图表的时间范围。
- 单击图表的时间选择器（图中2位置），可以修改当前图表的时间范围。

图 2. 时间选择器

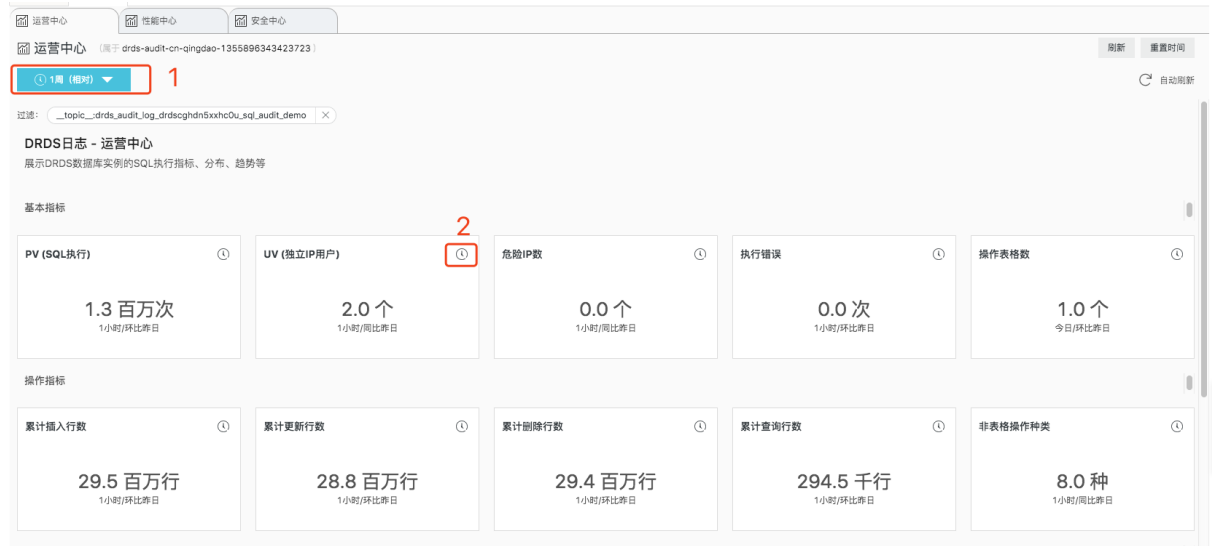


image.png | left | 827x373

运营中心展示 PolarDB-X 数据库的 SQL 执行指标、分布、趋势等。

图表	类型	默认时间范围	描述
PV（SQL执行）	单值	1小时（相对）	SQL 执行的次数
UV（独立 IP 用户）	单值	1小时（相对）	独立的用户及 IP 数目

图表	类型	默认时间范围	描述
危险 IP 数	单值	1小时（相对）	危险 IP 的数目，危险 IP 的定义详见：安全检测函数
执行错误	单值	1小时（相对）	执行错误的 SQL 数目
操作表格数	单值	1小时（相对）	SQL 操作的表格总数
累计插入行数	单值	1小时（相对）	插入操作累计插入的数据行数
累计更新行数	单值	1小时（相对）	更新操作累计更新的数据行数
累计删除行数	单值	1小时（相对）	删除操作累计删除的数据行数
累计查询行数	单值	1小时（相对）	查询操作累计返回的数据行数
非表格操作种类	单值	1小时（相对）	非表格操作的 SQL 种类，例如 show variables like
SQL 执行趋势	柱状图	1小时（相对）	SQL 执行的趋势分布以及对应的错误 SQL 的分布趋势
操作表格	流图	1小时（相对）	SQL 操作表格的分布情况
SQL 类型	流图	1小时（相对）	SQL 类型的按照时间的分布情况
操作用户分布	饼图	1小时（相对）	执行 SQL 用户的分布情况
SQL 执行类型分布	面积图	1小时（相对）	当前时间范围内 SQL 类型的比例
操作最多的表格 Top 50	表格	1小时（相对）	操作最多的表格列表，包括表格的名称以及对应的读、删、改、插的次数
执行分布（世界）	地图	1小时（相对）	执行 SQL 的 client IP 在世界地图上的分布情况
执行分布（中国）	地图	1小时（相对）	执行 SQL 的 client IP 在中国地图上的分布情况

性能中心

性能中心展示 PolarDB-X 数据库的性能指标、快慢分布、慢 SQL、高代价 SQL 的具体分布与来源等。

图表	类型	默认时间范围	描述
SQL 执行峰值	单值	1小时（相对）	每秒 SQL 执行条数的峰值
查询带宽峰值	单值	1小时（相对）	每秒查询 SQL 返回行数的峰值
插入带宽峰值	单值	1小时（相对）	每秒插入 SQL 插入的行数峰值
更新带宽峰值	单值	1小时（相对）	每秒更新 SQL 更新的行数峰值
删除带宽峰值	单值	1小时（相对）	每秒删除 SQL 删除的行数峰值
平均执行时间	单值	1小时（相对）	SQL 平均的执行时间
查询 SQL	单值	1小时（相对）	平均每秒查询 SQL 执行的条数
插入 SQL	单值	1小时（相对）	平均每秒插入 SQL 执行的条数
更新 SQL	单值	1小时（相对）	平均每秒更新 SQL 执行的条数
删除 SQL	单值	1小时（相对）	平均每秒删除 SQL 执行的条数
查询更新带宽趋势	折线图	1小时（相对）	查询 SQL、更新 SQL 操作行数随时间的分布情况
SQL 执行时间分布	饼图	1小时（相对）	SQL 执行时间的分布情况
慢 SQL 表格分布	饼图	1小时（相对）	慢 SQL（执行时间超过1s）的表格分布情况
慢 SQL 用户分布	饼图	1小时（相对）	慢 SQL（执行时间超过1s）的用户分布情况
慢 SQL 类型分布	饼图	1小时（相对）	慢 SQL（执行时间超过1s）的类型分布情况
慢 SQL 列表 Top 50	表格	1小时（相对）	慢 SQL（执行时间超过1s）的列表，包括：时间、客户端、时间、PolarDB-X实例、数据库、表格、用户、影响行数、SQL 类型、SQL 文本

图表	类型	默认时间范围	描述
SQL 模板执行时间 Top20	表格	1小时（相对）	按照 SQL 模板统计该模板 SQL 的执行情况，包括：SQL 模板 ID、总体耗时比例、执行次数、平均执行时间、平均影响行数、样例 SQL
事务执行影响行数 Top20	表格	1小时（相对）	事务影响行数的 Top20 列表，包括：事务 ID、影响行数
事务执行时间 Top20	表格	1小时（相对）	事务执行时间的 Top20 列表，包括：事务 ID、影响行数

安全中心

安全中心展示 PolarDB-X 数据库的失败 SQL、危险 SQL 和大批量删除、修改事件的详情、分布和趋势等。

图表	类型	默认时间范围	描述
错误数	单值	1小时（相对）	失败 SQL 的执行次数
大批量删除事件	单值	1小时（相对）	大批量删除事件（超过100行）的 SQL 数目
大批量修改事件	单值	1小时（相对）	大批量修改事件（超过100行）的 SQL 数
危险 SQL 执行	单值	1小时（相对）	危险 SQL（Drop、Truncate 操作）的数目
危险 IP 数	单值	1小时（相对）	危险 IP 的数目，危险 IP 的定义详见：安全检测函数
错误操作类型分布	面积图	1小时（相对）	失败 SQL 的类型分布
出错客户端外网分布	地图	1小时（相对）	失败 SQL 的客户端在中国地图的分布
错误最多的客户端	表格	1小时（相对）	失败 SQL 的客户端列表，包括IP， 错误数目、错误 SQL 类型、出错 SQL 样例
危险 SQL 执行列表	表格	1小时（相对）	危险 SQL 的列表，包括时间、IP、SQL、PolarDB-X 实例 ID，数据库、表格、用户

图表	类型	默认时间范围	描述
大批量删除事件 Top 50	表格	1小时（相对）	大批量删除 SQL 的列表，包括最早执行时间、最近执行时间、PolarDB-X 实例 ID、数据库、表格、执行次数、平均删除行数、平均时长、样例 SQL
大批量修改事件 Top 50	表格	1小时（相对）	大批量修改 SQL 的列表，包括最早执行时间、最近执行时间、PolarDB-X 实例 ID、数据库、表格、执行次数、平均修改行数、平均时长、样例 SQL

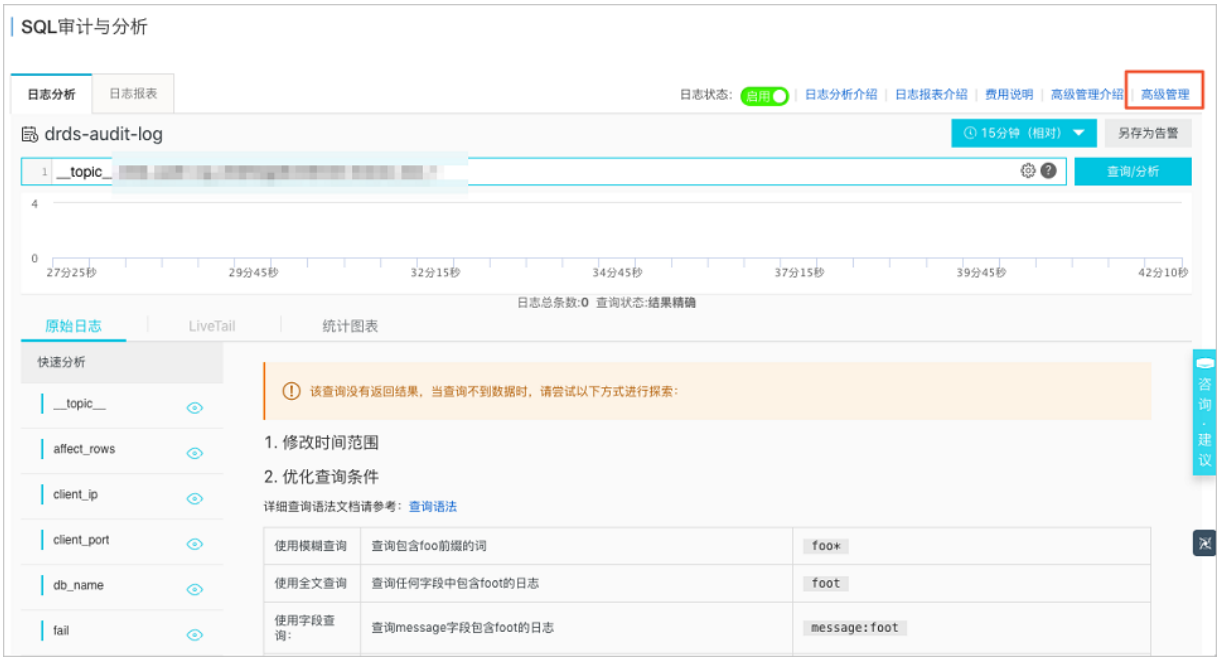
7.6. 高级管理

PolarDB-X SQL审计与分析支持高级管理，您可以通过高级管理跳转到日志服务控制台，修改SQL日志的存储时间、对SQL日志进实时订阅与消费、数据投递和对接其他可视化等高级操作。

背景信息

开启SQL日志审计后，在当前页面右上角单击高级管理可以跳转到日志服务控制台，修改日志存储时间、配置日志消费等高级操作。

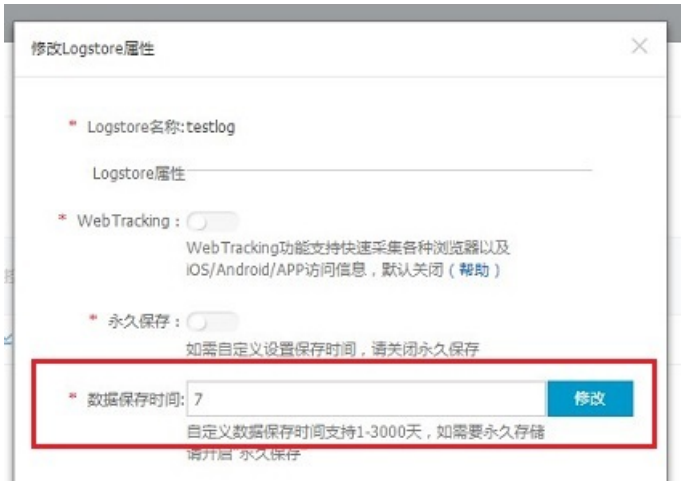
由于相同地区PolarDB-X数据库的SQL审计日志存储在同一Logstore中，如果修改了Logstore的审计日志存储时间，该修改会对该地域下所有PolarDB-X数据库生效。即仅支持修改同一地域所有数据库的日志保存时间，不支持修改某一数据库的日志保存时间。



操作步骤

1. 开启SQL日志审计后，在当前页面右上角单击高级管理跳转到日志服务控制台。

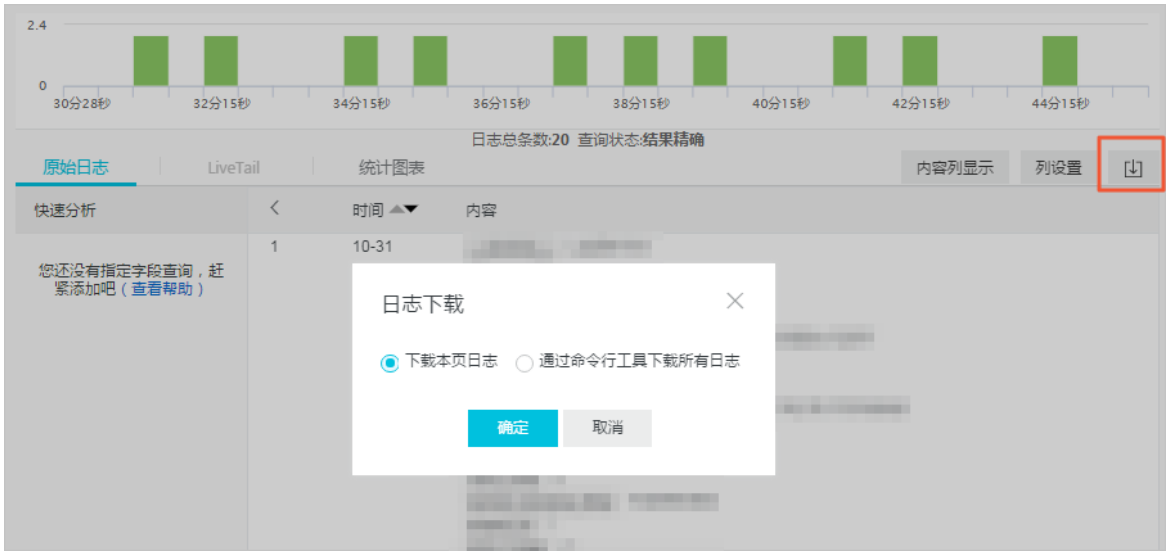
2. 单击操作列的修改。
3. 在弹出页面中修改数据保存时间，并单击修改。



执行结果

导出日志数据

1. 开启SQL日志审计后，单击原始日志页签右侧的日志下载按钮打开日志下载对话框。



2. 在日志下载对话框中单击下载本页日志以CSV格式将本页面的日志保存到本地。

3. 您也可以单击通过命令行工具下载所有日志。

   i. 单击下载日志对话框中的命令行工具CLI用户手册。
   ii. 安装命令行工具。
   iii. 单击安全信息管理页面链接查看并复制当前用户的密钥ID和KEY。
   iv. 单击复制命令行并用当前用户的密钥ID和KEY替换该命令行中 `【步骤2中的密钥ID】` 和 `【步骤2中的密钥Key】` 。
   v. 在CLI命令行工具中执行该命令。命令执行后，PolarDB-X SQL审计日志将自动下载并保存到运行命令的当前目录下的 `download\_data.txt` 文件中。

为子账号授权

详情请参见子账号授权为子账号赋予开启或操作SQL审计分析功能的权限。

其他高级操作

- 告警与通知
- 实时订阅与消费
- 数据投递
- 对接其他可视化

7.7. 为子账号授予SQL审计权限

子账号开通或使用PolarDB-X SQL 审计功能之前，需要由主账号为其授权。

背景信息

使用日志服务实时日志分析，需要如下权限：

操作类型	支持的操作账号类型
开通日志服务（全局一次性操作）	主账号
授权PolarDB-X写入数据到您的日志服务 （全局一次性操作）	• 主账号 • 具备 AliyunLogFullAccess 权限的子账号 • 具备指定权限的子账号
使用日志分析功能	• 主账号 • 具备 AliyunLogFullAccess 权限的子账号 • 具备指定权限的子账号

您也可以根据需求为子账号授权。

- 为子账号授予日志服务的全部操作权限：授予全部管理权限 AliyunLogFullAccess 。详情请参见RAM 用户。
- 主账号开启PolarDB-X SQL审计分析后，为子账号授予日志查看的权限：授予只读权限 AliyunLogReadOnlyAccess 。详情请参见RAM 用户。
- 仅为子账号授予开通及及使用PolarDB-X SQL 审计与分析的权限，不授予其他日志服务管理权限：创建自定义授权策略，并为子账号授予该自定义授权策略。

操作步骤

操作步骤

1. 登录 RAM 控制台。
2. 在策略管理中打开自定义授权策略页签。
3. 在页面右上角单击新建授权策略。
4. 单击空白模板，并输入策略名称和策略内容。 请替换参数后，输入以下策略内容。您也可以添加其他自定义授权内容。

   说明 请将 ${Project} 与 ${Logstore} 替换为您的日志服务 Project 和 Logstore 名称。

```
{


"Version"
:

"1"
,
```

```
"Statement"
:

[

{

"Action"
:

"log:GetProject"
,

"Resource"
:

"acs:log:*:*:project/${Project}"
,

"Effect"
:

"Allow"
```

```
        },

        {

            "Action"
            :
            "log:CreateProject"
            ,

            "Resource"
            :
            "acs:log:*:*:project/*"
            ,

            "Effect"
            :
            "Allow"

        },

        {
```

```
"Action"
:

"log:ListLogStores"
,




"Resource"
:

"acs:log:*:*:project/${Project}/logstore/*"
,




"Effect"
:

"Allow"




},




{




"Action"
:
```

```
"log:CreateLogStore"
,




"Resource"
:
"acs:log:*:*:project/${Project}/logstore/*"
,




"Effect"
:
"Allow"




},




{




"Action"
:
"log:GetIndex"
,
```

```
"Resource"
:
"acs:log:*:*:project/${Project}/logstore/${Logstore}"
,
"Effect"
:
"Allow"
},
{
"Action"
:
"log:CreateIndex"
,
"Resource"
:
"acs:log:*:*:project/${Project}/logstore/${Logstore}"
,
```

```
"Effect"
:

"Allow"




},




{




"Action"
:

"log:UpdateIndex"
,




"Resource"
:

"acs:log:*:*:project/${Project}/logstore/${Logstore}"
,




"Effect"
:

"Allow"
```

```
},

{

"Action"
:
"log:CreateDashboard"
,

"Resource"
:
"acs:log:*:*:project/${Project}/dashboard/*"
,

"Effect"
:
"Allow"

},
```

```
{



"Action"
:
"log:UpdateDashboard"
,



"Resource"
:
"acs:log:*:*:project/${Project}/dashboard/*"
,



"Effect"
:
"Allow"



},



{
```

```
"Action"
:
"log:CreateSavedSearch"
,
"Resource"
:
"acs:log:*:*:project/${Project}/savedsearch/*"
,
"Effect"
:
"Allow"
},
{
"Action"
:
"log:UpdateSavedSearch"
,
```

```
"Resource"
:

"acs:log:*:*:project/${Project}/savedsearch/*"
,




"Effect"
:

"Allow"




}




]




}
```

image.png | left | 744x568

5. 单击新建授权策略。
6. 在左侧导航栏中单击用户管理。
7. 找到子账号并单击对应的授权。
8. 添加上文中创建的自定义授权策略，并单击确定。

8.数据库管理

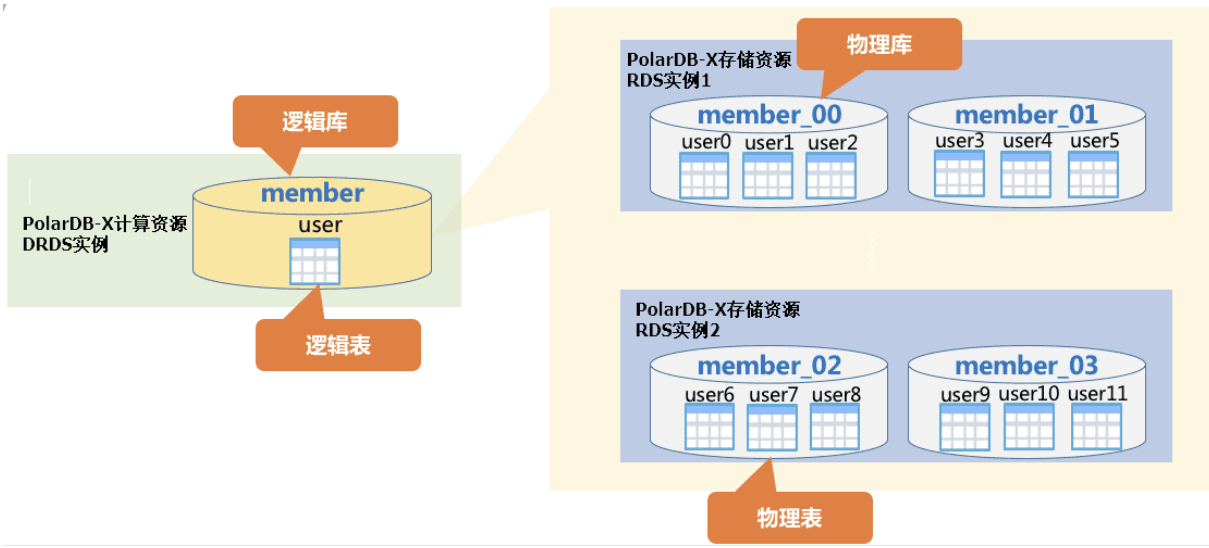
8.1. 概述

本文主要介绍PolarDB-X的数据库详情。

概念上，PolarDB-X数据库类似MySQL和Oracle中的数据库。

使用上，PolarDB-X提供一个完整的数据库使用接口，业务应用程序直接连接至PolarDB-X数据库，与使用单机数据库效果一致。

物理存储上，PolarDB-X底层存储由一个或者多个私有定制RDS实例组成，一个PolarDB-X数据库由底层一个或多个私有定制RDS实例所提供的数据库组成。



针对单个实例，数据库管理包括创建数据库、查看数据库详情、配置读写分离、数据表管理、分库管理、白名单设置、数据导入、只读实例管理、平滑扩容、查看监控信息等功能。

8.2. 查看数据库信息

产品控制台提供查看数据库具体信息、删除数据库、重置密码、只读用户管理等功能。

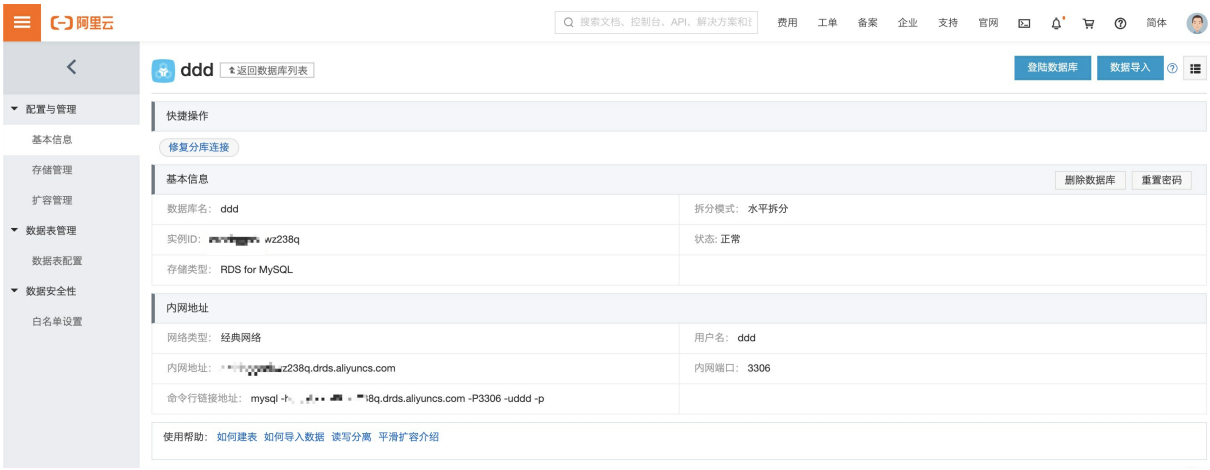
背景信息

具体操作如下：

操作步骤

1. 在产品控制台左侧菜单栏单击**数据库列表**。
2. 在**数据库列表**页选择需要查看的数据库，单击名称进入**基本信息**页。

执行结果



在此页面可以查看数据库名称、状态、工作模式等信息。

其中，命令行链接地址是指登录到数据库的链接地址。由于 PolarDB-X 完全兼容 MySQL 协议，通过 MySQL 客户端可以操作 PolarDB-X 数据库。将**命令行链接地址**复制粘贴到操作系统终端中，输入密码即可登录使用 PolarDB-X 数据库。

注意：

- 一些旧版 MySQL 客户端对用户名长度有限制，不能大于16个字符。 PolarDB-X 使用的库名和用户名相同，建库时库名超过16个字符，则会报错。
- 由于使用 MySQL 客户端，在使用 HINT 时，在命令行上务必添加 -c 参数。 PolarDB-X 的 HINT 是通过注释实现（类似/*...*/）。如果没有加 -c 参数，会丢失注释，从而导致 PolarDB-X 的 HINT 丢失。

8.3. 创建数据库

PolarDB-X 1.0支持RDS MySQL（即私有定制RDS MySQL）和PolarDB MySQL两种存储类型。本文将介绍如何在PolarDB-X 1.0控制台上根据不同存储类型创建对应的数据库。

前提条件

您需要创建实例后才能创建数据库。

> **说明**
>
> - 创建实例时所选择的存储类型将决定您在创建数据库时所使用的存储类型，例如您在创建实例时选择的是RDS MySQL，则您在创建数据库时存储类型即固定为RDS for MySQL不可变更。
> - PolarDB-X 1.0存储层私有定制RDS与计算层DRDS实例配合更加紧密，可以获得更优秀的性能表现和更稳定的服务链路，推荐您选择RDS for MySQL存储类型，详情请参见概述。

构建存储类型为RDS MySQL的数据库

关于如何构建存储类型为RDS MySQL的数据库，详细操作步骤请参见在私有定制RDS实例上构建数据库。

构建存储类型为PolarDB MySQL的数据库

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。

5. 在实例**基本信息**页，单击右上角**创建数据库**。
6. 在弹出的对话框中，根据您的需要选择**拆分模式**并设置相应参数。
   - 若您选择拆分模式为**水平拆分**，请参见如下步骤：
     a. 在**填写基本信息**页，设置如下参数后，单击**下一步**。

参数	说明
拆分模式	选择**水平拆分**。 **说明** PolarDB-X可以将数据按照拆分规则分拆到多个库表中。通过水平拆分，实现数据库的横向扩展，可线性提升数据库整体的存储容量以及并发吞吐。
存储类型	此处的存储类型默认与创建PolarDB-X 1.0计算资源实例时选择的存储类型保持一致。例如您在创建PolarDB-X 1.0计算资源实例时选择了**PolarDB**，则此处的存储类型即固定为 **PolarDB for MySQL**，无需选择。
数据库名	输入数据库名称。 **说明** 数据库名称需满足如下要求： ■ 由小写字母、数字和下划线（_）组成。 ■ 以字母开头，且长度不超过24字符。
字符集	您可以选择字符集为**uft8**、**gbk**、**latin1**或**uft8mb4**。
链接密码	输入数据库链接密码。 **说明** 链接密码需满足如下要求： ■ 密码长度需为8~30个字符。 ■ 需至少包含大写字母、小写字母、数字、下划线（_）中的任意三种。
确认密码	再次输入数据库链接密码进行确认。

     b. 在**选择POLARDB**页，设置如下参数：
     ■ 若您还未购买任何PolarDB MySQL集群，请单击**购买新POLARDB**并参见以下步骤完成购买。

a. 选择**商品类型**为**包年包月**或**按量付费**。

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

b. 设置如下参数。

选项	说明
地域	通过选择地域和可用区配置PolarDB集群所在的物理位置。 **说明** 存储层PolarDB集群和计算层DRDS实例需在同一地域（可用区可以不同）。例如计算层DRDS实例所在地域为华东1（杭州），则此时存储层PolarDB集群的地域取值即固定为华东1（杭州），无需选择。
创建方式	固定为**创建主集群**，无需选择。
网络类型	固定为**专有网络**，无需选择。
VPC网络和VPC交换机	■ 如果您已创建符合您网络规划的VPC，直接选择该VPC。例如，如果您已创建ECS，且该ECS所在的VPC符合您的规划，那么选择该VPC。 ■ 如果您未创建符合您网络规划的VPC，您可以自行创建VPC和交换机，详情请参见使用专有网络。
兼容性	存储层PolarDB集群的MySQL版本需与计算层DRDS实例的MySQL版本保持一致。 **说明** ■ 若计算层DRDS实例使用的是MySQL 8.0版本，则此时存储层PolarDB集群的MySQL版本即固定为**MySQL 8.0**版本，无需选择。 ■ 若计算层DRDS实例使用的是MySQL 5版本，则此时存储层PolarDB集群的MySQL版本即固定为**MySQL 5.6**版本，无需选择。
系列	固定为**标准版**，无需选择。
节点规格	按需选择。所有节点均为独享型，性能稳定可靠。关于各规格的具体信息，请参见计费项概览。

选项	说明
节点个数	■ 默认节点个数为2，无需选择。系统将自动创建一个与主节点规格相同的只读节点。 ■ 如果主节点故障，系统会自动将只读节点切换为新的主节点，并重新生成一个只读节点。 ■ 关于只读节点的更多信息，请参见产品架构。
存储费用	无需选择。系统会根据实际数据使用量按小时计费，详情请参见产品价格。 **说明** 创建集群时无需选择存储容量，存储容量随数据量的增减而自动弹性伸缩。
时区	设置集群时区，默认时区为UTC+08:00。
表名大小写	设置集群表名是否区分大小写，默认**不区分大小写**。当本地数据库区分大小时，您可以选择区分大小写，便于您迁移数据。 **说明** 集群创建后该参数无法修改，请谨慎选择。
删除（释放）集群时	设置删除（释放）集群时的备份保留策略，默认**保留最后一个备份（释放前自动备份）**。 ■ **保留最后一个备份（释放前自动备份）**：删除集群时保留最后一个备份。 ■ **保留全部备份**：删除集群时保留所有备份。 ■ **不保留备份（释放后无法恢复）**：删除集群时不保留任何备份。 **说明** 删除（释放）集群时保留备份可能会产生少量费用，您可以随时删除备份来节省成本，详情请参见计费项概览。
资源组	资源组是单个云账号下将一组相关资源进行统一管理的容器，一个资源只能归属于一个资源组。根据不同的业务场景，您可以将资源组映射为项目、应用或组织等概念，详情请参见RAM资源分组与授权。
购买时长	选择实例购买的时长以及是否开启自动续费。 **说明** 仅当商品类型选择**包年包月**时，支持该参数。

c. 单击**立即购买**。

d. 在**确认订单**页面，确认商品信息并选中服务协议，单击**去支付**。

e. 在**支付**页面，确认待支付订单详情和金额，单击**订购**开通服务。

f. 开通成功后页面会自动跳转至**预检**页面。

- 若您已经购买了PolarDB集群，您可以参见如下步骤进行配置：
  a. 单击**使用已有POLARDB**。
  b. 选中目标PolarDB集群，单击 > 图标将其从左侧待选框移动到右侧已选框中。
  c. 单击**下一步**，进入**预检**页面。
- 若您选择拆分模式为**垂直拆分**，请设置如下参数后，单击**下一步**。

参数	说明
拆分模式	选择**垂直拆分**。 **说明** 您可以批量导入不同PolarDB MySQL集群上已有的多个数据库，利用PolarDB-X 1.0实例级账号进行统一授权管理，通过一个PolarDB-X 1.0链接即可完成横跨多个PolarDB MySQL集群数据库间的联合查询及事务写入操作。
存储类型	此处的存储类型默认与创建PolarDB-X 1.0计算资源实例时选择的存储类型保持一致。例如您在创建PolarDB-X 1.0计算资源实例时选择了**PolarDB**，则此处的存储类型即固定为 **PolarDB for MySQL**，无需选择。
添加数据库	i. 在PolarDB集群待选框中选中目标PolarDB集群。 ii. 从目标PolarDB集群已有的数据库中选中目标数据库，单击 > 将其移动到右边已选框。
字符集	您可以选择字符集为**uft8**、**gbk**、**latin1**或**uft8mb4**。
账号类型	您可以**创建新账号**或**选择已有账号**。
数据库账号	输入需要创建的新账号名称。 **说明** 仅当账号类型选择**创建新账号**时支持该参数。
链接密码	输入数据库链接密码。 **说明** 链接密码需满足如下要求： ○ 密码长度需为8~30个字符。 ○ 需至少包含大写字母、小写字母、数字、下划线（_）中的任意三种。
确认密码	再次输入数据库链接密码进行确认。 **说明** 仅当选择账号类型为**创建新账号**时支持该参数。

7. 在**预检**页面，等待预检结果显示为**成功**，单击**下一步**。
8. 在**建库预览**页面确认建库信息，单击**下一步**完成数据库创建。

> **说明** 当拆分模式为**水平拆分**时，PolarDB-X 1.0为默认为每个PolarDB集群创建8个物理库。此时，在**建库预览**页，您将看到总物理分库数量为已选择的PolarDB集群数量的8倍。

9. 创建完成之后，在**数据库管理**页面，您可以查看已添加的数据库，您还可以在**账号管理**中看到垂直拆分时新建的账号。

9.网络与连接

9.1. 连接数据库

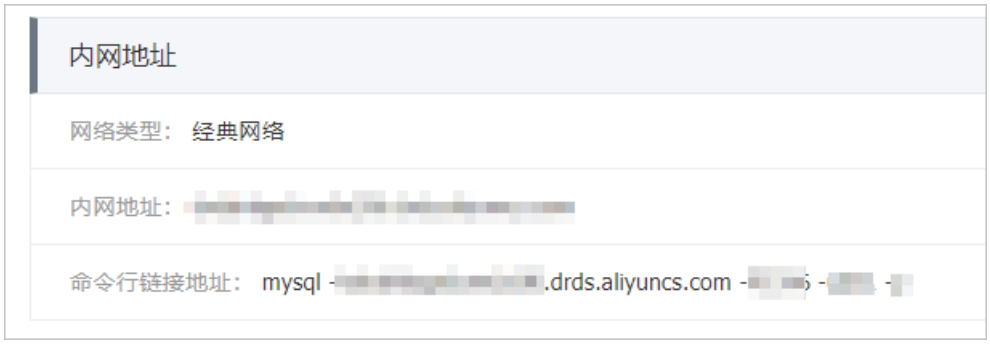
本文介绍如何连接PolarDB-X 1.0数据库并进行SQL操作。

前提条件

您需要创建数据库后才能连接数据库并进行SQL操作。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中，单击**数据库管理**。
6. 在数据库列表页，找到目标数据库，单击数据库ID，进入数据库**基本信息**页。
7. 在**内网地址**区域，找到**命令行链接地址**，获取由 ip(domain) 、 port 、 dbname 和 user 组成的连接信息。



> **说明** 如果您需要通过临时开通公网地址进行访问试用，请参见外网访问。为确保数据库安全，建议您正式访问时关闭公网连接地址。

8. 获取连接信息后，您可以通过如下方式进行连接：
   - 如果您的服务器安装了MySQL，您可以通过如下MySQL命令行连接PolarDB-X 1.0数据库。

     ```
     //MySQL 命令行连接
     mysql -h${DRDS_IP_ADDRESS} -P${DRDS_PORT} -u${user} -p${password} -D${DRDS_DBNAME}
     ```

   - PolarDB-X 1.0支持通过如下第三方客户端进行连接，您可以去对应的官方网站下载客户端。
     - MySQL Workbench（推荐）
     - SQLyog
     - Sequel Pro
     - Navicat for MySQL

> **说明** 第三方GUI客户端可执行基础的数据库操作，包括数据的增删改查和DDL操作，对于工具高级特性，PolarDB-X 1.0可能并不支持。

- PolarDB-X 1.0支持通过如下符合MySQL官方交互协议的第三方程序代码进行连接：
  - JDBC Driver for MySQL (Connector/J)
  - Python Driver for MySQL (Connector/Python)
  - C++ Driver for MySQL (Connector/C++)
  - C Driver for MySQL (Connector/C)
  - ADO.NET Driver for MySQL (Connector/NET)
  - ODBC Driver for MySQL (Connector/ODBC)
  - PHP Drivers for MySQL (mysqli, ext/mysqli, PDO_MYSQL, PHP_MYSQLND)
  - Perl Driver for MySQL (DBD::mysql)
  - Ruby Driver for MySQL (ruby-mysql)

连接示例

- JDBC Driver for MySQL （Connector/J）程序代码示例。

```
//JDBC
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection("jdbc:mysql://drdsxxxxx.drds.aliyuncs.com:3306/doc_test","doc_test","doc_test_password");
//...
conn.close();
```

- 应用端连接池配置示例。

```
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">
<property name="url" value="jdbc:mysql://drdsxxxxx.drds.aliyuncs.com:3306/doc_test" />
<property name="username" value="doc_test" />
<property name="password" value="doc_test_password" />
<property name="filters" value="stat" />
<property name="maxActive" value="100" />
<property name="initialSize" value="20" />
<property name="maxWait" value="60000" />
<property name="minIdle" value="1" />
<property name="timeBetweenEvictionRunsMillis" value="60000" />
<property name="minEvictableIdleTimeMillis" value="300000" />
<property name="testWhileIdle" value="true" />
<property name="testOnBorrow" value="false" />
<property name="testOnReturn" value="false" />
<property name="poolPreparedStatements" value="true" />
<property name="maxOpenPreparedStatements" value="20" />
<property name="asyncInit" value="true" />
</bean>
```

> **说明** 推荐使用Druid连接池连接PolarDB-X 1.0，关于Druid的详细信息请参见Druid Github资源。

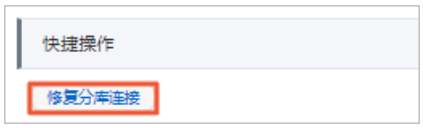
9.2. 修复分库连接

在使用PolarDB-X 1.0数据库过程中需要访问PolarDB-X 1.0存储资源实例，如果存储层RDS的网络配置发生变更（如切换可用区等），PolarDB-X 1.0与存储层RDS实例的网络连通性会被破坏，导致无法访问PolarDB-X 1.0数据库，此时需要修复两者间的网络连接。本文将介绍如何在PolarDB-X 1.0控制台上手动修复分库连接。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中单击**配置与管理 > 数据库管理**。
6. 在数据库列表中找到目标数据库，单击数据库名称。

7. 在目标数据库的**基本信息**页，单击**快捷操作**区域的**修复分库连接**。



8. 在弹出的对话框中，单击**确定**。

9.3. 外网访问

PolarDB-X 1.0实例会默认分配一个内网地址，应用和MySQL客户端可以在阿里云ECS上连接PolarDB-X 1.0实例的数据库。如果需要从非阿里云网络访问PolarDB-X 1.0，则需要为PolarDB-X 1.0实例申请外网地址。本文将介绍如何申请和释放外网地址。

申请外网地址

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。
5. 在**连接信息**区域，单击右上角**申请外网地址**。

6. 在弹出的对话框中，单击**确定**。

> 说明
>
> - 为确保数据安全，请将应用的IP添加到实例中数据库的IP白名单中，详情请参见设置白名单。
> - 为确保数据安全，当外网地址不需要时，请及时释放外网地址。

释放外网地址

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。
5. 在**连接信息**区域，单击右上角**释放外网地址**。

6. 在弹出的对话框中，单击**确定**。

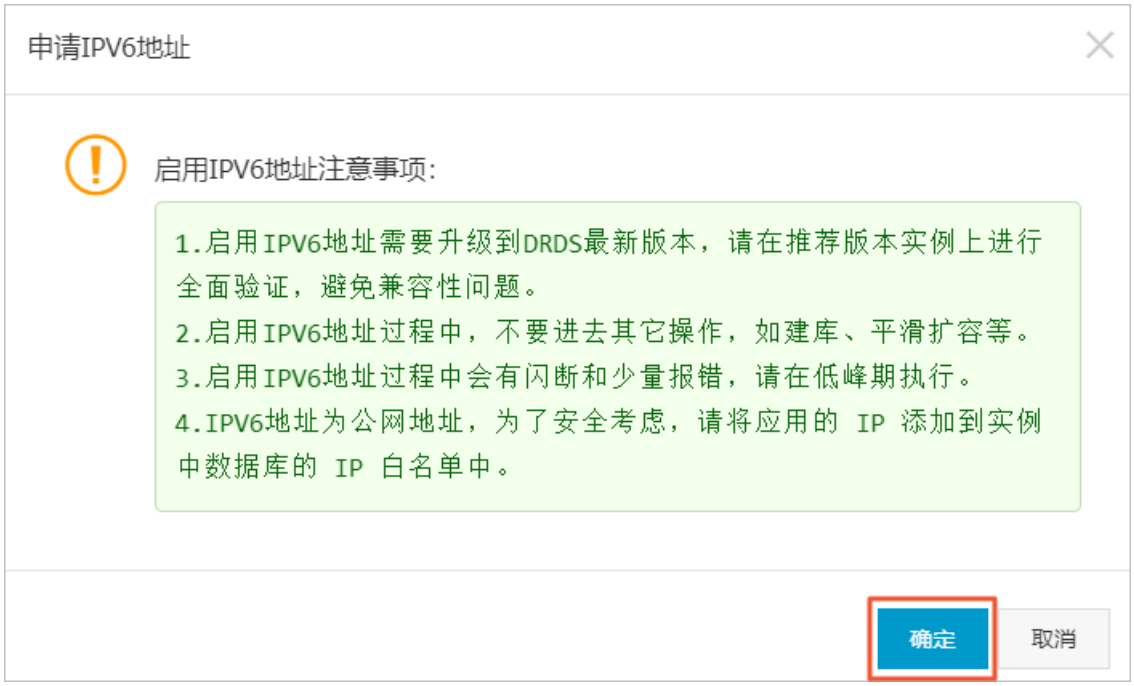
申请IPV6地址

如果您需要通过公网IPV6地址访问PolarDB-X 1.0，则需要为目标实例申请IPV6地址。

> 说明
>
> - 实例所属地域需为华东1（杭州）。
> - 实例版本需为5.3.2-1623342及以上。
> - 只读实例不支持申请IPV6地址。
> - 目前仅支持申请公网IPV6地址。

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。
5. 在**连接信息**区域，单击右上角**申请IPV6地址**。

6. 在弹出的对话框中，单击**确定**。



7. 申请IPV6地址成功后，由于IPV6目前仅支持公网地址，直接使用**连接信息**区域中的**公网地址：**域名访问即可。

10.扩容管理

10.1. 平滑扩容

背景信息

1.

10.2. 热点扩容

背景信息

1.

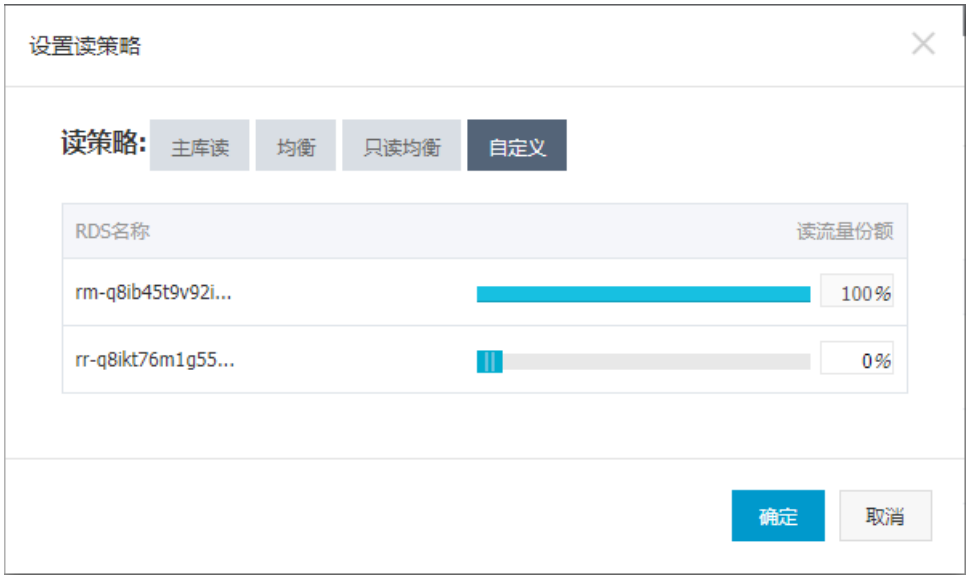
11.存储管理

操作步骤

1. 数据库基本信息页面，单击左侧菜单栏的**存储管理**。进入存储管理页面中可针对专属 RDS 实例进行升降配、添加只读实例、释放、设置读策略等运维操作。

执行结果

1. 如需设置读写分离，需在对应的 RDS 实例右侧单击**设置读策略**，在对话框选择读策略并配置比例，单击**确定**。读写分离的原理介绍请参见读写分离。



注意：

- 读写比例在容量管理页面是以 RDS 实例为单位设置的。如果一个 PolarDB-X 数据库含有多个 RDS 实例，则需要针对每个 RDS 实例设置读写比例。
- RDS 只读实例需要释放时，需要在 PolarDB-X 控制台中把读权重置为0，否则流量会继续走到原有只读实例，导致执行失败。
- 读写分离以 PolarDB-X 数据库为基本单位，如果同一个 RDS 只读实例在多个 PolarDB-X 数据库上使用，需要在每个 PolarDB-X 数据库上都将其权重设置为0。

设置完成后，可以连接 PolarDB-X，执行 SHOW NODE 指令查看实际读流量分布。

```
mysql> show node;
+------+--------------------------------------------------+-------------------+------------------+---------------------+--------------------+
| ID   | NAME                                             | MASTER_READ_COUNT | SLAVE_READ_COUNT | MASTER_READ_PERCENT | SLAVE_READ_PERCENT |
+------+--------------------------------------------------+-------------------+------------------+---------------------+--------------------+
| 0    | [illegible]                                      |                 1 |                0 | 100%                | 0%                 |
| 1    | [illegible]                                      |                 0 |                0 | 0%                  | 0%                 |
| 2    | [illegible]                                      |                 0 |                0 | 0%                  | 0%                 |
| 3    | [illegible]                                      |                 0 |                0 | 0%                  | 0%                 |
| 4    | [illegible]                                      |                 0 |                0 | 0%                  | 0%                 |
| 5    | [illegible]                                      |                 0 |                0 | 0%                  | 0%                 |
| 6    | [illegible]                                      |                 0 |                0 | 0%                  | 0%                 |
| 7    | [illegible]                                      |                 0 |                0 | 0%                  | 0%                 |
+------+--------------------------------------------------+-------------------+------------------+---------------------+--------------------+
8 rows in set (0.01 sec)
```

通过PolarDB-X HINT 可以指定在主实例或只读实例执行 SQL，详情请参见PolarDB-X 自定义HINT。

12.数据表管理

12.1. 调整拆分键

您可以在PolarDB-X控制台上管理数据表，调整数据表的拆分键。

前提条件

您需要先连接数据库并使用SQL语句创建表后，才能在控制台上管理数据表。

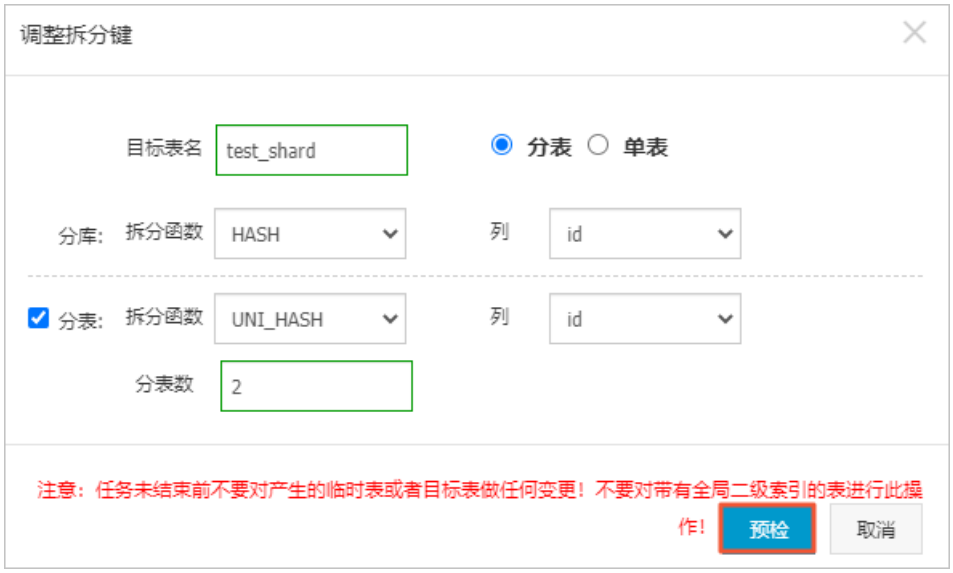
背景信息

在PolarDB-X中，一张逻辑表的拆分方式由拆分函数（包括分片数目与路由算法）与拆分键（包括拆分键的MySQL 数据类型）共同定义。只有当PolarDB-X使用了相同的拆分函数和拆分键时，才会被认为分库与分表使用了相同的拆分方式。相同的拆分方式让PolarDB-X可以根据拆分键的值定位到唯一的物理分库和物理分表。当一张逻辑表的分库拆分方式与分表拆分方式不一致时，若SQL查询没有同时带上分库条件与分表条件，则PolarDB-X在查询过程会进行全分库扫描或全分表扫描操作，详情请参见关闭全表扫描。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中单击**配置与管理 > 数据库管理**。
6. 在数据库列表中找到目标数据库，单击数据库名称。

7. 在左侧导航栏中，单击**数据表管理 > 数据表配置**
8. 找到目标数据表，单击右侧**操作**栏中的**调整拆分键**。
9. 在弹出的对话框中，根据您的业务需求调整拆分键设置，单击**预检**。



> 注意
>
> - 预检任务未结束前不要对产生的临时表或目标表做任何变更。
> - 不要对带有全局二级索引的表进行调整拆分键的操作。更多拆分键的选择和分表形式，请参见CREATE TABLE和拆分函数概述。

10. 在弹出的对话框中，等待预检结果显示成功后，单击**调整**。

12.2. 关闭全表扫描

PolarDB-X默认为新创建的表开启全表扫描功能，但频繁使用全表扫描功能会占用数据库资源降低查询效率，本文将介绍如何在控制台上手动关闭全表扫描。

前提条件

- 您需要先连接数据库并使用SQL语句创建表后，才能在控制台上管理数据表。
- 目标表需已指定拆分键，设置拆分键详细操作步骤请参见调整拆分键。

注意事项

如果PolarDB-X拆分表没有开启全表扫描功能，则在访问该表时必须在WHERE子句中包含拆分键。否则，PolarDB-X将提示TDDL-4510错误代码。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中单击**配置与管理 > 数据库管理**。
6. 在数据库列表中找到目标数据库，单击数据库名称。

7. 在左侧导航栏中，单击**数据表管理 > 数据表配置**
8. 找到目标数据表，单击右侧**操作**栏中的**高级设置**。

> **说明** 仅当表属性为**分库**时支持设置全表扫描功能，您可以通过调整拆分键进行分库拆分。

9. 在弹出的对话框中，选择**否**，单击**确定**。

12.3. 查看表结构

您可以在PolarDB-X控制台上管理数据表，查看数据表的表结构详情。

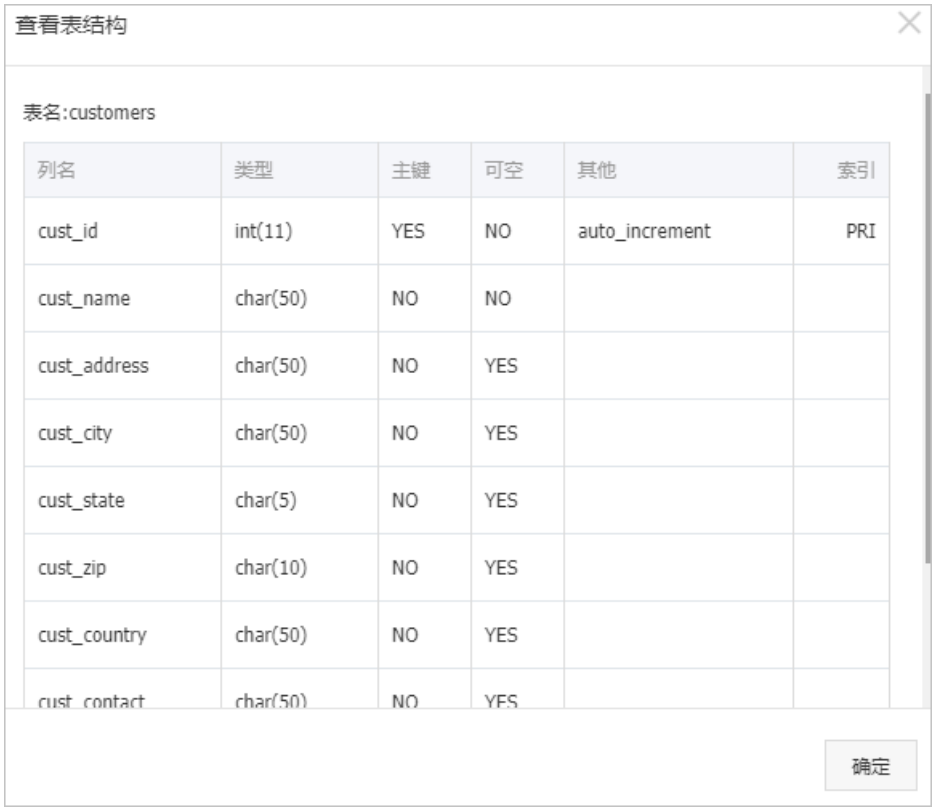
前提条件

您需要先连接数据库并使用SQL语句创建表后，才能在控制台上管理数据表。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID。
5. 在左侧导航栏中单击**配置与管理 > 数据库管理**。
6. 在数据库列表中找到目标数据库，单击数据库名称。

7. 在左侧导航栏中，单击**数据表管理 > 数据表配置**
8. 找到目标数据表，单击右侧**操作**栏中的**查看表结构**，您可以在弹出对话框查看目标数据表的表结构详情（如列名、数据类型、是否为主键等）。



12.4. 删除数据表

本文将介绍如何删除PolarDB-X的数据表。

暂不支持在PolarDB-X控制台上直接删除数据表，您可以登录数据库后使用SQL语句删除目标数据表，详情请参见连接数据库和使用SQL语句删除表。

13.设置白名单

PolarDB-X 1.0支持白名单设置，只有已加入白名单规则的IP才可以访问PolarDB-X 1.0的数据库。本文将介绍如何设置白名单。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例ID，单击实例ID。
5. 单击左侧导航栏**配置与管理 > 数据库管理**。
6. 选中目标数据库，单击数据库名称。
7. 单击左侧导航栏**数据安全性 > 白名单设置**。
8. 单击**手动修改**，在弹出的对话框中，输入允许访问的IP白名单地址。

> **说明**
>
> - 数据库创建完成后，默认允许所有IP地址访问。出于数据安全考虑，建议仅将您的WEB服务器外网IP或IP段加入白名单。
> - 您可以填写IP地址（如192.168.0.1）和IP段（如192.168.0.0/24、192.168.10.*、192.168.10.2-192.168.10.30等）。
> - 每个IP白名单内最多添加300个IP或IP段。
> - 多个IP地址间用英文逗号（,）隔开（如192.168.0.1,192.168.0.0/24）。

9. 单击**确定**。

14.分布式事务

14.1. 基于 MySQL 5.7 的分布式事务

如果您的 MySQL 版本 ≥ 5.7，并且 DRDS 版本 ≥ 5.3.4 时，将会自动开启 XA 分布式事务，使用体验和单机 MySQL 数据库一致，无需特殊指令开启。

背景信息

如果不满足上述要求，请您移步 基于 MySQL 5.6 的分布式事务文档。

分布式事务的原理请参考文档 DRDS 分布式事务基本原理。

对于 MySQL 5.7 及更高版本，DRDS 默认基于 XA 事务协议进行分布式事务。

如何使用？

DRDS 分布式事务使用体验和单机 MySQL 数据库完全一致，例如：

- `SET AUTOCOMMIT=0` 开启一个事务；
- `COMMIT` 提交当前事务；
- `ROLLBACK` 回滚当前事务。

如果事务中的 SQL 仅涉及单个分片，DRDS 会将其作为单机事务直接下发给 MySQL；如果事务中的 SQL 语句修改了多个分片的数据，DRDS 会自动地将当前事务升级为分布式事务。

常见问题

Q: 使用 DRDS 分布式事务需要在控制台开启吗？

A: 不需要显式开启，只要版本达到要求，即可以直接像单机 MySQL 那样使用事务。

Q: DRDS 分布式事务对版本是否有要求？

A: DRDS 新版分布式事务要求 MySQL 版本 ≥ 5.7、DRDS 版本 ≥ 5.3.4。如果您的 MySQL 或 DRDS 版本不满足上述要求，建议您升级到新版，或查看基于 MySQL 5.6 的分布式事务文档。

14.2. 基于 MySQL 5.6 的分布式事务

⚠ **本文描述的分布式事务针对 MySQL 5.6 版本用户，或 DRDS 版本低于 5.3.4 的用户。**

背景信息

分布式事务的原理请参考文档 DRDS 分布式事务基本原理。

对于 MySQL 5.6 版本，由于 MySQL XA 协议实现尚不成熟，DRDS 自主实现了 2PC 事务策略用于分布式事务。MySQL 5.7 及更高版本中，推荐您使用 XA 事务策略。

如何使用

如果某个事务可能涉及多个数据分库，则需要将当前事务声明为分布式事务。如果事务仅涉及单个数据分库，则无需开启分布式事务，直接像 MySQL 单机事务那样即可，无需额外操作。

分布式事务开启方式如下：

在事务开启后，执行 `SET drds_transaction_policy = '...'` 即可。

MySQL 命令行客户端使用示例（以 2PC 事务为例）

```
SET AUTOCOMMIT=0;
SET drds_transaction_policy = '2PC'; -- 建议 MySQL 5.6 用户使用
.... -- 业务 SQL
COMMIT; -- 或 ROLLBACK
```

Java JDBC 代码示例（以 2PC 事务为例）

```
conn.setAutoCommit(false);
try (Statement stmt = conn.createStatement()) {
  stmt.execute("SET drds_transaction_policy = '2PC'");
}
// ... 运行业务 SQL ...
conn.commit(); // 或 rollback()
```

常见问题

如何在 Spring 框架中使用 DRDS 分布式事务？

如果使用 Spring 的 `@Transactional` 注解开启事务，可以通过扩展事务管理器来开启 DRDS 分布式事务。

代码示例：

```
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
public class DrdsTransactionManager extends DataSourceTransactionManager {
  public DrdsTransactionManager(DataSource dataSource) {
    super(dataSource);
  }
  @Override
  protected void prepareTransactionalConnection(Connection con, TransactionDefinition definition) throw
s SQLException {
    try (Statement stmt = con.createStatement()) {
      stmt.executeUpdate("SET drds_transaction_policy = '2PC'"); // 以 2PC 为例
    }
  }
}
```

之后，在 Spring 配置中将上述类实例化，例如：

```
<bean id="drdsTransactionManager" class="my.app.DrdsTransactionManager">
  <property name="dataSource" ref="yourDataSource" />
</bean>
```

对于需要开启 DRDS 分布式事务的类，加上注解 `@Transactional("drdsTransactionManager")` 即可。

15.数据导入导出

15.1. 数据迁移或同步方案概览

本文汇总了PolarDB-X 1.0支持的数据迁移或同步的方案。

> **说明** 下表中√表示支持，×表示不支持。

方案	数据量	全量迁移	增量迁移
DTS • PolarDB-X 1.0实例间的数据实时同步 • 从PolarDB-X 1.0同步至DataHub • 从PolarDB-X 1.0同步至云原生数据仓库AnalyticDB MySQL • 从PolarDB-X 1.0同步至AnalyticDB for PostgreSQL **说明** • 仅基于RDS MySQL创建的PolarDB-X 1.0数据库支持通过DTS进行数据同步。 • AnalyticDB MySQL实例版本需为2.0或3.0。 • AnalyticDB for PostgreSQL实例版本需为4.3或6.0。	大	√	√
	大	×	√
	大	√	√
	大	√	√
评估导入导出 • 从RDS迁移至PolarDB-X 1.0 • 从MyCAT评估导入至PolarDB-X 1.0	大	√	√
	大	√	√
MySQL官方命令 • 通过 `source` 命令将SQL文本文件导入到PolarDB-X 1.0。 • 通过 `mysqldump` 命令从PolarDB-X 1.0导出SQL文本文件 详细操作步骤，请参见使用mysqldump导入导出数据。	小（少于2000万条数据记录）	√	×
编程方式 • 从任意数据库、文本文件导入到PolarDB-X 1.0。 • 从PolarDB-X 1.0导出到任意数据库、文本文件。 详细操作步骤，请参见使用程序进行大数据导入。	大	√	×
数据集成 • 从MaxCompute（原ODPS）导入到PolarDB-X 1.0。 • 从PolarDB-X 1.0导出到MaxCompute（原ODPS）。 详细操作步骤，请参见通过数据集成导入导出数据	大	√	×

15.2. 使用mysqldump导入导出数据

PolarDB-X支持MySQL官方数据导出工具mysqldump。本文将介绍如何使用mysqldump导入导出PolarDB-X数据

注意事项

- 关于mysqldump命令的详细说明，请参见 MySQL 官方文档。
- mysqldump 适合小数据量(低于1000万)的离线导入导出。如果需要完成更大数据量或者实时的数据迁移任务，请参考阿里云提供的数据传输服务。

场景一：从 MySQL 中导出数据到文本文件

1. 从MySQL中导出数据到文本文件。

   输入如下命令，从MySQL导出表结构和数据（假设导出文件为 dump.sql）。

   ```
   mysqldump -h ip -P port -u user -ppassword --default-character-set=char-set --net_buffer_length=10240 --no-create-db --skip-add-locks --skip-lock-tables --skip-tz-utc --set-charset  [--hex-blob] [--no-data] database [table1 table2 table3...] > dump.sql
   ```

   参数说明

参数名	说明	是否必选
ip	DRDS 实例的IP。	✔
port	DRDS 实例的端口。	
user	DRDS 的用户名。	✔
password	DRDS 的密码，注意前面有个 -p，之间没有空格。	✔
char-set	指定的编码。	✔
--hex-blob	使用十六进制格式导出二进制字符串字段。如果有二进制数据就必须使用本选项。影响的字段类型包括BINARY、VARBINARY、BLOB。	
--no-data	不导出数据。	
table	指定导出某个表。默认导出该数据库所有的表。	

2. 修改建表语句。

   从 MySQL导出的数据文件包含每个表的建表语句。如果直接在在 DRDS 上执行这些建表语句，会在 DRDS 上建立一个单表。如果要对某个表进行分库分表，那么需要手工对建表语句进行修改，DRDS 建表语句的语法请参考 CREATE TABLE 语句。

3. 导入数据文件到 DRDS。您可以通过如下两种方式的任意一种导入数据文件到 DRDS。
   - 通过 `mysql -h ip -P port -u user --default-character-set=char-set` 命令登录目标PolarDB-X，执行 `source /yourpath/dump.sql` 命令将数据导入到目标PolarDB-X。

> **说明** 该方式会把所有的步骤回显到屏幕上，速度略慢，但是可以观察导入过程。

- 直接通过 `mysql -h ip -P port -u user --default-character-set=char-set< /yourpath/dump.sql` 命令将数据导入到目标PolarDB-X。

> **说明**
>
> - 上述两个命令中 `default-character-set` 要设置成实际的数据编码。如果是Windows平台，SOURCE命令指定的文件路径需要对分隔符转义。
> - 导入的时候，由于某些 DRDS 和 MySQL 实现上的不同，可能会报错（如 `ERROR 1231 (HY000): [a29ef6461c00000][10.117.207.130:3306][****]Variable @saved_cs_client can't be set to the value of @@character_set_client`）。此类错误信息并不影响导入数据的正确性。

场景二：从一个PolarDB-X导入到另一个PolarDB-X

假设您之前有一个测试环境的PolarDB-X，测试完毕以后，需要把测试过程中的一些表结构和数据导入到生产环境中的PolarDB-X中，那么可以按照如下步骤进行操作。

1. 从源PolarDB-X中导出数据到文本文件。

场景三

操作步骤

1. 导入数据文件到 DRDS。请参考场景一第 3 步。
2. 手工创建 Sequence 对象。

   mysqldump 并不会导出 DRDS 中的 Sequence 对象，所以如果在源 DRDS 中使用了 Sequence 对象，并且需要在目标 DRDS 中继续使用相同的 Sequence 对象，则需要手工在目标 DRDS 中创建同名的 Sequence 的对象。具体步骤如下：

   a. 在源 DRDS 上执行 SHOW SEQUENCES，获取当前 DRDS 中的 Sequence 对象的状态。

   b. 在目标 DRDS 数据库上通过 CREATE SEQUENCE 命令创建新的 Sequence 对象。

   Sequence 命令详情请参考全局唯一数字序列 。

执行结果

场景三：从 DRDS 导出数据到 MySQL

从 DRDS 导出数据到 MySQL，和在 DRDS 之间相互导入数据的过程类似，也分为以下几个步骤。 1. 从源 DRDS 中导出表结构和数据。请参考场景一第 1 步。

1. 手工修改拆分表的 DDL 语句。

   DRDS 中拆分表的建表语句和 MySQL 并不兼容。为了后继导入到 MySQL 中，需手工修改导出的 SQL 文件，删除以下关键字：

   - DBPARTITION BY hash(partition_key):
   - TBPARTITION BY hash(partition_key):
   - TBPARTITIONS N
   - BROADCAST

   例如一个拆分表语句导出如下：

```
CREATE TABLE multi_db_single_tbl
(id int,
name varchar(30),
primary key(id)) dbpartition by hash(id);
```

需修改成以下语句：

```
CREATE TABLE multi_db_single_tbl
(id int,
name varchar(30),
primary key(id));
```

2. 导入修改以后的文件。请参考场景一第 3 步。

15.3. 使用程序进行大数据导入

本文介绍如何通过编写代码的方式，离线导入大数据量到 DRDS 数据库。

背景信息

假设当前数据库有一个表需要导入到 DRDS 数据库中，数据量大致为814万，表结构如下。

```
CREATE TABLE `post` (
 `postingType` int NOT NULL,
 `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
 `acceptedAnswer` bigint(20) DEFAULT NULL,
 `parentId` bigint(20) DEFAULT NULL,
 `score` int DEFAULT NULL
 `tags` varchar(128) DEFAULT NULL,
 PRIMARY KEY (`id`)
);
```

数据库之间大数据量的迁移，建议把原始数据导出成一个文本文件，然后通过程序或者命令的方式导入到目标数据库。

对于上一节的 post 表，可以通过 SELECT INTO 语法将数据从 MySQL 导出到一个名为 stackoverflow.csv 的文件中。在 MySQL 客户端执行以下命令：

```
SELECT postingType,id,acceptedAnswer,parentId,score,tags
INTO OUTFILE '/tmp/stackoverflow.csv'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM test_table;
```

在 DRDS 数据库上建表

由于导出的数据文件不包括表结构，所以需要手工在 DRDS 目标数据库上建立表，并且根据实际情况设置拆分键。

例如以下是按照 id 对 post 表进行分库。

```
CREATE TABLE `post` (
 `postingType` int NOT NULL,
 `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
 `acceptedAnswer` bigint(20) DEFAULT NULL,
 `parentId` bigint(20) DEFAULT NULL,
 `score` int DEFAULT NULL,
 `tags` varchar(128) DEFAULT NULL,
 PRIMARY KEY (`id`)
) DBPARTITION BY hash(id) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

导入数据到 DRDS 数据库

导出数据文件以后，可以通过代码的方式读取文件内容，然后导入到 DRDS 数据库中。为了提高效率，建议通过批量插入的方式。

以下是用 Java 写的一个 Demo。

测试场景：插入8143801条数据，耗时916秒，TPS 在9000左右。

测试客户端配置：i5、8G、SSD。

测试 DRDS 配置：4C4G。

```
public static void main(String[] args) throws IOException, URISyntaxException, ClassNotFoundException,
    SQLException {
    URL url = Main.class.getClassLoader().getResource("stackoverflow.csv");
    File dataFile = new File(url.toURI());
    String sql = "insert into post(postingType,id,acceptedAnswer,parentId,score,tags) values(?,?,?,?,?,?)";
    int batchSize = 10000;
    try (
      Connection connection = getConnection("XXXXX.drds.aliyuncs.com", 3306, "XXXXX",
        "XXXX",
        "XXXX");
      BufferedReader br = new BufferedReader(new FileReader(dataFile))) {
      String line;
      PreparedStatement st = connection.prepareStatement(sql);
      long startTime = System.currentTimeMillis();
      int batchCount = 0;
      while ((line = br.readLine()) != null) {
        String[] data = line.split(",");
        st.setInt(1, Integer.valueOf(data[0]));
        st.setInt(2, Integer.valueOf(data[1]));
        st.setObject(3, "".equals(data[2]) ? null : Integer.valueOf(data[2]));
        st.setObject(4, "".equals(data[3]) ? null : Integer.valueOf(data[3]));
        st.setObject(5, "".equals(data[4]) ? null : Integer.valueOf(data[4]));
        if (data.length >= 6) {
          st.setObject(6, data[5]);
        }
        st.addBatch();
        if (++batchCount % batchSize == 0) {
          st.executeBatch();
          System.out.println(String.format("insert %d record", batchCount));
        }
      }
      if (batchCount % batchSize != 0) {
        st.executeBatch();
```

```
            st.executeBatch();
        }
        long cost = System.currentTimeMillis() - startTime;
        System.out.println(String.format("Take %d second，insert %d record, tps %d", cost/1000,batchCount
, batchCount/(cost/1000) ));
      }
    }
    /**
     * 获取数据库连接
     *
     * @param host    数据库地址
     * @param port    端口
     * @param database 数据库名称
     * @param username 用户名
     * @param password 密码
     * @return
     * @throws ClassNotFoundException
     * @throws SQLException
     */
    private static Connection getConnection(String host, int port, String database, String username, String pa
ssword)
        throws ClassNotFoundException, SQLException {
      Class.forName("com.mysql.jdbc.Driver");
      String url = String.format(
          "jdbc:mysql://%s:%d/%s?autoReconnect=true&socketTimeout=600000&rewriteBatchedStatements=t
rue", host, port,
          database);
      Connection con = DriverManager.getConnection(url, username, password);
      return con;
    }
```

15.4. 通过数据集成导入导出数据

本文介绍如何通过数据集成在PolarDB-X中进行数据导入和导出。 数据集成是阿里巴巴集团提供的数据同步平台。该平台具备可跨异构数据存储系统、可靠、安全、低成本、可弹性扩展等特点，可为20多种数据源提供不同网络环境下的离线（全量或增量）数据进出通道。 使用数据集成，您可以在PolarDB-X完成以下数据同步任务：

背景信息

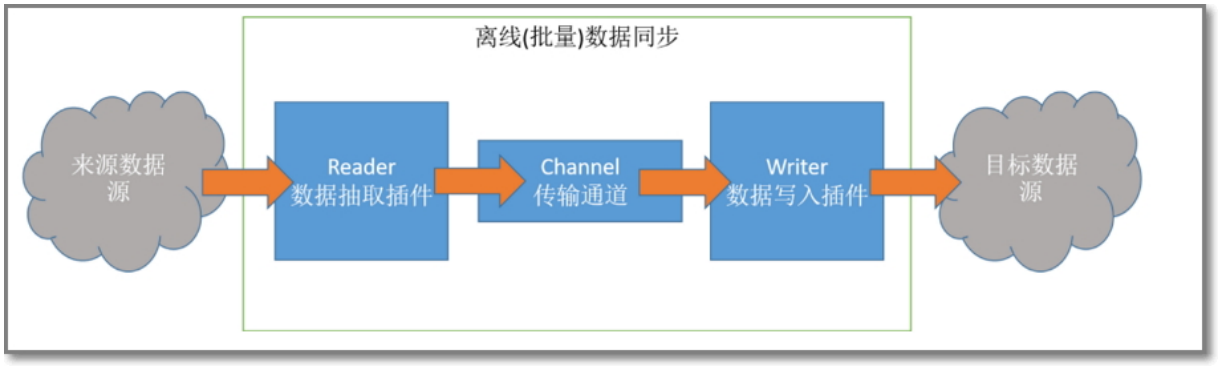
- 将PolarDB-X的数据同步到到其他的数据源里，并将数据进行相应的处理；
- 将处理好的其他数据源数据同步到PolarDB-X。

本文包含以下内容：

- 流程概述
- 准备工作
- 添加数据源
- 通过数据集成导入数据
- 通过数据集成导出数据

数据同步流程主要包含以下几个步骤：

- 第一步：数据源端新建表
- 第二步：添加数据源
- 第三步：向导模式或脚本模式配置同步任务
- 第四步：运行同步任务，检查目标端的数据质量



1

准备工作

使用数据集成在PolarDB-X进行数据导入导出之前，请先注册阿里云账号并参考创建RAM子账号文档，完成以下准备工作：

操作步骤

1. 开通阿里云官网实名认证账号，创建好账号的访问秘钥，即 AccessKeys。
2. 开通 MaxCompute，这样会自动产生一个默认的 ODPS 的数据源，并使用主账号登录大数据开发套件。
3. 创建项目。您可以在项目中协作完成工作流，共同维护数据和任务等，因此使用大数据开发套件之前需要先创建一个项目。
4. 如果想通过子账号创建数据集成任务，可以赋予其相应的权限。

执行结果

新添加数据源

下面以添加PolarDB-X的数据源为例。

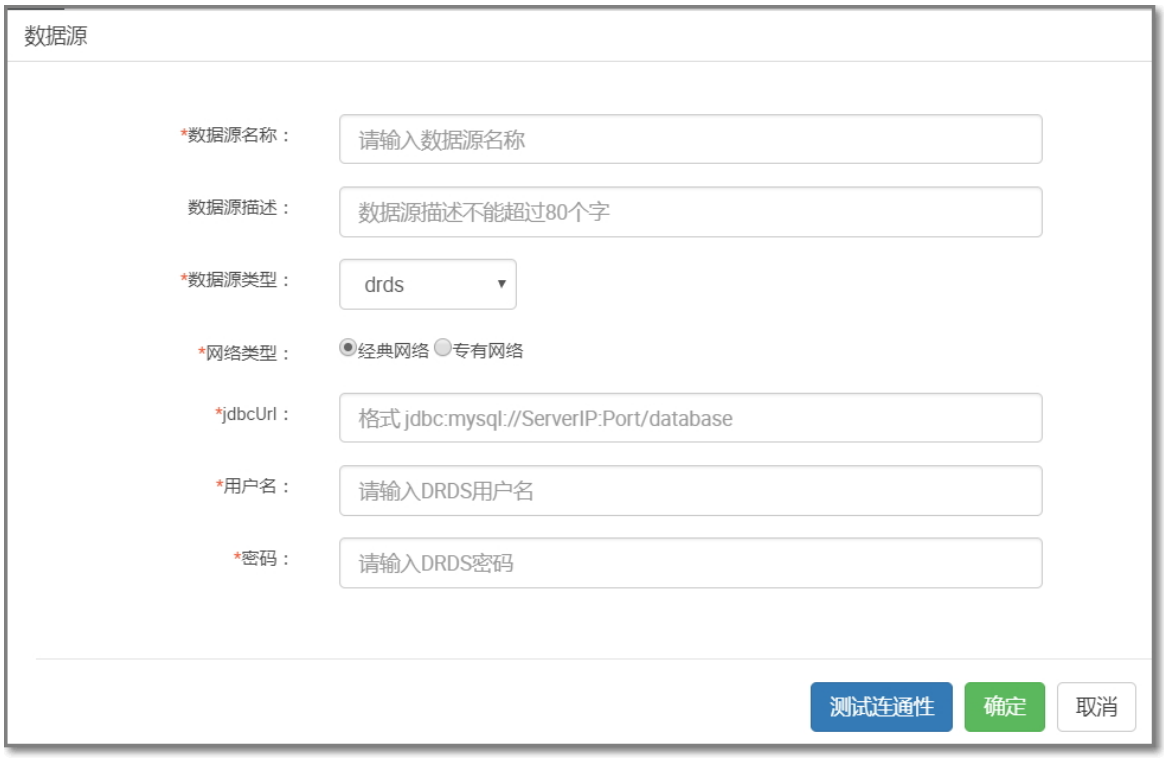
注意：只有项目管理员角色才能够新建数据源，其他角色的成员仅能查看数据源。

1. 以项目管理员身份登录数加管理控制台。
2. 在**项目列表**中对应项目的操作栏单击**进入工作区**。
3. 进入顶部菜单栏中的**数据集成**页面，单击左侧导航栏中的**数据源**。

4. 点击右上角的**新增数据源**，如下图所示：

2

5. 在新增数据源弹出框中填写相关配置项，如下图所示：



3

针对PolarDB-X数据源配置项的具体说明如下：

- 数据源名称：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过 60 个字符 。
- 数据源描述：对数据源进行简单描述，不得超过 80 个字符 。
- 数据源类型：当前选择的数据源类型 DRDS。
- 网络类型：当前选择的网络类型。
- JDBCUrl：JDBC 连接信息，格式为 `jdbc://mysql://serverIP:Port/database` 。
- 用户名/密码：对应的用户名和密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

通过数据集成导入数据

下文以通过数据集成的向导模式将 MaxCompute（原 ODPS）数据同步到PolarDB-X为例。

1. 在数据集成页面，新建同步任务。

4

- **向导模式**：向导模式是可视化界面配置同步任务， 一共涉及五步选择来源，选择目标，字段映射，通道控制，预览保存五个步骤。在每个不同的数据源之间，这几步的界面可能有不同的内容。向导模式可以转换成脚本模式。
- **脚本模式**：进入脚本界面你可以选择相应的模板，此模板包含了同步任务的主要参数，将相关的信息填写完整，但是脚本模式不能转化成向导模式。

2. 选择数据来源。 选择 MaxCompute 数据源及源头表 mytest，数据浏览默认是收起的，选择后单击**下一步**：

5

3. 选择目标。 选择PolarDB-X数据源及目标表contact_infos，选择后单击**下一步**：

6

- preSql：执行数据同步任务之前率先执行的 SQL 语句。目前向导模式只允许执行一条 SQL 语句，脚本模式可以支持多条 SQL 语句，例如清除旧数据。
- postSql：执行数据同步任务之后执行的 SQL 语句。目前向导模式只允许执行一条 SQL 语句，脚本模式可以支持多条 SQL 语句，例如加上某一个时间戳。

4. 选择字段的映射关系。左侧**源头表字段**和右侧**目标表字段**为一一对应的关系，如下图所示。

7

5. 在通道控制页面单击**下一步**，配置作业速率上限和脏数据检查规则。

8

- 作业速率上限：是指数据同步作业可能达到的最高速率，其最终实际速率受网络环境、数据库配置等的影响。
- 作业并发数：作业速率上限=作业并发数 * 单并发的传输速率。

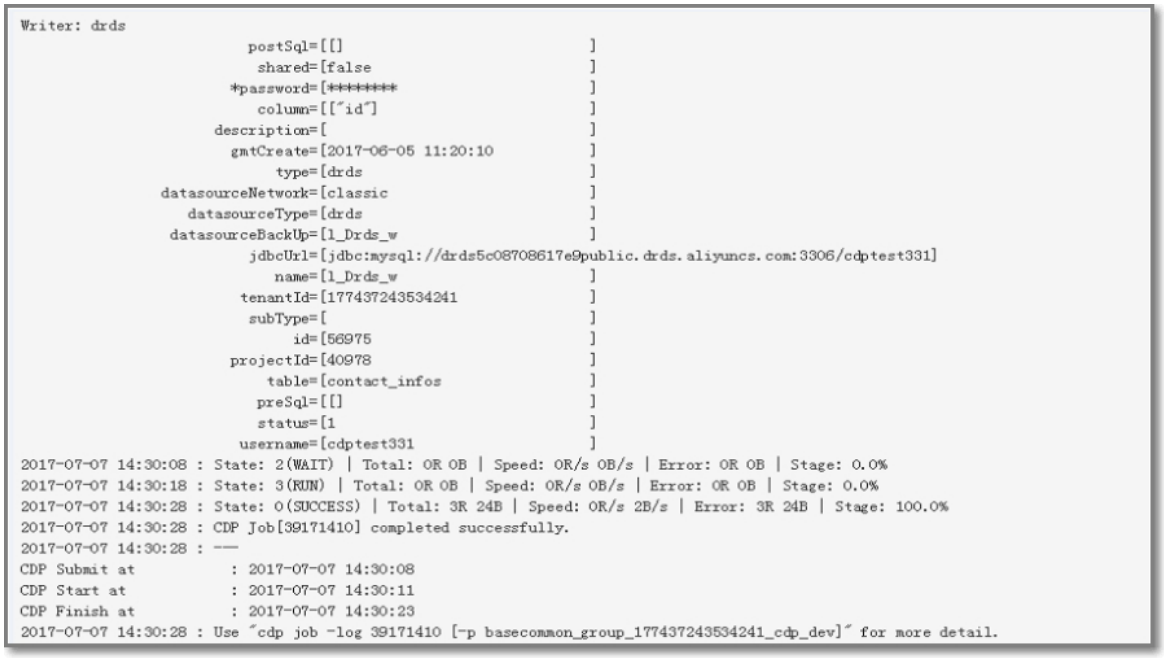
当作业速率上限已选定的情况下，可以根据以下原则选择并发数：

- 如果你的数据源是线上的业务库，建议您不要将并发数设置过大，以防对线上库造成影响；
- 如果您对数据同步速率特别在意，建议您选择最大作业速率上限和较大的作业并发数。

6. 完成以上配置后，上下滚动鼠标可查看任务配置，确定无误后单击**保存**。

9

7. 单击**运行任务**，直接运行同步任务结果。您可以将同步任务提交到调度系统中，调度系统会按照配置属性从第二天开始自动定时执行。



10

** 脚本模式配置同步任务**

```
{
 "type": "job",
 "version": "1.0",
 "configuration": {
  "reader": {
   "plugin": "odps",
   "parameter": {
    "datasource": "lzz_odps",//数据源的名称，建议都添加数据源后进行同步
    "table": "mytest",//数据来源的表名
    "partition": "",//分区信息
    "column": [
     "id"
    ]
   }
  },
  "writer": {
   "plugin": "drds",
   "parameter": {
    "datasource": "l_Drds_w",//数据源的名称，建议都添加数据源后进行同步
    "table": "contact_infos",//目的表名
    "preSql": [],//导入前准备语句
    "postSql": [],//导入后准备语句
    "column": [
     "id"
    ]
   }
  },
  "setting": {
   "speed": {
    "mbps": "1",//一个并发的速率上线是1MB/S
    "concurrent": "1"//并发的数目
   }
  }
 }
}
```

通过数据集成导出数据

下文以通过向导模式将PolarDB-X数据同步到 MaxCompute 为例。

1. 在数据集成页面，新建同步任务。

11

2. 选择数据来源。选择PolarDB-X数据源及源头表 bit_type_test。数据浏览默认是收起的，选择后单击**下一步**，如下图所示：

12

- **过滤条件**：筛选条件，DrdsReader 根据指定的 column、table、where 条件拼接 SQL，并根据这个 SQL 进行数据抽取 。例如在做测试时，可以将 where 条件指定实际业务场景，往往会选择当天的数据进行同步，可以将 where 条件指定为 `STRTODATE('${bdp.system.bizdate}', '%Y%m%d') <= taday AND taday < DATEADD(STRTODATE('${bdp.system.bizdate}', '%Y%m%d'), interval 1 day)` 。

- **切分键**：您可以将源数据表中某一列作为切分键，切分之后可进行并发数据同步。目前仅支持整型字段；建议使用主键或有索引的列作为切分键。

3. 选择 MaxCompute 数据源及目标表 mytest，选择后单击**下一步**。

13

4. 单击**下一步**，选择字段的映射关系。左侧**源头表字段**和右侧**目标表字段**为一一对应的关系：

14 您也可以单击“添加一行”增加映射关系：

- 可以输入常量，输入的值需要使用英文单引号包括，如'abc'、'123'等；
- 可以配合调度参数使用，如'${bdp.system.bizdate}'等；
- 可以输入关系数据库支持的函数，如 now()、count(1)等；
- 如果您输入的值无法解析，则类型显示为'-'。

5. 在通道控制页面单击**下一步**，配置作业速率上限和脏数据检查规则。

15

- 作业速率上限：是指数据同步作业可能达到的最高速率，其最终实际速率受网络环境、数据库配置等的影响。
- 作业并发数：作业速率上限=作业并发数 * 单并发的传输速率。

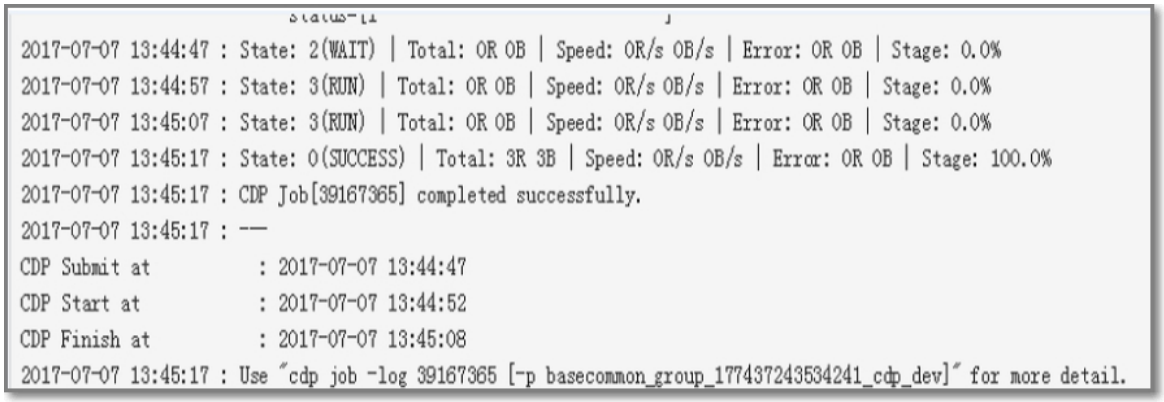
当作业速率上限已选定的情况下，可以按以下原则选择并发数：

- 如果你的数据源是线上的业务库，建议您不要将并发数设置过大，以防对线上库造成影响；
- 如果您对数据同步速率特别在意，建议您选择最大作业速率上限和较大的作业并发数。

6. 完成以上配置后，上下滚动鼠标可查看任务配置。确认无误后单击**保存**。

16

7. 单击**运行任务**直接运行同步任务结果。您也可以将同步任务提交到调度系统中，调度系统会按照配置属性从第二天开始自动定时执行。



17

脚本模式配置同步任务

```
{
 "type": "job",
 "version": "1.0",
 "configuration": {
  "reader": {
   "plugin": "drds",
   "parameter": {
    "datasource": "l_Drds_w",//数据源的名称，建议都添加数据源后进行同步
    "table": "bit_type_test",/数据来源的表名
    "where": "",
    "splitPk": "col2",//切分键
    "column": [
     "idbit"
    ]
   }
  },
  "writer": {
   "plugin": "odps",
   "parameter": {
    "datasource": "lzz_odps",//数据源的名称，建议都添加数据源后进行同步
    "table": "mytest",
    "truncate": true,
    "partition": "",//分区信息
    "column": [
     "id"
    ]
   }
  },
  "setting": {
   "speed": {
    "mbps": "1",//作业速率上限
    "concurrent": "1"//并发数
   },
   "errorLimit": {
    "record": "234"//错误记录数
   }
  }
 }
}
```

16.自定义控制指令

16.1. 概览

本文汇总了PolarDB-X支持的辅助SQL指令供您参考。

自定义控制指令

- 帮助语句：展示PolarDB-X所有辅助SQL指令及其说明。
- 查看规则和节点拓扑类语句
- SQL 调优类语句
- 统计信息查询类语句
- SHOW PROCESSLIST指令与KILL指令
- SEQUENCE 相关语句

16.2. 帮助语句

您可以通过SHOW HELP帮助语句查看PolarDB-X所有辅助类SQL语句及语句的详细描述。

示例

```
mysql> show help;
+---------------------------------------+------------------------------------------------------+-----------------------------------------
--+
| STATEMENT                              | DESCRIPTION                                          | EXAMPLE                                  |
+---------------------------------------+------------------------------------------------------+-----------------------------------------
--+
| show rule                              | Report all table rule                                |                                          |
| show rule from TABLE                   | Report table rule                                    | show rule from user                      |
| show full rule from TABLE              | Report table full rule                               | show full rule from user                 |
| show topology from TABLE               | Report table physical topology                       | show topology from user
|
| show partitions from TABLE             | Report table dbPartition or tbPartition columns      | show partitions fr
om user                  |
| show broadcasts                        | Report all broadcast tables                          |                                          |
| show datasources                       | Report all partition db threadPool info              |                                          |
| show node                              | Report master/slave read status                      |                                          |
| show slow                              | Report top 100 slow sql                              |                                          |
| show physical_slow                     | Report top 100 physical slow sql                     |                                          |
| clear slow                             | Clear slow data                                      |                                          |
| trace SQL                              | Start trace sql, use show trace to print profiling data | trace select count(*) from us
er; show trace |
| show trace                             | Report sql execute profiling info                    |                                          |
| explain SQL                            | Report sql plan info                                 | explain select count(*) from user        |
| explain detail SQL                     | Report sql detail plan info                          | explain detail select count(*) from use
r   |
| explain execute SQL                    | Report sql on physical db plan info                  | explain execute select count(*) f
rom user   |
| show sequences                         | Report all sequences status                          |                                          |
| create sequence NAME [start with COUNT] | Create sequence                                     | create sequence test start
with 0        |
| alter sequence NAME [start with COUNT]  | Alter sequence                                      | alter sequence test start with
100000     |
| drop sequence NAME                     | Drop sequence                                        | drop sequence test                       |
+---------------------------------------+------------------------------------------------------+-----------------------------------------
--+
20 rows in set (0.00 sec)
```

16.3. 查看规则和节点拓扑类语句

查看规则和节点拓扑类语句如下：

背景信息

- SHOW RULE [FROM tablename] 语句
- SHOW FULL RULE [FROM tablename] 语句
- SHOW TOPOLOGY FROM tablename 语句
- SHOW PARTITIONS FROM tablename 语句
- SHOW BROADCASTS 语句
- SHOW DATASOURCES 语句

- SHOW NODE 语句

1. SHOW RULE [FROM tablename]

使用说明：

- `show rule`：查看数据库下每一个逻辑表的拆分情况；
- `show rule from tablename`：查看数据库下指定逻辑表的拆分情况。

重要列详解：

- **BROADCAST**：是否为广播表（0：否，1：是）；
- **DB_PARTITION_KEY**：分库的拆分键，没有分库的话，值为空；
- **DB_PARTITION_POLICY**：分库的拆分策略，取值包括哈希或 YYYYMM、YYYYDD、YYYYWEEK 等日期策略；
- **DB_PARTITION_COUNT**：分库数；
- **TB_PARTITION_KEY**：分表的拆分键，没有分表的话，值为空；
- **TB_PARTITION_POLICY**：分表的拆分策略，取值包括哈希或 MM、DD、MMDD、WEEK 等日期策略；
- **TB_PARTITION_COUNT**：分表数。

```
mysql> show rule;
+------+-------------+----------+-----------------+--------------------+-------------------+-----------------+--------------------+-----------------+
| ID   | TABLE_NAME  | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COUNT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+-------------+----------+-----------------+--------------------+-------------------+-----------------+--------------------+-----------------+
|  0 | dept_manager |      0 |                 | NULL               |                 1 |                 | NULL               |                 1 |
|  1 | emp          |      0 | emp_no          | hash               |                 8 | id              | hash               |                 2 |
|  2 | example      |      0 | shard_key       | hash               |                 8 |                 | NULL               |                 1 |
+------+-------------+----------+-----------------+--------------------+-------------------+-----------------+--------------------+-----------------+
3 rows in set (0.01 sec)
```

2. SHOW FULL RULE [FROM tablename]

查看数据库下逻辑表的拆分规则，比 SHOW RULE 指令展示的信息更加详细。

重要列详解：

- **BROADCAST**：是否为广播表（0：否，1：是）；
- **JOIN_GROUP**：保留字段，暂时无意义。
- **ALLOW_FULL_TABLE_SCAN**：分库分表在没有指定分表键值的情况下是否允许查询数据，如果配置为 true，此时需要扫描每一个物理表来查找出符合条件的数据，简称为全表扫描；
- **DB_NAME_PATTERN**：DB_NAME_PATTERN 中 {} 之间的 0 为占位符，执行 SQL 时会被 DB_RULES_STR 计算出的值替代，并保持位数。比如，DB_NAME_PATTERN 的值为 SEQ_{0000}_RDS，DB_RULES_STR 的值为 [1,2,3,4]，则会产生4个 DB_NAME，分别为 SEQ_0001_RDS、SEQ_0002_RDS、SEQ_0003_RDS、SEQ_0004_RDS；
- **DB_RULES_STR**：具体的分库规则；
- **TB_NAME_PATTERN**：TB_NAME_PATTERN 中 {} 之间的 0 为占位符，执行 SQL 时会被 TB_RULES_STR 计算出的值替代，并保持位数。比如，TB_NAME_PATTERN 的值为 table_{00}，TB_RULES_STR 的值为 [1,2,3,4,5,6,7,8]，则会产生8张表，分别为 table_01、table_02、table_03、table_04、table_05、

table_06、table_07、table_08；

- TB_RULES_STR：分表规则；
- PARTITION_KEYS：分库和分表键集合，对于既分库又分表的情形，分库键在前，分表键在后；
- DEFAULT_DB_INDEX：单库单表存放的分库。

```
mysql> show full rule;
+------+--------------+-----------+------------+----------------------+--------------------------------------------------+-----------------------------------------------------+----------------+----------------------------------+----------------+------------------------------------------------+
| ID   | TABLE_NAME   | BROADCAST | JOIN_GROUP | ALLOW_FULL_TABLE_SCAN | DB_NAME_PATTERN                                  | DB_RULES_STR                                        | TB_NAME_PATTERN | TB_RULES_STR                     | PARTITION_KEYS | DEFAULT_DB_INDEX                               |
+------+--------------+-----------+------------+----------------------+--------------------------------------------------+-----------------------------------------------------+----------------+----------------------------------+----------------+------------------------------------------------+
|    0 | dept_manager |         0 | NULL       |                    0 | SEQ_TEST_*************RGKKSEQ_TEST_****_0000_RDS | NULL                                                | dept_manager   | NULL                             | NULL           | SEQ_TEST_*************RGKKSEQ_TEST_****_0000_RDS |
|    1 | emp          |         0 | NULL       |                    1 | SEQ_TEST_148*************SEQ_TEST_****_{0000}_RDS | ((#emp_no,1,8#).longValue().abs() % 8)              | emp_{0}        | ((#id,1,2#).longValue().abs() % 2) | emp_no     id | SEQ_TEST_*************RGKKSEQ_TEST_****_0000_RDS |
|    2 | example      |         0 | NULL       |                    1 | SEQ_TEST_*************RGKKSEQ_TEST_****_{0000}_RDS | ((#shard_key,1,8#).longValue().abs() % 8).intdiv(1) | example        | NULL                             | shard_key      | SEQ_TEST_*************RGKKSEQ_TEST_****_0000_RDS |
+------+--------------+-----------+------------+----------------------+--------------------------------------------------+-----------------------------------------------------+----------------+----------------------------------+----------------+------------------------------------------------+
3 rows in set (0.01 sec)
```

3. SHOW TOPOLOGY FROM tablename

查看指定逻辑表的拓扑分布，展示该逻辑表保存在哪些分库中，每个分库下包含哪些分表。

```
mysql> show topology from emp;
+------+-------------------------------------------------+-----------+
| ID   | GROUP_NAME                                      | TABLE_NAME |
+------+-------------------------------------------------+-----------+
|  0 | SEQ_TEST_*************RGKKSEQ_TEST_****_0000_RDS | emp_0    |
|  1 | SEQ_TEST_*************RGKKSEQ_TEST_****_0000_RDS | emp_1    |
|  2 | SEQ_TEST_*************RGKKSEQ_TEST_****_0001_RDS | emp_0    |
|  3 | SEQ_TEST_*************RGKKSEQ_TEST_****_0001_RDS | emp_1    |
|  4 | SEQ_TEST_*************RGKKSEQ_TEST_****_0002_RDS | emp_0    |
|  5 | SEQ_TEST_*************RGKKSEQ_TEST_****_0002_RDS | emp_1    |
|  6 | SEQ_TEST_*************RGKKSEQ_TEST_****_0003_RDS | emp_0    |
|  7 | SEQ_TEST_*************RGKKSEQ_TEST_****_0003_RDS | emp_1    |
|  8 | SEQ_TEST_*************RGKKSEQ_TEST_****_0004_RDS | emp_0    |
|  9 | SEQ_TEST_*************RGKKSEQ_TEST_****_0004_RDS | emp_1    |
| 10 | SEQ_TEST_*************RGKKSEQ_TEST_****_0005_RDS | emp_0    |
| 11 | SEQ_TEST_*************RGKKSEQ_TEST_****_0005_RDS | emp_1    |
| 12 | SEQ_TEST_*************RGKKSEQ_TEST_****_0006_RDS | emp_0    |
| 13 | SEQ_TEST_*************RGKKSEQ_TEST_****_0006_RDS | emp_1    |
| 14 | SEQ_TEST_*************RGKKSEQ_TEST_****_0007_RDS | emp_0    |
| 15 | SEQ_TEST_*************RGKKSEQ_TEST_****_0007_RDS | emp_1    |
+------+-------------------------------------------------+-----------+
16 rows in set (0.01 sec)
```

4. SHOW PARTITIONS FROM tablename

查看分库分表键集合，分库键和分表键之间用逗号分割。如果最终结果有两个值，说明是既分库又分表的情形，第一个是分库键，第二个是分表键。如果结果只有一个值，说明是分库不分表的情形，该值是分库键。

```
mysql> show partitions from emp;
+-----------+
| KEYS      |
+-----------+
| emp_no,id |
+-----------+
1 row in set (0.00 sec)
```

5. SHOW BROADCASTS

查看广播表列表。

```
mysql> show broadcasts;
+------+------------+
| ID   | TABLE_NAME |
+------+------------+
|  0 | ***2      |
|  1 | ***_tbl   |
+------+------------+
2 rows in set (0.01 sec)
```

6. SHOW DATASOURCES

查看底层存储信息，包含数据库名、数据库分组名、连接信息、用户名、底层存储类型、读写权重、连接池信息等。

重要列详解：

- **SCHEMA**：数据库名；
- **GROUP**：数据库分组名。分组的目标是管理多组数据完全相同的数据库，比如通过 RDS（MySQL）进行数据复制后的主备数据库。主要用来解决读写分离、主备切换的问题；
- **URL**：底层 RDS（MySQL）的连接信息；
- **TYPE**：底层存储类型，目前只支持 RDS（MySQL)；
- **READ_WEIGHT**：读权重。在主实例的读压力比较大的时候，可以通过 DRDS 读写分离功能将读流量进行分流，减轻 RDS 主实例的压力。DRDS 会自动识别读写流量，引导写流量进入 RDS 主实例，读流量则按配置的权重流向所有 RDS 实例；
- **WRITE_WEIGHT**：写权重。解释见上。

```
mysql> show datasources;
+------+--------------------------+------------------------------------------+----------------------------------------------+------
--------------------------------------------------------------------+-----------+-------+------+------+------+-------------+---------
-+-------------+-------------+-----------------------------------------+------------+-------------+
| ID   | SCHEMA              | NAME                              | GROUP                            | URL
| USER    | TYPE | INIT | MIN  | MAX  | IDLE_TIMEOUT | MAX_WAIT | ACTIVE_COUNT | POOLING_COUNT | ATOM
| READ_WEIGHT | WRITE_WEIGHT |
+------+--------------------------+------------------------------------------+----------------------------------------------+------
--------------------------------------------------------------------+-----------+-------+------+------+------+-------------+---------
-+-------------+-------------+-----------------------------------------+------------+-------------+
|   0 | seq_test_148************* | rds****************_seq_test_****_0000_****_1 | SEQ_TEST_***********
**RGKKSEQ_TEST_****_0000_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_tes
t_****_0000 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************_seq_test
_****_0000_**** | 10      | 10       |
|   1 | seq_test_148************* | rds****************_seq_test_****_****_****_2| SEQ_TEST_************
*RGKKSEQ_TEST_****_0001_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_test
_****_0001 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************_seq_test_
****_0001_**** | 10      | 10       |
|   2 | seq_test_148************* | rds****************_seq_test_****_0002_****_3 | SEQ_TEST_***********
**RGKKSEQ_TEST_****_0002_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_tes
t_****_0002 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************_seq_test
_****_0002_**** | 10      | 10       |
|   3 | seq_test_148************* | rds****************_seq_test_****_0003_****_4 | SEQ_TEST_***********
**RGKKSEQ_TEST_****_0003_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_tes
t_****_0003 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************_seq_test
_****_0003_**** | 10      | 10       |
|   4 | seq_test_148************* | rds****************_seq_test_****_0004_****_5 | SEQ_TEST_***********
**RGKKSEQ_TEST_****_0004_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_tes
t_****_0004 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************_seq_test
_****_0004_**** | 10      | 10       |
|   5 | seq_test_148************* | rds****************_seq_test_****_0005_****_6 | SEQ_TEST_***********
**RGKKSEQ_TEST_****_0005_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_tes
t_****_0005 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************_seq_test
_****_0005_**** | 10      | 10       |
|   6 | seq_test_148************* | rds****************_seq_test_****_0006_****_7 | SEQ_TEST_***********
**RGKKSEQ_TEST_****_0006_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_tes
t_****_0006 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************
_seq_test_****_0006_**** | 10      | 10       |
|   7 | seq_test_148************* | rds****************_seq_test_****_0007_****_8 | SEQ_TEST_***********
**RGKKSEQ_TEST_****_0007_RDS | jdbc:mysql://rds****************.mysql.rds.aliyuncs.com:3306/seq_tes
t_****_0007 | ********0 | mysql | 0   | 24  | 72  | 15       | 5000   | 0       | 1        | rds****************_seq_test
_****_0007_**** | 10      | 10       |
+------+--------------------------+------------------------------------------+----------------------------------------------+------
--------------------------------------------------------------------+-----------+-------+------+------+------+-------------+---------
-+-------------+-------------+-----------------------------------------+------------+-------------+
8 rows in set (0.01 sec)
```

7. SHOW NODE

查看物理库的读写次数（历史累计数据）、读写权重（历史累计数据）。

重要列详解:

- **MASTER_READ_COUNT**：RDS 主实例处理的只读查询次数（历史累计数据）；
- **SLAVE_READ_COUNT**：RDS 备实例处理的只读查询次数（历史累计数据）；
- **MASTER_READ_PERCENT**：RDS 主实例处理的只读查询占比（注意该列显示的是累计的实际数据占比，并不是用户配置的百分比）；
- **SLAVE_READ_PERCENT**：RDS 备实例处理的只读查询占比（注意该列显示的是累计的实际数据占比，并不是用户配置的百分比）。

注意：

- 事务中的只读查询会被发送到 RDS 主实例；
- 由于 `MASTER_READ_PERCENT`，`SLAVE_READ_PERCENT` 这两列代表的是历史累计数据，更改读写权重的配比后，这几个数值并不能立即反应最新的读写权重配比，需累计一段比较长的时间才行。

```
mysql> show node;
+------+-----------------------------------------------+-----------------+-----------------+--------------------+-------------------+
| ID   | NAME                                          | MASTER_READ_COUNT | SLAVE_READ_COUNT | MASTER_READ_PERCENT | SLAVE_READ_PERCENT |
+------+-----------------------------------------------+-----------------+-----------------+--------------------+-------------------+
| 0    | SEQ_TEST_*************RGKKSEQ_TEST_****_0000_RDS |              12 |               0 | 100%               | 0%                |
| 1    | SEQ_TEST_*************RGKKSEQ_TEST_****_0001_RDS |               0 |               0 | 0%                 | 0%                |
| 2    | SEQ_TEST_*************RGKKSEQ_TEST_****_0002_RDS |               0 |               0 | 0%                 | 0%                |
| 3    | SEQ_TEST_*************RGKKSEQ_TEST_****_0003_RDS |               0 |               0 | 0%                 | 0%                |
| 4    | SEQ_TEST_*************RGKKSEQ_TEST_****_0004_RDS |               0 |               0 | 0%                 | 0%                |
| 5    | SEQ_TEST_*************RGKKSEQ_TEST_****_0005_RDS |               0 |               0 | 0%                 | 0%                |
| 6    | SEQ_TEST_*************RGKKSEQ_TEST_****_0006_RDS |               0 |               0 | 0%                 | 0%                |
| 7    | SEQ_TEST_*************RGKKSEQ_TEST_****_0007_RDS |               0 |               0 | 0%                 | 0%                |
+------+-----------------------------------------------+-----------------+-----------------+--------------------+-------------------+
8 rows in set (0.01 sec)
```

16.4. SQL 调优类语句

SQL 调优类语句如下：

背景信息

- SHOW [FULL] SLOW [WHERE expr] [limit expr] 语句
- SHOW [FULL] PHYSICAL_SLOW [WHERE expr] [limit expr] 语句
- CLEAR SLOW 语句
- EXPLAIN SQL 语句
- EXPLAIN DETAIL SQL 语句
- EXPLAIN EXECUTE SQL 语句
- TRACE SQL 和 SHOW TRACE 语句
- CHECK TABLE tablename 语句
- SHOW TABLE STATUS 语句

1. SHOW [FULL] SLOW [WHERE expr] [limit expr]

执行时间超过 1 秒的 SQL 语句是慢 SQL，逻辑慢 SQL 是指应用发送到 DRDS 的慢 SQL。如何排查慢 SQL 可以参考文档 排查 DRDS 慢 SQL。

- SHOW SLOW ：查看自 DRDS 启动或者上次执行 CLEAR SLOW 以来最慢的 100 条逻辑慢 SQL（注意，这里记录的是最慢的 100 个，缓存在 DRDS 系统中，当实例重启或者执行 CLEAR SLOW 时会丢失）；
- SHOW FULL SLOW ：查看实例启动以来记录的所有逻辑慢 SQL（持久化到 DRDS 的内置数据库中）。该记录数有一个上限（具体数值跟购买的实例规格相关），DRDS 会滚动删除比较老的慢 SQL 语句。实例的规格如果是 4C4G 的话，最多记录 10000 条慢 SQL 语句（包括逻辑慢 SQL 和 物理慢 SQL）；实例的规格如果是 8C8G 的话，最多记录 20000 条慢 SQL 语句（包括逻辑慢 SQL 和 物理慢 SQL），其它规格依此类推。

重要列详解：

- HOST：来源 IP；
- START_TIME：执行开始时间；
- EXECUTE_TIME：执行时间；
- AFFECT_ROW：对于 DML 语句是影响行数；对于查询语句是返回的记录数。

```
mysql> show slow where execute_time > 1000 limit 1;
+-----------+---------------------+--------------+------------+-----------+
| HOST      | START_TIME          | EXECUTE_TIME | AFFECT_ROW | SQL       |
+-----------+---------------------+--------------+------------+-----------+
| 127.0.0.1 | 2016-03-16 13:02:57 |         2785 |          7 | show rule |
+-----------+---------------------+--------------+------------+-----------+
1 row in set (0.02 sec)
```

2. SHOW [FULL] PHYSICAL_SLOW [WHERE expr] [limit expr]

执行时间超过1秒的 SQL 语句是慢 SQL，物理慢 SQL 是指 DRDS 发送到 RDS 的慢 SQL。如何排查慢 SQL 可以参考文档 排查 DRDS 慢 SQL。

- SHOW PHYSICAL_SLOW ：查看自 DRDS 启动或者上次执行 CLEAR SLOW 以来最慢的 100 条物理慢 SQL（注意，这里记录的是最慢的 100 个，缓存在 DRDS 系统中，当实例重启或者执行 CLEAR SLOW 时会丢失）；
- SHOW FULL PHYSICAL_SLOW ：查看实例启动以来记录的所有物理慢 SQL（持久化到 DRDS 的内置数据库中）。该记录数有一个上限（具体数值跟购买的实例规格相关），DRDS 会滚动删除比较老的慢 SQL 语句。实例的规格如果是 4C4G 的话，最多记录 10000 条慢 SQL 语句（包括逻辑慢 SQL 和 物理慢 SQL）；实例的规格如果是 8C8G 的话，最多记录 20000 条慢 SQL 语句（包括逻辑慢 SQL 和 物理慢 SQL），其它规格依此类推。

重要列详解：

- GROUP_NAME：数据库分组；
- START_TIME：执行开始时间；
- EXECUTE_TIME：执行时间；
- AFFECT_ROW：对于 DML 语句是影响行数；对于查询语句是返回的记录数。

```
mysql> show physical_slow;
+----------------+----------------------------------+--------------------+--------------+------------------+-------------------------+------------------+------------+-----------------+
| GROUP_NAME     | DBKEY_NAME                       | START_TIME         | EXECUTE_TIME | SQL_EXECUTE_TIME | GETLOCK_CONNECTION_TIME | CREATE_CONNECTION_TIME | AFFECT_ROW | SQL          |
+----------------+----------------------------------+--------------------+--------------+------------------+-------------------------+------------------+------------+-----------------+
| TDDL5_00_GROUP | db218249098_sqa_zmf_tddl5_00_3309 | 2016-03-16 13:05:38 |        1057 |             1011 |
0 |                0 |          1 | select sleep(1) |
+----------------+----------------------------------+--------------------+--------------+------------------+-------------------------+------------------+------------+-----------------+
1 row in set (0.01 sec)
```

3. CLEAR SLOW

清空自 DRDS 启动或者上次执行 `CLEAR SLOW` 以来最慢的 100 条逻辑慢 SQL 和 最慢的 100 条物理慢 SQL。注意： `SHOW SLOW` 和 `SHOW PHYSICAL_SLOW` 展示的是最慢的 100 个 SQL，如果长时间未执行 `CLEAR SLOW` ，可能都是非常老的 SQL 了，一般执行过 SQL 优化之后，建议都执行下 `CLEAR SLOW` ，等待系统运行一段时间，再查看下慢 SQL 的优化效果。

```
mysql> clear slow;
Query OK, 0 rows affected (0.00 sec)
```

4. EXPLAIN SQL

查看指定 SQL 在 DRDS 层面的执行计划，注意这条 SQL 不会实际执行。关于该指令的更多案例请参考文档 SQL 优化基本概念、SQL 优化方法 。

示例：

查看 `select * from doctest` 这条 SQL 的执行计划（doctest 这张表是按照 id 列进行分库的）。从执行计划可以看出该 SQL 会下推到每个分库，然后将执行结果聚合。

```
mysql> explain select * from doctest;
+-----------------------------------------------+-------------------------------------+--------+
| GROUP_NAME                                    | SQL                                 | PARAMS |
+-----------------------------------------------+-------------------------------------+--------+
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0000_RDS | select `doctest`.`id` from `doctest` | {}     |
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0001_RDS | select `doctest`.`id` from `doctest` | {}     |
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0002_RDS | select `doctest`.`id` from `doctest` | {}     |
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0003_RDS | select `doctest`.`id` from `doctest` | {}     |
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0004_RDS | select `doctest`.`id` from `doctest` | {}     |
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0005_RDS | select `doctest`.`id` from `doctest` | {}     |
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0006_RDS | select `doctest`.`id` from `doctest` | {}     |
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0007_RDS | select `doctest`.`id` from `doctest` | {}     |
+-----------------------------------------------+-------------------------------------+--------+
8 rows in set (0.00 sec)
```

查看 `select * from doctest where id = 1` 这条 SQL 的执行计划（doctest 这张表是按照 id 列进行分库的）。从执行计划可以看出该 SQL 会根据拆分键（id）计算出具体的分库，将 SQL 直接下推到该分库，然后执行结果聚合。

```
mysql> explain select * from doctest where id = 1;
+----------------------------------------------+-------------------------------------------------------------+--------+
| GROUP_NAME                                   | SQL                                                         | PARAMS |
+----------------------------------------------+-------------------------------------------------------------+--------+
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0001_RDS | select `doctest`.`id` from `doctest` where (`doctest`.`id` = 1) | {}     |
+----------------------------------------------+-------------------------------------------------------------+--------+
1 row in set (0.01 sec)
```

5. EXPLAIN DETAIL SQL

查看指定 SQL 在 DRDS 层面的执行计划。注意这条 SQL 不会实际执行。关于该指令的更多案例请参考文档 SQL 优化基本概念、SQL 优化方法 。

```
mysql> explain detail select * from doctest where id = 1;
+----------------------------------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+--------+
| GROUP_NAME                                   | SQL
| PARAMS |
+----------------------------------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+--------+
| DOCTEST_1488704345426RCUPDOCTEST_CAET_0001_RDS | Query from doctest as doctest
  keyFilter:doctest.id = 1
  queryConcurrency:SEQUENTIAL
  columns:[doctest.id]
  tableName:doctest
  executeOn:DOCTEST_1488704345426RCUPDOCTEST_CAET_0001_RDS
 | NULL   |
+----------------------------------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+--------+
1 row in set (0.02 sec)
```

6. EXPLAIN EXECUTE SQL

查看底层存储的执行计划，等同于 MYSQL 的 EXPLAIN 语句。关于该指令的更多案例请参考文档 SQL 优化基本概念、SQL 优化方法 。

```
mysql> explain execute select * from tddl_mgr_log limit 1;
+----+-------------+--------------+------+---------------+------+---------+------+------+-------+
| id | select_type | table        | type | possible_keys | key  | key_len | ref  | rows | Extra |
+----+-------------+--------------+------+---------------+------+---------+------+------+-------+
|  1 | SIMPLE      | tddl_mgr_log | ALL  | NULL          | NULL | NULL    | NULL |    1 | NULL  |
+----+-------------+--------------+------+---------------+------+---------+------+------+-------+
1 row in set (0.07 sec)
```

7. TRACE SQL 和 SHOW TRACE

查看具体 SQL 的执行情况。TRACE [SQL] 和 SHOW TRACE 要结合使用。注意 TRACE SQL 和 EXPLAIN SQL 的区别在于 TRACE SQL 会实际执行该语句。 关于该指令的更多案例请参考文档 SQL 优化基本概念、SQL 优化方法 。

例如查看 `select 1` 这条语句的执行情况。

```
mysql> trace select 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.03 sec)
mysql> show trace;
+------+----------+----------------+---------------------------------+--------------+-------------------------+------+-----------+-------
-+
| ID  | TYPE    | GROUP_NAME    | DBKEY_NAME                | TIME_COST(MS) | CONNECTION_TIME_COST(MS) | R
OWS | STATEMENT | PARAMS |
+------+----------+----------------+---------------------------------+--------------+-------------------------+------+-----------+-------
-+
|   0 | Optimize | DRDS       | DRDS                  | 3        | 0.00             |   0 | select 1 | NULL  |
|   1 | Query   | TDDL5_00_GROUP | db218249098_sqa_zmf_tddl5_00_3309 | 7        | 0.15             |   1 | select
1 | NULL  |
+------+----------+----------------+---------------------------------+--------------+-------------------------+------+-----------+-------
-+
2 rows in set (0.01 sec)
```

8. CHECK TABLE tablename

对数据表进行检查。主要用于 DDL 建表失败的情形，应用案例请参考文档 DDL 常见问题处理 。

- 对于拆分表，检查底层物理分表是否有缺失的情况，底层的物理分表的列和索引是否是一致；
- 对于单库单表，检查表是否存在。

```
mysql>  check table tddl_mgr_log;
+-----------------------+-------+----------+----------+
| TABLE           | OP   | MSG_TYPE | MSG_TEXT |
+-----------------------+-------+----------+----------+
| TDDL5_APP.tddl_mgr_log | check | status  | OK     |
+-----------------------+-------+----------+----------+
1 row in set (0.56 sec)
mysql> check table tddl_mg;
+-------------------+-------+----------+-----------------------------------------+
| TABLE        | OP   | MSG_TYPE | MSG_TEXT                  |
+-------------------+-------+----------+-----------------------------------------+
| TDDL5_APP.tddl_mg | check | Error   | Table 'tddl5_00.tddl_mg' doesn't exist |
+-------------------+-------+----------+-----------------------------------------+
1 row in set (0.02 sec)
```

9. SHOW TABLE STATUS [LIKE 'pattern' | WHERE expr]

获取表的信息，该指令聚合了底层各个物理分表的数据。

重要列详解：

- NAME：表名称；
- ENGINE：表的存储引擎；
- VERSION：表的存储引擎的版本；
- ROW_FORMAT：行格式，主要是 Dynamic、Fixed、Compressed 这三种格式。动态（Dynamic）行的行长度可变，例如 VARCHAR 或 BLOB 类型字段；固定（Fixed）行是指行长度不变，例如 CHAR 和 INTEGER 类型字段；
- ROWS：表中的行数；
- AVG_ROW_LENGTH：平均每行包括的字节数；
- DATA_LENGTH：整个表的数据量（单位：字节）；
- MAX_DATA_LENGTH：表可以容纳的最大数据量；
- INDEX_LENGTH：索引占用磁盘的空间大小；
- CREATE_TIME：表的创建时间；
- UPDATE_TIME：表的最近更新时间；
- COLLATION：表的默认字符集和字符排序规则；
- CREATE_OPTIONS：指表创建时的其他所有选项。

```
mysql> show table status like 'multi_db_multi_tbl';
+--------------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+-----------+----------------+---------------------+-------------+------------+-----------------+----------+----------------+---------+
| NAME               | ENGINE | VERSION | ROW_FORMAT | ROWS | AVG_ROW_LENGTH | DATA_LENGTH | MAX_DATA_LENGTH | INDEX_LENGTH | DATA_FREE | AUTO_INCREMENT | CREATE_TIME         | UPDATE_TIME | CHECK_TIME | COLLATION      | CHECKSUM | CREATE_OPTIONS | COMMENT |
+--------------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+-----------+----------------+---------------------+-------------+------------+-----------------+----------+----------------+---------+
| multi_db_multi_tbl | InnoDB |      10 | Compact    |    2 |          16384 |       16384 |               0 |        16384 |         0 |         100000 | 2017-03-27 17:43:57.0 | NULL        | NULL       | utf8_general_ci | NULL     |                |         |
+--------------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+-----------+----------------+---------------------+-------------+------------+-----------------+----------+----------------+---------+
1 row in set (0.03 sec)
```

和 DRDS 的 SCAN HINT 结合，还可以查看每个物理分表的数据量。具体请参考 HINT 语法。

```
mysql> /!TDDL:SCAN='multi_db_multi_tbl'*/show table status like 'multi_db_multi_tbl';
+---------------------+--------+---------+------------+------+----------------+-------------+-----------------+-------------+-----------+----------------+---------------------+-------------+------------+-----------------+----------+----------------+---------+--------------+
| Name                | Engine | Version | Row_format | Rows | Avg_row_length | Data_length | Max_data_length | Index_length | Data_free | Auto_increment | Create_time         | Update_time | Check_time | Collation       | Checksum | Create_options | Comment | Block_format |
+---------------------+--------+---------+------------+------+----------------+-------------+-----------------+-------------+-----------+----------------+---------------------+-------------+------------+-----------------+----------+----------------+---------+--------------+
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    1 |          16384 |       16384 |               0 |       16384 |         0 |              2 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_1 | InnoDB |  10 | Compact    |    0 |              0 |       16384 |               0 |       16384 |         0 |              1 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
| multi_db_multi_tbl_0 | InnoDB |  10 | Compact    |    1 |          16384 |       16384 |               0 |       16384 |         0 |              3 | 2017-03-27 17:43:57 | NULL        | NULL       | utf8_general_ci |     NULL |                |         | Original     |
+---------------------+--------+---------+------------+------+----------------+-------------+-----------------+-------------+-----------+----------------+---------------------+-------------+------------+-----------------+----------+----------------+---------+--------------+
16 rows in set (0.04 sec)
```

16.5. 统计信息查询类语句

DRDS 提供以下语句用于查询实时统计信息。

背景信息

- SHOW [FULL] STATS

- SHOW DB STATUS
- SHOW FULL DB STATUS

SHOW [FULL] STATS

查看整体的统计信息，这些信息都是瞬时值。 注意不同版本的 DRDS `SHOW FULL STATS` 的结果是有区别的。

重要列说明：

- **QPS**：应用到 DRDS 的 QPS，通常称为逻辑 QPS；
- **RDS_QPS**：DRDS 到 RDS 的 QPS，通常称为物理 QPS；
- **ERROR_PER_SECOND**：每秒的错误数，包含 SQL 语法错误，主键冲突，系统错误，连通性错误等各类错误总和；
- **VIOLATION_PER_SECOND**：每秒的主键或者唯一键冲突；
- **MERGE_QUERY_PER_SECCOND**：通过分库分表，从多表中进行的查询；
- **ACTIVE_CONNECTIONS**：正在使用的连接；
- **CONNECTION_CREATE_PER_SECCOND**：每秒创建的连接数；
- **RT(MS)**：应用到 DRDS 的响应时间，通常称为逻辑 RT（响应时间）；
- **RDS_RT(MS)**：DRDS 到 RDS/MySQL 的响应时间，通常称为物理 RT；
- **NET_IN(KB/S)**：DRDS 收到的网络流量；
- **NET_OUT(KB/S)**：DRDS 输出的网络流量；
- **THREAD_RUNNING**：正在运行的线程数；
- **HINT_USED_PER_SECOND**：每秒带 HINT 的查询的数量；
- **HINT_USED_COUNT**：启动到现在带 HINT 的查询总量；
- **AGGREGATE_QUERY_PER_SECCOND**：每秒聚合查询的频次；
- **AGGREGATE_QUERY_COUNT**：聚合查询总数（历史累计数据）；
- **TEMP_TABLE_CREATE_PER_SECCOND**：每秒创建的临时表的数量；
- **TEMP_TABLE_CREATE_COUNT**：启动到现在创建的临时表总数量；
- **MULTI_DB_JOIN_PER_SECCOND**：每秒跨库 JOIN 的数量；
- **MULTI_DB_JOIN_COUNT**：启动到现在跨库 JOIN 的总量。

示例：

```
mysql> show stats;
+------+---------+----------+------------------+-----------------+----------------------+--------------------+--------+------------+----
----------+---------------+----------------+
| QPS  | RDS_QPS | SLOW_QPS | PHYSICAL_SLOW_QPS | ERROR_PER_SECOND | MERGE_QUERY_PER_SECOND
| ACTIVE_CONNECTIONS | RT(MS) | RDS_RT(MS) | NET_IN(KB/S) | NET_OUT(KB/S) | THREAD_RUNNING |
+------+---------+----------+------------------+-----------------+----------------------+--------------------+--------+------------+----
----------+---------------+----------------+
| 1.77 |  1.68 |  0.03 |        0.03 |       0.02 |           0.00 |           7 | 157.13 |   51.14 |    134.49 |      1.48 |
1 |
+------+---------+----------+------------------+-----------------+----------------------+--------------------+--------+------------+----
----------+---------------+----------------+
1 row in set (0.01 sec)
mysql> show full stats;
+------+---------+----------+------------------+-----------------+----------------------+----------------------+--------------------+----
--------------------------+--------+------------+-------------+---------------+---------------+--------------------+----------------+---
--------------------------+----------------------+-----------------------------+----------------------+-------------------------+---------
-----------+-------+---------+-------------+-----------+
| QPS  | RDS_QPS | SLOW_QPS | PHYSICAL_SLOW_QPS | ERROR_PER_SECOND | VIOLATION_PER_SECOND | ME
RGE_QUERY_PER_SECOND | ACTIVE_CONNECTIONS | CONNECTION_CREATE_PER_SECOND | RT(MS) | RDS_RT(
MS) | NET_IN(KB/S) | NET_OUT(KB/S) | THREAD_RUNNING | HINT_USED_PER_SECOND | HINT_USED_COUNT | A
GGREGATE_QUERY_PER_SECOND | AGGREGATE_QUERY_COUNT | TEMP_TABLE_CREATE_PER_SECOND | TEM
P_TABLE_CREATE_COUNT | MULTI_DB_JOIN_PER_SECOND | MULTI_DB_JOIN_COUNT | CPU  | FREEMEM | FUL
LGCCOUNT | FULLGCTIME |
+------+---------+----------+------------------+-----------------+----------------------+----------------------+--------------------+----
--------------------------+--------+------------+-------------+---------------+---------------+--------------------+----------------+---
--------------------------+----------------------+-----------------------------+----------------------+-------------------------+---------
-----------+-------+---------+-------------+-----------+
| 1.63 |  1.68 |  0.03 |        0.03 |       0.02 |           0.00 |           0.00 |           6 |              0.01 | 157.13 |    5
1.14 |    134.33 |      1.21 |         1 |          0.00 |        54 |             0.00 |          663 |              0.00 |
512 |            0.00 |         516 | 0.09% |  6.96% |     76446 |  21326906 |
+------+---------+----------+------------------+-----------------+----------------------+----------------------+--------------------+----
--------------------------+--------+------------+-------------+---------------+---------------+--------------------+----------------+---
--------------------------+----------------------+-----------------------------+----------------------+-------------------------+---------
-----------+-------+---------+-------------+-----------+
1 row in set (0.01 sec)
```

SHOW DB STATUS

用于查看物理库容量/性能信息，所有返回值为实时信息。 容量信息通过 MySQL 系统表获得，与真实容量情况可能有差异。

重要列说明：

- **NAME：** 代表一个 DRDS DB，此处显示的是 DRDS 内部标记，与 DRDS DB 名称不同；
- **CONNECTION_STRING：** 分库的连接信息；
- **PHYSICAL_DB：**分库名称， `TOTAL` 行代表一个 DRDS DB 中所有分库容量的总和；
- **SIZE_IN_MB：** 分库中数据占用的空间，单位为 MB；
- **RATIO：** 单个分库数据量在当前 DRDS DB 总数据量中的占比；
- **THREAD_RUNNING：** 物理数据库实例当前正在执行的线程情况，含义与 MySQL `SHOW GLOBAL STATUS` 指令返回值的含义相同，详情请参考 MySQL 文档。

示例：

```
mysql> show db status;
+------+--------------------------+-------------------+-------------------+------------+--------+----------------+
| ID   | NAME                     | CONNECTION_STRING | PHYSICAL_DB       | SIZE_IN_MB | RATIO  | THREAD_RUNNING |
+------+--------------------------+-------------------+-------------------+------------+--------+----------------+
|    1 | drds_db_1516187088365daui | 100.100.64.1:59077 | TOTAL             |  13.109375 | 100%   |              3 |
|    2 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 |   1.578125 | 12.04% |                |
|    3 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 |     1.4375 | 10.97% |                |
|    4 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0002 |     1.4375 | 10.97% |                |
|    5 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0003 |     1.4375 | 10.97% |                |
|    6 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0004 |   1.734375 | 13.23% |                |
|    7 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0005 |   1.734375 | 13.23% |                |
|    8 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0006 |   2.015625 | 15.38% |                |
|    9 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0007 |   1.734375 | 13.23% |                |
+------+--------------------------+-------------------+-------------------+------------+--------+----------------+
```

SHOW FULL DB STATUS [LIKE {tablename}]

用于查看物理库表容量和性能信息，所有返回值为实时信息。 容量信息通过 MySQL 系统表获得，与真实容量情况可能有差异。

重要列说明：

- **NAME：** 代表一个 DRDS DB。此处显示的是 DRDS 内部标记，与 DRDS DB 名称不同；
- **CONNECTION_STRING：** 分库的连接信息；
- **PHYSICAL_DB：**分库名称， `TOTAL` 行代表经过 LIKE 关键字筛选后得到的分库容量的总和。如果没有 LIKE，则为全部分库容量的总和；
- **PHYSICAL_TABLE：**分表名称， `TOTAL` 行代表经过 LIKE 关键字筛选后得到的分表容量的总和。如果没有LIKE，则为全部分表容量的总和；
- **SIZE_IN_MB：** 分表中数据占用的空间，单位为 MB；
- **RATIO:** 单个分表数据量在当前筛选出的分表总数据量中的占比；
- **THREAD_RUNNING：** 物理数据库实例当前正在执行的线程情况，含义与 MySQL `SHOW GLOBAL STATUS` 指令返回值的含义相同。详情请参考 MySQL 文档。

示例：

```
mysql> show full db status like hash_tb;
+------+--------------------------+-------------------+-------------------+----------------+------------+--------+----------------+
| ID   | NAME                     | CONNECTION_STRING | PHYSICAL_DB       | PHYSICAL_TABLE | SIZE_IN_MB | RATIO  | THREAD_RUNNING |
+------+--------------------------+-------------------+-------------------+----------------+------------+--------+----------------+
|    1 | drds_db_1516187088365daui | 100.100.64.1:59077 | TOTAL             |                |     19.875 | 100%   |              3 |
|    2 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 | TOTAL          |    3.03125 | 15.25% |                |
|    3 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 | hash_tb_00     |   1.515625 | 7.63%  |                |
|    4 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 | hash_tb_01     |   1.515625 | 7.63%  |                |
|    5 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 | TOTAL          |        2.0 | 10.06% |                |
|    6 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 | hash_tb_02     |   1.515625 | 7.63%  |                |
|    7 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 | hash_tb_03     |   0.484375 | 2.44%  |                |
```

```
|  8|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0002|TOTAL      |  3.03125|15.25%|
|
|  9|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0002|hash_tb_04   |  1.515625|7.63
% |         |
| 10|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0002|hash_tb_05   |  1.515625|7.6
3% |         |
| 11|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0003|TOTAL      |  1.953125|9.83%
|         |
| 12|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0003|hash_tb_06   |  1.515625|7.6
3% |         |
| 13|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0003|hash_tb_07   |   0.4375|2.2%
|         |
| 14|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0004|TOTAL      |  3.03125|15.25%
|         |
| 15|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0004|hash_tb_08   |  1.515625|7.6
3% |         |
| 16|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0004|hash_tb_09   |  1.515625|7.6
3% |         |
| 17|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0005|TOTAL      |  1.921875|9.67%
|         |
| 18|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0005|hash_tb_11   |  1.515625|7.6
3% |         |
| 19|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0005|hash_tb_10   |  0.40625|2.04
% |         |
| 20|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0006|TOTAL      |  3.03125|15.25%
|         |
| 21|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0006|hash_tb_12   |  1.515625|7.6
3% |         |
| 22|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0006|hash_tb_13   |  1.515625|7.6
3% |         |
| 23|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0007|TOTAL      |   1.875|9.43% |
|
| 24|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0007|hash_tb_14   |  1.515625|7.6
3% |         |
| 25|drds_db_1516187088365daui|100.100.64.1:59077|drds_db_xzip_0007|hash_tb_15   |  0.359375|1.8
1% |         |
+------+-------------------------+-------------------+-------------------+---------------+------------+--------+---------------+
```

16.6. SHOW PROCESSLIST指令与KILL指令

获取 DRDS 版本号，DRDS 自助升级的方法以及更多的版本介绍请参考版本说明文档。

背景信息

操作步骤

1. 当 DRDS 版本号小于 5.1.28-1408022 时，DRDS 仅支持物理连接的 SHOW PROCESSLIST 与 KILL 功能，请参考老版本文档老版本 SHOW PROCESSLIST 指令与 KILL 指令。
2. 当 DRDS 版本号大于等于 5.1.28-1408022 时，DRDS 支持逻辑连接与物理连接的 SHOW PROCESSLIST 与 KILL 功能，请继续阅读此文档。

执行结果

SHOW PROCESSLIST 指令

DRDS 中，可以使用 `SHOW PROCESSLIST` 指令查看 DRDS 中的连接与正在执行的 SQL 等信息。

语法：

```
SHOW [FULL] PROCESSLIST
```

示例：

```
mysql> SHOW PROCESSLIST\G
   ID: 1971050
 USER: admin
 HOST: 111.111.111.111:4303
   DB: drds_test
COMMAND: Query
 TIME: 0
 STATE:
 INFO: show processlist
1 row in set (0.01 sec)
```

结果集各字段含义：

- ID：连接的 ID，为一个 Long 型数字。
- USER：建立此连接所使用的用户名。
- HOST：建立此连接的机器的 IP 与端口。
- DB：此连接所访问的库名。
- COMMAND，目前有两种取值：
  - Query，代表当前连接正在执行 SQL 语句；
  - Sleep，代表当前连接正处于空闲状态。
- TIME， 连接处于当前状态持续的时间：
  - 当 COMMAND 为 Query 时，代表当此连接上正在执行的 SQL 已经执行的时间；
  - 当 COMMAND 为 Sleep 时，代表当此连接空闲的时间。
- STATE：目前无意义，恒为空值。
- INFO：
  - 当 COMMAND 为 Query 时，为此连接上正在执行的 SQL 的内容。当不带 FULL 参数时，最多返回正在执行的 SQL 的前 30 个字符。当带 FULL 参数时，最多返回正在执行的 SQL 的前1000个字符；
  - 当 COMMAND 为其他值时，无意义，为空值。

SHOW PHYSICAL_PROCESSLIST 指令

DRDS 中，可以使用 `SHOW PHYSICAL_PROCESSLIST` 指令查看底层所有 MySQL/RDS 上正在执行的 SQL 信息。

语法：

```
SHOW [FULL] PHYSICAL_PROCESSLIST
```

当 SOL 比较长的时候，`SHOW PHYSICAL_PROCESSLIST` 会截断，这时可以使用 `SHOW FULL PHYSICAL_PROCESSLIST` 获取完整 SQL。

返回结果中每一列的含义与 MySQL 的 `SHOW PROCESSLIST` 指令等价，请参考 SHOW PROCESSLIST Syntax。

注意：与 MySQL 不同，DRDS 返回的物理连接的 ID 列为一个字符串，并非一个数字。

示例：

```
mysql> SHOW PHYSICAL_PROCESSLIST\G
*************************** 1. row ***************************
ID: 0-0-521414
USER: tddl5
DB: tddl5_00
COMMAND: Query
TIME: 0
STATE: init
INFO: show processlist
*************************** 2. row ***************************
ID: 0-0-521570
USER: tddl5
DB: tddl5_00
COMMAND: Query
TIME: 0
STATE: User sleep
INFO: /*DRDS /88.88.88.88/b67a0e4d8800000/ */ select sleep(1000)
2 rows in set (0.01 sec)
```

KILL 指令

KILL 指令用于终止一个正在执行的 SQL。

DRDS 使用 DRDS 在 MySQL/RDS 上创建的用户名连接 MySQL/RDS，所以一般直接连接 MySQL/RDS 是没有权限对 DRDS 发起的请求进行 KILL 操作的。

如果需要终止一个 DRDS 上正在执行的 SQL，需要使用 MySQL 命令行、DMS 等工具连接 DRDS，在 DRDS 上执行 KILL 指令。

语法：

```
KILL PROCESS_ID | 'PHYSICAL_PROCESS_ID' | 'ALL'
```

有三种用法：

1. **终止一个特定的逻辑 SQL：** `KILL PROCESS_ID`

   PROCESS_ID 为 `SHOW [FULL] PROCESSLIST` 指令返回的 ID 列。

   DRDS 中，`KILL PROCESS_ID` 指令会将此连接正在执行的逻辑 SQL 与物理 SQL 均终止掉，并断开此连接。

   DRDS 不支持 `KILL QUERY` 指令。
2. **终止一个特定的物理 SQL：** `KILL 'PHYSICAL_PROCESS_ID'`

   其中的 PHYSICAL_PROCESS_ID 来自 `SHOW PHYSICAL_PROCESS_ID` 指令返回的 ID 列。

注意：由于 PHYSICAL_PROCESS_ID 列为一个字符串，并非一个数字，因此 KILL 指令中，PHYSICAL_PROCESS_ID 需要使用单引号括起来。

示例：

```
mysql> KILL '0-0-521570';
Query OK, 0 rows affected (0.01 sec)
```

3. **终止当前库上所有通过 DRDS 执行的物理 SQL：** `KILL 'ALL'`

   当底层 MySQL/RDS 因为一些 SQL 导致压力非常大的时候，可以使用 `KILL 'ALL'` 指令终止当前 DRDS 库上所有正在执行的物理 SQL。

   符合以下条件的物理 PROCESS 会被 `KILL 'ALL'` 指令终止：

   - 该 PROCESS 的 User 是 DRDS 在 MySQL/RDS 上所创建的用户名；
   - 该 PROCESS 正在执行查询，也即 COMMAND 为 Query。

16.7. 老版本 SHOW PROCESSLIST 指令与 KILL 指令

获取 DRDS 版本号，DRDS 自助升级的方法以及更多的版本介绍请参考版本说明文档。

背景信息

操作步骤

1. 当 DRDS 版本号小于 5.1.28-1408022 时，DRDS 仅支持物理连接的 SHOW PROCESSLIST 与 KILL 功能，请继续阅读此文档。
2. 当 DRDS 版本号大于等于 5.1.28-1408022 时，DRDS 支持逻辑连接与物理连接的 SHOW PROCESSLIST 与 KILL 功能，请参考文档 SHOW PROCESSLIST 指令与 KILL 指令。

执行结果

SHOW PROCESSLIST 指令

DRDS 中，可以使用 `SHOW PROCESSLIST` 指令查看底层所有 MySQL/RDS 上正在执行的 SQL 信息。

语法：

```
SHOW [FULL] PROCESSLIST
```

当 SQL 比较长的时候，`SHOW PROCESSLIST` 会截断，这时可以使用 `SHOW FULL PROCESSLIST` 获取完整 SQL。

返回结果中每一列的含义与 MySQL 的 `SHOW PROCESSLIST` 指令等价，请参考 SHOW PROCESSLIST Syntax。

示例：

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
      ID: 0-0-521414
    USER: tddl5
      DB: tddl5_00
 COMMAND: Query
    TIME: 0
   STATE: init
    INFO: show processlist
 ROWS_SENT: NULL
ROWS_EXAMINED: NULL
 ROWS_READ: NULL
*************************** 2. row ***************************
      ID: 0-0-521570
    USER: tddl5
      DB: tddl5_00
 COMMAND: Query
    TIME: 0
   STATE: User sleep
    INFO: /*DRDS /88.88.88.88/b67a0e4d8800000/ */ select sleep(1000)
 ROWS_SENT: NULL
ROWS_EXAMINED: NULL
 ROWS_READ: NULL
2 rows in set (0.01 sec)
```

KILL 指令

KILL 指令用于终止一个正在执行的 SQL。

DRDS 使用 DRDS 在 MySQL/RDS 上创建的用户名连接 MySQL/RDS，所以一般直接连接 MySQL/RDS 是没有权限对 DRDS 发起的请求进行 KILL 操作的。

如果需要终止一个 DRDS 上正在执行的 SQL，需要使用 MySQL 命令行、DMS 等工具连接 DRDS，在 DRDS 上执行 KILL 指令。

语法：

```
KILL 'PROCESS_ID' | 'ALL'
```

有两种用法：

1. **终止一个特定的 SQL：** `KILL 'PROCESS_ID'`

   其中的 PROCESS_ID 来自 `SHOW PROCESSLIST` 指令返回的 ID 列。

   注意：与 MySQL 不同，DRDS 返回的 ID 列为一个字符串，并非一个数字，因此 KILL 指令中，PROCESS_ID 需要使用单引号括起来。

   示例：

   ```
   mysql> KILL '0-0-521570';
   Query OK, 0 rows affected (0.01 sec)
   ```

2. **终止当前库上所有通过 DRDS 执行的 SQL：** `KILL 'ALL'`

当底层 MySQL/RDS 因为一些 SQL 导致压力非常大的时候，可以使用 `KILL 'ALL'` 指令终止当前 DRDS 库上所有正在执行的 SQL。

符合以下条件的 PROCESS 会被 `KILL 'ALL'` 指令终止：

- 该 PROCESS 的 User 是 DRDS 在 MySQL/RDS 上所创建的用户名；
- 该 PROCESS 正在执行查询，即 COMMAND 为 Query。

注意：较低版本的 DRDS 实例不支持 `KILL 'ALL'` 功能，执行的时候会报错。可以将 DRDS 实例升级到最新版本，升级方法请参考升级实例版本。

16.8. 回收站管理

DRDS 支持表回收站功能，详见文档：表回收站，本文档将介绍如何通过命令行管理表回收站。

背景信息

操作指令

强制删除表

通过在 DROP 语句末添加 PURGE 参数强制删除表，不放入回收站

```
DROP TABLE table-name PURGE
```

展示回收站

```
SHOW recyclebin
mysql> show recyclebin;
+------+--------------------------+---------------+---------------------+
| ID   | NAME                     | ORIGINAL_NAME | CREATED             |
+------+--------------------------+---------------+---------------------+
|    0 | BIN_2FGVWOE1F8JDYTR0EKIQ | rb1           | 2018-07-05 10:23:17 |
+------+--------------------------+---------------+---------------------+
1 row in set (0.01 sec)
```

恢复表

```
FLASHBACK TABLE bin-name TO BEFORE DROP [RENAME TO new-name]
mysql> flashback table BIN_2FGVWOE1F8JDYTR0EKIQ to before drop;
Query OK, 0 rows affected (0.14 sec)
```

清理表

```
PURGE TABLE bin-name
mysql> purge table BIN_2B0KUPYRZJ6C1UV4BBYF;
Query OK, 0 rows affected (0.13 sec)
```

清空回收站

```
PURGE recyclebin
```

17.自定义HINT

17.1. HINT简介

HINT 作为一种 SQL 补充语法，在关系型数据库中扮演着非常重要的角色。它允许用户通过相关的语法影响 SQL 的执行方式，对 SQL 进行特殊的优化。同样，DRDS 也提供了特殊的 HINT 语法。

背景信息

本文适用于 DRDS 5.3 及以上版本，其他版本请参考DRDS 5.2 HINT 简介

例如，假设已知目标数据在某些分库的分表中，需要直接将 SQL 下发到该分库执行，就可以使用 DRDS 自定义 HINT 来完成。

```
SELECT /*+TDDL:node('node_name')*/ * FROM table_name;
```

这个 SQL 语句中 `/*` 和 `*/` 之间的语句就是 DRDS 的自定义 HINT，即 `+TDDL:node('node_name')`，它指定了 SQL 语句在特定的 RDS 分库上执行。

注意：

- **DRDS 自定义 HINT 支持 `/*+TDDL:hint_command*/` 和 `/!+TDDL:hint_command*/` 两种格式。**
- **如果使用 `/*+TDDL:hint_command*/` 格式，在使用 MySQL 官方命令行客户端执行带有 DRDS 自定义 HINT 的 SQL 时，请在登录命令中加上 `-c` 参数。否则，由于 DRDS 自定义 HINT 是以 MySQL 注释 形式使用的，该客户端会将注释语句删除后再发送到服务端执行，导致 DRDS 自定义 HINT 失效。具体请查看 MySQL 官方客户端命令。**

DRDS 自定义 HINT 语法

基本语法：

```
/*+TDDL: hint_command [hint_command ...]*/
/!+TDDL: hint_command [hint_command ...]*/
```

DRDS 自定义 HINT 基于MySQL 注释，HINT 语句位于 `/*` 与 `*/` 或 `/!` 与 `*/` 之间，并且必须以 `+TDDL:` 开头。其中 `hint_command` 是 DRDS 自定义 HINT 命令，与具体的操作相关，多个 `hint_command` 之间使用空格分割。

例子：

```
# 查询每个分库中的物理表名
/*+TDDL:scan()*/SHOW TABLES;
# 将查询下发到 RDS 只读实例的 0000 分库上
/*+TDDL:node(0) slave()*/SELECT * FROM t1;
```

例子中 `/*+TDDL:scan()*/` 和 `/*+TDDL:node(0) slave()*/` 为 DRDS 自定义 HINT 部分，以 `+TDDL:` 开头。`scan()`、`node(0)`、`slave()` 为 DRDS 自定义 HINT 命令，多个 HINT 命令之间使用空格分割。

在 SQL 语句中使用 HINT：

DRDS 支持在 DML、DDL、DAL 语句中使用 HINT，具体语法如下：

- 对于所有支持 HINT 的语句，允许在语句前指定 HINT，如

```
/*+TDDL: ... */ SELECT ...
/*+TDDL: ... */ INSERT ...
/*+TDDL: ... */ REPLACE ...
/*+TDDL: ... */ UPDATE ...
/*+TDDL: ... */ DELETE ...
/*+TDDL: ... */ CREATE TABLE ...
/*+TDDL: ... */ ALTER TABLE ...
/*+TDDL: ... */ DROP TABLE ...
/*+TDDL: ... */ SHOW ...
...
```

- 对于 DML 语句，允许在首个关键字之后指定 HINT，如

```
SELECT /*+TDDL: ... */ ...
INSERT /*+TDDL: ... */ ...
REPLACE /*+TDDL: ... */ ...
UPDATE /*+TDDL: ... */ ...
DELETE /*+TDDL: ... */ ...
...
```

注意：不同 HINT 支持的语句范围可能不同，实际支持情况请参考具体 HINT 命令说明文档

使用多个 HINT：

DRDS 支持在 HINT 语句中使用多个 HINT 命令

```
SELECT /*+TDDL:node(0) slave()*/ ...;
```

DRDS 不支持通过以下方式使用多个 HINT 命令

```
# 不支持 单条 SQL 语句中包含多个 HINT 语句
SELECT /*+TDDL:node(0)*/ /*+TDDL:slave()*/ ...;
# 不支持 HINT 语句中 包含重复的 HINT 命令
SELECT /*+TDDL:node(0) node(1)*/ ...;
```

DRDS 自定义 HINT 分类

根据操作类型的不同，DRDS 的自定义 HINT 主要可以分为以下几类：

- 读写分离
- 自定义 SQL 超时时间
- 指定分库执行 SQL
- 扫描全部分库分表

DRDS 自定义 HINT 兼容性

DRDS 5.3 及以上版本，向下兼容大部分DRDS 5.2 自定义 HINT，详细对照关系如下

DRDS 5.2 HINT	支持情况	对应的 DRDS 5.3 HINT
读写分离	支持 5.2 HINT 语法	5.3 读写分离
备库延迟切断	不支持，添加后不产生效果	无
自定义 SQL 超时时间	支持 5.2 HINT 语法	自定义 SQL 超时时间
指定分库执行 SQL	部分支持，"通过分库键值指定 SQL 在分库上执行" 迁移至 扫描全部分库分表	指定分库执行 SQL
扫描全部分库分表	支持，增加 “根据条件计算物理表名称” 和 “显式指定物理表名” 功能	扫描全部分库分表

17.2. 自定义SQL超时时间

在 DRDS 中，DRDS 节点与 RDS 的默认的 SQL 执行超时时间是 900 秒（可以调整），但是对于某些特定的慢 SQL，其执行时间可能超过了 900 秒 。针对这种慢 SQL，DRDS 提供了调整超时时间的自定义 HINT。通过这个自定义 HINT 可以任意调整 SQL 执行时长。

背景信息

本文适用于 DRDS 5.3 及以上版本，其他版本请参考DRDS 5.2 自定义 SQL 超时时间 HINT 文档

DRDS 自定义 SQL 超时时间 HINT 的语法如下：

```
/*+TDDL:SOCKET_TIMEOUT(time)*/
```

其中，`SOCKET_TIMEOUT` 的单位是毫秒。通过该 HINT 用户可以根据业务需要，自由调整 SQL 语句的超时时间。

注意：

- **DRDS 自定义 HINT 支持 `/*+TDDL:hint_command*/` 和 `/!+TDDL:hint_command*/` 两种格式。**
- **如果使用 `/*+TDDL:hint_command*/` 格式，在使用 MySQL 官方命令行客户端执行带有 DRDS 自定义 HINT 的 SQL 时，请在登录命令中加上 `-c` 参数。否则，由于 DRDS 自定义 HINT 是以 MySQL 注释 形式使用的，该客户端会将注释语句删除后再发送到服务端执行，导致 DRDS 自定义 HINT 失效。具体请查看 MySQL 官方客户端命令。**

示例

- 设置 SQL 超时时间为40秒：

```
/*+TDDL:SOCKET_TIMEOUT(40000)*/SELECT * FROM t_item;
```

注意：超时时间设置得越长，占用数据库资源的时间就会越长。如果同一时间长时间执行的 SQL 过多，可能消耗大量的数据库资源，从而导致无法正常使用数据库服务。所以，对于长时间执行的 SQL 语句，尽量对 SQL 语句进行优化。

17.3. 扫描全部或部分分库分表

除了可以将SQL单独下发到一个或多个分库执行，POlarDB-X还提供了扫描全部/部分分库与分表的 SCAN HINT 。使用 SCAN HINT ，您可以一次将SQL下发到每一个分库执行，比如查看某个分库上的所有分表，或者查看某个逻辑表的每张物理表中的数据量等。

背景信息

本文适用于PolarDB-X 5.3及以上版本，其他版本请参考PolarDB-X 5.2扫描全部分库分表 HINT文档

通过 SCAN HINT ，可以指定如下四种执行SQL的方式：

- 在所有分库的所有分表上执行。
- 在指定分库的所有分表上执行。
- 在指定分库分表上执行，根据条件计算物理表名称。
- 在指定分库分表上执行，显式指定物理表名。

SCAN HINT 支持DML、DDL和部分DAL语句。

- POlarDB-X自定义HINT支持 /*+TDDL:hint_command*/ 和 /!+TDDL:hint_command*/ 两种格式。
- 如果使用 /*+TDDL:hint_command*/ 格式，在使用MySQL官方命令行客户端执行带有POlarDB-X自定义HINT的SQL时，请在登录命令中加上 -c 参数。否则，由于POlarDB-X自定义HINT是以MySQL注释形式使用的，该客户端会将注释语句删除后再发送到服务端执行，导致POlarDB-X自定义HINT失效。具体请查看MySQL官方客户端命令。

语法

```
# SCAN HINT
# 将 SQL 语句下发到所有分库的所有分表上执行
SCAN()
# 将 SQL 语句下发到指定分库的所有分表上执行
SCAN(NODE="node_list")          # 指定分库
# 将 SQL 语句下发到指定分库分表上执行，根据条件计算物理表名称
SCAN(
 [TABLE=]"table_name_list"      # 逻辑表名
 ,CONDITION="condition_string"  # 使用 TABLE 和 CONDITION 中的内容计算物理库表名称
 [,NODE="node_list"] )          # 过滤通过 CONDITION 计算出的结果，仅保留指定物理库
# 将 SQL 语句下发到指定分库分表上执行，显式指定物理表名
SCAN(
 [TABLE=]"table_name_list"      # 逻辑表名
 ,REAL_TABLE=("table_name_list") # 物理表名，对所有物理库使用相同的物理表名
 [,NODE="node_list"] )          # 过滤通过 CONDITION 计算出的结果，仅保留指定物理库
# 物理/逻辑表名列表
table_name_list:
  table_name [,table_name]...
# 物理库列表，支持 GROUP_KEY 和 GROUP 的序号,可以通过 `SHOW NODE` 语句获得
node_list:
  {group_key | group_index} [,{group_key | group_index}]...
# 支持 SQL WHERE 的语法，需要为每一张表设置条件，如：t1.id = 2 and t2.id = 2
condition_string:
  where_condition
```

从5.4.1版本开始，POlarDB-X在拆分表的物理表名中增加了4个字符的随机串，请务必使用 SHOW TOPOLOGY 命令获取逻辑表拓扑和实际的物理表名。

示例

- 在所有分库的所有分表上执行

```
SELECT /*+TDDL:scan()*/ COUNT(1) FROM t1
```

执行后会下发SQL语句到 t1 的所有物理表上执行，并将结果集合并后返回。

- 在指定分库的所有分表上执行

```
SELECT /*+TDDL:scan(node='0,1,2')*/ COUNT(1) FROM t1
```

执行后会首先计算出 t1 在 0000、0001和0002分库上的所有物理表，然后下发SQL语句并将结果集合并后返回。

- 按条件在指定分表上执行

```
SELECT /*+TDDL:scan('t1', condition='t1.id = 2')*/ COUNT(1) FROM t1
```

执行后会首先计算出逻辑表 t1 满足 condition 条件的所有物理表，然后下发SQL语句并将结果集合并后返回。

- 按条件在指定分表上执行，有JOIN的情况

```
SELECT /*+TDDL:scan('t1, t2', condition='t1.id = 2 and t2.id = 2')*/ * FROM t1 a JOIN t2 b ON a.id = b.id WHERE b.name = "test"
```

执行后会首先计算出逻辑表 t1 和 t2 满足 condition 条件的所有物理表，然后下发SQL语句并将结果集合并后返回。

使用该自定义注释需要保证两张表的分库和分表数量一致，否则POlarDB-X计算出的两个键值对应的分库不一致，就会报错。

- 在指定分库分表上执行，显式指定物理表名

```
SELECT /*+TDDL:scan('t1', real_table=("t1_00", "t1_01"))*/ COUNT(1) FROM t1
```

执行后会下发SQL语句到所有分库的 t1_00 和 t1_01 分表上，合并结果集后返回。

- 在指定分库分表上执行，显式指定物理表名，有JOIN的情况

```
SELECT /*+TDDL:scan('t1, t2', real_table=("t1_00,t2_00", "t1_01,t2_01"))*/ * FROM t1 a JOIN t2 b ON a.id = b.id WHERE b.name = "test";
```

执行后会下发SQL语句到所有分库的 t1_00 、 t2_00 、 t1_01 和 t2_01 分表上，合并结果集后返回。

17.4. 指定分库执行 SQL

在使用 DRDS 的过程中，如果遇到某个 DRDS 不支持的 SQL 语句，可以通过 DRDS 提供的 NODE HINT ，直接将 SQL 下发到一个或多个分库上去执行。此外如果需要单独查询某个分库或者已知分库的某个分表中的数据，也可以使用 NODE HINT ，直接将 SQL 语句下发到分库中执行。

背景信息

本文适用于 DRDS 5.3 及以上版本，其他版本请参考DRDS 5.2 指定分库执行 SQL HINT 文档

NODE HINT 支持通过分片名指定 SQL 在分库上执行。其中分片名是 DRDS 中分库的唯一标识，可以通过 SHOW NODE 语句得到。

通过分库名指定 SQL 在分库上执行

通过分库名指定 SQL 在分库上执行分两种使用方式，分别是指定 SQL 在某个分库上执行和指定 SQL 在多个分库上执行。

注意：如果在目标表包含 Sequence 的 INSERT 语句上使用了指定分库的 HINT，那么 Sequence 将不生效。更多相关信息，请参考 Sequence 限制及注意事项。

- 指定 SQL 在某个分库上执行：

  ```
  /*+TDDL:node('node_name')*/
  ```

  node_name 为分片名，通过这个 DRDS 自定义 HINT，就可以将 SQL 下发到 node_name 对应的分库中执行。

- 指定 SQL 在多个分库上执行：

  ```
  /*+TDDL:node('node_name'[,'node_name1','node_name2'])*/
  ```

  在参数中指定多个分片名，将 SQL 下发到多个分库上执行，分片名之间使用逗号分隔。

注意：

- 使用该自定义 HINT 时，DRDS 会将 SQL 直接下发到分库上执行，所以在 SQL 语句中，表名必须是该分库中已经存在的表名。
- NODE HINT 支持 DML、DDL、DAL 语句。

注意：

- 从版本 5.4.1 开始，DRDS 在拆分表的物理表名中增加了4个字符的随机串，请务必使用 SHOW TOPOLOGY 命令获取逻辑表拓扑和实际的物理表名。
- 从版本 5.4.4 开始，DRDS 提供开关来控制拆分表的物理表名中是否包含随机串，默认为开启，可以在控制台“参数设置”的数据库级别参数中，将“是否启用随机物理表名 ENABLE_RANDOM_PHY_TABLE_NAME”改为 false 来关闭，也可以用 HINT 来实现语句级别的控制：/*+TDDL:cmd_extra(ENABLE_RANDOM_PHY_TABLE_NAME=FALSE)*/

注意：

- **DRDS 自定义 HINT 支持 /*+TDDL:hint_command*/ 和 /!+TDDL:hint_command*/ 两种格式。**
- **如果使用 /*+TDDL:hint_command*/ 格式，在使用 MySQL 官方命令行客户端执行带有 DRDS 自定义 HINT 的 SQL 时，请在登录命令中加上 -c 参数。否则，由于 DRDS 自定义 HINT 是以 MySQL 注释 形式使用的，该客户端会将注释语句删除后再发送到服务端执行，导致 DRDS 自定义 HINT 失效。具体请查看 MySQL 官方客户端命令。**

示例

对于名为 drds_test 的 DRDS 数据库， SHOW NODE 的结果如下：

```
mysql> SHOW NODE\G
*************************** 1. row ******************
           ID: 0
         NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
```

```
 MASTER_READ_COUNT: 212
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 2. row ******************
         ID: 1
       NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 3. row ******************
         ID: 2
       NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 4. row ******************
         ID: 3
       NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 5. row ******************
         ID: 4
       NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 6. row ******************
         ID: 5
       NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 7. row ******************
         ID: 6
       NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 8. row ******************
         ID: 7
       NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
8 rows in set (0.02 sec)
```

可以看到每个分库都有 NAME 这个属性，这就是分库的分片名。每个分片名都唯一对应一个分库名，比如 DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS 这个分片名对应的分库名是 drds_test_vtla_0003 。得到了分片名，就可以使用 DRDS 的自定义 HINT 指定分库执行 SQL 语句了。

- 指定 SQL 在第 0 个分库上执行：

```
SELECT /*TDDL:node('DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS')*/ * FROM table_name;
```

- 指定 SQL 在多个分库上执行：

```
SELECT /*TDDL:node('DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS','DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS')*/ * FROM table_name;
```

这条 SQL 语句将在 DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS ， DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS 这两个分片上执行。

- 查看 SQL 在第 0 个分库上物理执行计划：

```
/*TDDL:node('DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS')*/ EXPLAIN SELECT * FROM table_name; ```
```

17.5. 读写分离

DRDS 提供了一种针对应用层透明的读写分离实现。但是由于 RDS 主实例与只读实例之间数据的同步存在着毫秒级别的延迟，如果在主库中变更以后需要马上读取变更的数据，则需要保证将读取数据的 SQL 下发到主实例中。针对这种需求，DRDS 提供了读写分离自定义 HINT，指定将 SQL 下发到主实例或者只读实例。

背景信息

本文适用于 DRDS 5.3 及以上版本，早期版本请参考DRDS 5.2 HINT 文档

```
/*+TDDL:
  master()
  |slave()
*/
```

在该自定义 HINT 中可以指定 SQL 是在主实例上执行还是在只读实例上执行。对于 /*+TDDL:slave()*/ ，如果一个主 RDS 实例存在多个只读实例，那么 DRDS 会根据所分配的权重随机选择一个只读实例执行 SQL 语句。

注意：

- **DRDS 自定义 HINT 支持 /*+TDDL:hint_command*/ 和 /!+TDDL:hint_command*/ 两种格式。**
- **如果使用 /*+TDDL:hint_command*/ 格式，在使用 MySQL 官方命令行客户端执行带有 DRDS 自定义 HINT 的 SQL 时，请在登录命令中加上 -c 参数。否则，由于 DRDS 自定义 HINT 是以 MySQL 注释 形式使用的，该客户端会将注释语句删除后再发送到服务端执行，导致 DRDS 自定义 HINT 失效。具体请查看 MySQL 官方客户端命令。**

示例

- 指定 SQL 在主实例上执行：

```
SELECT /*+TDDL:master()*/ * FROM table_name;
```

在 SQL 第一个关键字之后添加 `/*+TDDL:master()*/` 这个自定义 HINT 后，这条 SQL 将被下发到主实例上执行。

- 指定 SQL 在只读实例上执行：

```
SELECT /*+TDDL:slave()*/ * FROM table_name;
```

在 SQL 第一个关键字之后添加 `/*+TDDL:slave()*/` 这个自定义 HINT 后，这条 SQL 将会根据所分配的权重被随机下发到某个只读实例上执行。

注意：

- 此读写分离自定义 HINT 仅仅针对非事务中的读 SQL 语句生效，如果 SQL 语句是写 SQL 或者 SQL 语句在事务中，那么还是会下发到 RDS 的主实例执行。
- DRDS 针对 `/*+TDDL:slave()*/` 自定义 HINT，会从只读实例中按照权重随机选取一个下发 SQL 语句执行。若只读实例不存在时，不会报错，而是选取主实例执行。

18.SQL调优指南

18.1. 查询慢SQL明细

PolarDB-X将执行时间超过1秒的SQL定义为慢SQL，包括逻辑慢SQL和物理慢SQL。本文介绍这两种慢SQL的定义和查询方式。

背景信息

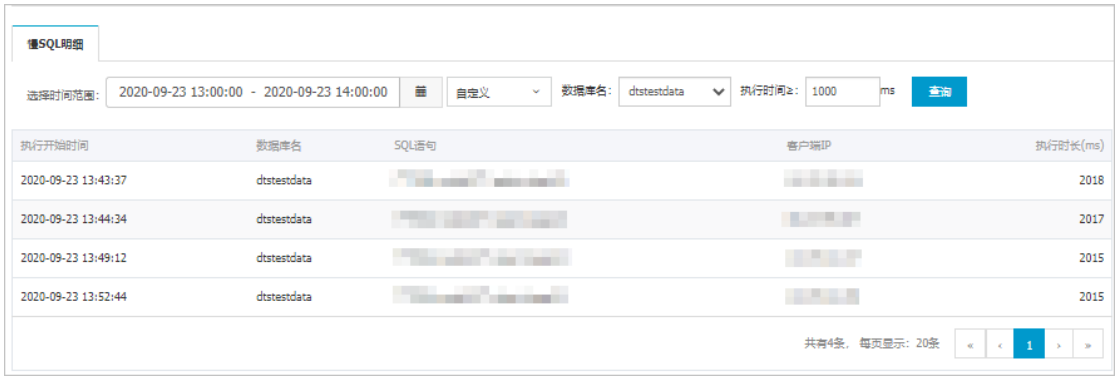
逻辑慢SQL和物理慢SQL的定义如下：

- 逻辑慢SQL：客户端发送到PolarDB-X的慢SQL。
- 物理慢SQL：PolarDB-X发送到存储层MySQL的慢SQL。

每个PolarDB-X节点最多保存5000条慢SQL明细，超过限制数量的慢SQL明细会被滚动删除。

操作步骤

1. 登录云原生分布式数据库控制台。
2. 在页面左上角选择目标实例所在地域。
3. 在左侧导航栏中，单击**实例列表**。
4. 找到目标实例，单击实例ID，进入目标实例**基本信息**页。
5. 您可以查看逻辑慢SQL的详细信息，具体操作步骤如下：
   i. 单击左侧导航栏的**慢SQL明细**。
   ii. 选择**时间范围**、**数据库名**及**执行时间**，单击**查询**，可查看慢SQL信息。

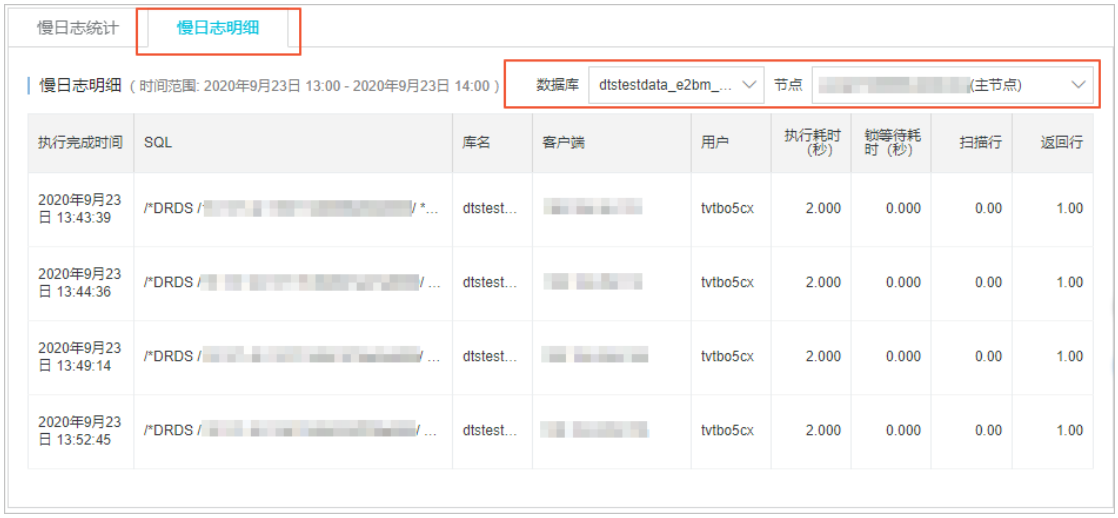


   iii. 选择**时间范围**、**数据库名**及**执行时间**，单击**查询**，可查看慢SQL信息。
6. 在PolarDB-X上定位到逻辑慢SQL，您也可以存储层MySQL上找到相应的物理慢SQL信息，具体步骤如下：

> **说明** 目前只支持查询存储层为PolarDB MySQL上的物理慢SQL。

   i. 单击左侧导航栏的**数据库管理**，选择目标数据库。
   ii. 单击**存储管理**，选择目标集群，进入存储层MySQL**基本信息**界面。
   iii. 在左侧导航栏中，单击**慢SQL**。
   iv. 在右上角设置时间，单击**查看**，可以查询特定时间范围的**慢日志趋势**、**慢日志统计**和**慢日志明细**。

v. 单击 **慢日志明细**，选择目标数据库和主节点，可以查看慢日志详细信息。



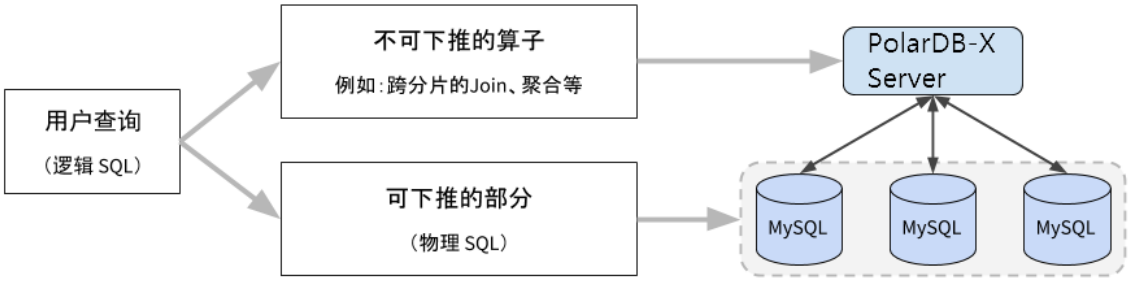
> **说明** 更多慢SQL分析诊断功能介绍，详情请参见慢SQL。

18.2. SQL调优基础概念

在使用PolarDB-X的过程中，可能出现性能不符合预期的慢SQL。SQL调优的过程，就是通过分析SQL的执行计划、各阶段运行时长等信息，找出导致SQL执行慢的原因，继而解决问题。

PolarDB-X架构

PolarDB-X是一个支持计算存储分离架构的数据库产品。当一条查询SQL（称为逻辑SQL）发往PolarDB-X节点时，PolarDB-X会将其分成可下推的、和不可下推的两部分，可下推的部分也被称为物理SQL。



原则上，PolarDB-X会：

- 尽可能将用户SQL下推到MySQL上执行。
- 对于无法下推的部分算子，选择最优的方式来执行。

下推和执行计划

EXPLAIN指令将会打印SQL的执行计划。它的用法非常简单：只要在SQL最前面加上 `EXPLAIN` 即可。下面通过几个例子来展示PolarDB-X的执行方式。

示例一：

```
> explain select c_custkey, c_name, c_address from customer where c_custkey = 42;
LogicalView(tables="customer_2", sql="SELECT `c_custkey`, `c_name`, `c_address` FROM `customer` AS `customer` WHERE (`c_custkey` = ?)")
```

对于带有主键条件的查询来说，PolarDB-X只要将SQL直接下发到主键对应的分片上即可。因此，执行计划中只有一个 LogicalView 算子，下发的物理SQL基本和逻辑SQL是一样的。

LogicalView 算子代表下推到MySQL执行的查询，请参见执行计划和基本算子。

示例二：

```
> explain select c_nationkey, count(*) from customer group by c_nationkey;
HashAgg(group="c_nationkey", count(*)="SUM(count(*))")
  Gather(concurrent=true)
    LogicalView(tables="customer_[0-7]", shardCount=8, sql="SELECT `c_nationkey`, COUNT(*) AS `count(*)` FROM `customer` AS `customer` GROUP BY `c_nationkey`")
```

上述查询会统计：各个国家的客户数量分别有多少？考虑到分库分表，可以将它分成两个阶段来进行：

1. 在各个分表上进行 COUNT(*) 统计（这一步可被下推到MySQL上执行）。
2. 将结果汇总，将 COUNT(*) 的结果执行 SUM() ，得到最终的结果（这一步需要PolarDB-X节点来完成）。

从执行计划上，也可以看出这一点。其中：

- LogicalView 表示下发到各个分片的SQL按nation分组进行 COUNT(*) 统计。
- Gather 算子表示收集各个分片的结果。
- HashAgg 是聚合（Aggregate）的一种实现方式，以 c_nationkey 作为分组键将 COUNT(*) 的结果求 SUM() 。

更多的例子，详情请参见执行计划和基本算子。

18.3. SQL调优方法与实战

找出需调优的慢SQL后，先通过 EXPLAIN 查看执行计划，然后通过如下方法优化SQL：下推更多计算至存储层MySQL，适当增加索引，优化执行计划。

下推更多的计算

PolarDB-X会尽可能将更多的计算下推到存储层MySQL。下推计算能够减少数据传输，减少网络层和PolarDB-X层的开销，提升SQL语句的执行效率。PolarDB-X支持下推几乎所有算子，包括：

- 过滤条件，如 WHERE 或 HAVING 中的条件。
- 聚合算子，如 COUNT ， GROUP BY 等，会分成两阶段进行聚合计算。
- 排序算子，如 ORDER BY 。
- JOIN和子查询，两边JOIN Key分片方式必须一样，或其中一边为广播表。

如下示例讲解如何将更多的计算下推到MySQL来加速执行

```
> EXPLAIN select * from customer, nation where c_nationkey = n_nationkey and n_regionkey = 3;
Project(c_custkey="c_custkey", c_name="c_name", c_address="c_address", c_nationkey="c_nationkey", c_
phone="c_phone", c_acctbal="c_acctbal", c_mktsegment="c_mktsegment", c_comment="c_comment", n_
nationkey="n_nationkey", n_name="n_name", n_regionkey="n_regionkey", n_comment="n_comment")
 BKAJoin(condition="c_nationkey = n_nationkey", type="inner")
  Gather(concurrent=true)
   LogicalView(tables="nation", shardCount=2, sql="SELECT * FROM `nation` AS `nation` WHERE (`n_regi
onkey` = ?)")
  Gather(concurrent=true)
   LogicalView(tables="customer_[0-7]", shardCount=8, sql="SELECT * FROM `customer` AS `customer` W
HERE (`c_nationkey` IN ('?'))")
```

若执行计划中出现了BKAJOIN，BKAJOIN每次从左表获取一批数据，就会拼成一个 `IN` 查询取出右表相关联的行，并在最后执行JOIN操作。由于左表数据量很大，需要取很多次才能完成查询，执行很慢。

无法下推JOIN的原因是：当前情况下，nation是按主键 `n_nationkey` 切分的，而本查询的JOIN Key是 `c_custkey` ，二者不同，所以下推失败。

考虑到nation （国家）表数据量并不大、且几乎没有修改操作，可以将其重建成如下广播表：

```
--- 修改后 ---
CREATE TABLE `nation` (
 `n_nationkey` int(11) NOT NULL,
 `n_name` varchar(25) NOT NULL,
 `n_regionkey` int(11) NOT NULL,
 `n_comment` varchar(152) DEFAULT NULL,
 PRIMARY KEY (`n_nationkey`)
) BROADCAST; --- 声明为广播表
```

修改后，可以看到执行计划中不再出现JOIN，几乎所有计算都被下推到存储层MySQL执行了（ `LogicalView` 中），而上层仅仅是将结果收集并返回给用户（ `Gather` 算子），执行性能大大增强。

```
> EXPLAIN select * from customer, nation where c_nationkey = n_nationkey and n_regionkey = 3;
Gather(concurrent=true)
 LogicalView(tables="customer_[0-7],nation", shardCount=8, sql="SELECT * FROM `customer` AS `custom
er` INNER JOIN `nation` AS `nation` ON ((`nation`.`n_regionkey` = ?) AND (`customer`.`c_nationkey` =
`nation`.`n_nationkey`))")
```

更多关于下推的原理和优化，请参见查询改写与下推。

增加索引

如果下推SQL中出现（物理）慢SQL，可以给分表增加索引来解决，这里不再详述。

PolarDB-X自5.4.1 版本开始支持全局二级索引，可以通过增加GSI的方式使逻辑表拥有多个拆分维度。

下面以一个慢SQL作为例子来讲解如何通过GSI下推更多算子。

```
> EXPLAIN select o_orderkey, c_custkey, c_name from orders, customer
      where o_custkey = c_custkey and o_orderdate = '2019-11-11' and o_totalprice > 100;
Project(o_orderkey="o_orderkey", c_custkey="c_custkey", c_name="c_name")
 HashJoin(condition="o_custkey = c_custkey", type="inner")
  Gather(concurrent=true)
   LogicalView(tables="customer_[0-7]", shardCount=8, sql="SELECT `c_custkey`, `c_name` FROM `custo
mer` AS `customer`")
  Gather(concurrent=true)
   LogicalView(tables="orders_[0-7]", shardCount=8, sql="SELECT `o_orderkey`, `o_custkey` FROM `orde
rs` AS `orders` WHERE ((`o_orderdate` = ?) AND (`o_totalprice` > ?))")
```

执行计划中， orders 按照 o_orderkey 拆分而 customer 按照 c_custkey 拆分，由于拆分维度不同JOIN算子不能下推。

考虑到2019-11-11当天总价高于100的订单非常多，跨分片JOIN耗时很高，需要在 orders 表上创建一个GSI来使得JOIN算子可以下推。

查询中使用到了 orders 表的 o_orderkey ， o_custkey ， o_orderdate ， o_totalprice 四列，其中 o_orderkey ， o_custkey 分别是主表和索引表的拆分键， o_orderdate ， o_totalprice 作为覆盖列包含在索引中用于避免回表。

```
> create global index i_o_custkey on orders(`o_custkey`) covering(`o_orderdate`, `o_totalprice`)
    DBPARTITION BY HASH(`o_custkey`) TBPARTITION BY HASH(`o_custkey`) TBPARTITIONS 4;
```

增加GSI并通过 force index(i_o_custkey) 强制使用索引后，跨分片JOIN变为MySQL上的局部JOIN（ IndexScan 中），并且通过覆盖列避免了回表操作，查询性能得到提升。

```
> EXPLAIN select o_orderkey, c_custkey, c_name from orders force index(i_o_custkey), customer
      where o_custkey = c_custkey and o_orderdate = '2019-11-11' and o_totalprice > 100;
Gather(concurrent=true)
 IndexScan(tables="i_o_custkey_[0-7],customer_[0-7]", shardCount=8, sql="SELECT `i_o_custkey`.`o_ord
erkey`, `customer`.`c_custkey`, `customer`.`c_name` FROM `i_o_custkey` AS `i_o_custkey` INNER JOI
N `customer` AS `customer` ON (((`i_o_custkey`.`o_orderdate` = ?) AND (`i_o_custkey`.`o_custkey` =
`customer`.`c_custkey`)) AND (`i_o_custkey`.`o_totalprice` > ?))")
```

更多关于全局二级索引的使用细节，请参见使用全局二级索引。

执行计划调优

> **说明** 以下内容适用于PolarDB-X5.3.12或以上版本。

大多数情况下，PolarDB-X的查询优化器可以自动产生最佳的执行计划。但是，少数情况下，可能因为统计信息存在缺失、误差等，导致生成的执行计划不够好，这时，可以通过Hint来干预优化器行为，使之生成更好的执行计划。

如下示例将讲解执行计划的调优。

下面的查询，PolarDB-X查询优化器综合了JOIN两边的代价。

```
> EXPLAIN select o_orderkey, c_custkey, c_name from orders, customer
      where o_custkey = c_custkey and o_orderdate = '2019-11-15' and o_totalprice < 10;
Project(o_orderkey="o_orderkey", c_custkey="c_custkey", c_name="c_name")
 HashJoin(condition="o_custkey = c_custkey", type="inner")
  Gather(concurrent=true)
   LogicalView(tables="customer_[0-7]", shardCount=8, sql="SELECT `c_custkey`, `c_name` FROM `custo
mer` AS `customer`")
  Gather(concurrent=true)
   LogicalView(tables="orders_[0-7]", shardCount=8, sql="SELECT `o_orderkey`, `o_custkey` FROM `orde
rs` AS `orders` WHERE ((`o_orderdate` = ?) AND (`o_totalprice` < ?))")
```

但是，实际上2019-11-15这一天总价低于10元的订单数量很小，只有几条，这时候用BKAJOIN是比Hash JOIN更好的选择（关于BKAJOIN和Hash JOIN的介绍，请参见JOIN与子查询的优化和执行。

通过如下 `/*+TDDL:BKA_JOIN(orders, customer)*/` Hint强制优化器使用BKAJOIN（LookupJOIN）：

```
> EXPLAIN /*+TDDL:BKA_JOIN(orders, customer)*/ select o_orderkey, c_custkey, c_name from orders, custo
mer
      where o_custkey = c_custkey and o_orderdate = '2019-11-15' and o_totalprice < 10;
Project(o_orderkey="o_orderkey", c_custkey="c_custkey", c_name="c_name")
 BKAJoin(condition="o_custkey = c_custkey", type="inner")
  Gather(concurrent=true)
   LogicalView(tables="orders_[0-7]", shardCount=8, sql="SELECT `o_orderkey`, `o_custkey` FROM `orde
rs` AS `orders` WHERE ((`o_orderdate` = ?) AND (`o_totalprice` < ?))")
  Gather(concurrent=true)
   LogicalView(tables="customer_[0-7]", shardCount=8, sql="SELECT `c_custkey`, `c_name` FROM `custo
mer` AS `customer` WHERE (`c_custkey` IN ('?'))")
```

可以选择执行加如下Hint的查询：

```
/*+TDDL:BKA_JOIN(orders, customer)*/ select o_orderkey, c_custkey, c_name from orders, customer where
o_custkey = c_custkey and o_orderdate = '2019-11-15' and o_totalprice < 10;
```

以上操作加快了SQL查询速度。为了让Hint发挥作用，可以将应用中的SQL加上Hint，或者更方便的方式是使用执行计划管理（Plan Management）功能对该SQL固定执行计划。具体操作如下：

```
BASELINE FIX SQL /*+TDDL:BKA_JOIN(orders, customer)*/ select o_orderkey, c_custkey, c_name from order
s, customer where o_custkey = c_custkey and o_orderdate = '2019-11-15';
```

这样一来，对于这条SQL（参数可以不同），PolarDB-X都会采用如上固定的执行计划。

更多关于执行计划管理的信息，请参见执行计划管理。

18.4. SQL调优进阶

18.4.1. 查询优化器介绍

查询优化器通过优化逻辑计划从而输出物理计划，其主要阶段包含查询改写和计划枚举。

PolarDB-X接收到一条SQL后的执行过程大致如下：

1. 语法解析器（Parser）将SQL文本解析成抽象语法树（AST）。
2. 语法树被转化成基于关系代数的逻辑计划。
3. 优化器（Optimizer）对逻辑计划进行优化得到物理计划。
4. 执行器（Executor）执行该计划，得到查询结果并返回给客户端。

本文将会介绍查询优化器的基本原理，包含如下几个方面：

- 关系代数算子。
- 查询改写（RBO阶段）。
- 查询计划枚举（CBO阶段）。

关系代数算子

一条SQL查询在数据库系统中通常被表示为一棵关系代数算子组成的树，有如下场景的算子：

- `Project`：用于描述SQL中的SELECT列，包括函数计算。
- `Filter`：用于描述SQL中的WHERE条件。
- JOIN：用于描述SQL中的JOIN，其对应的物理算子有HashJoin、 BKAJoin、Nested-Loop Join、SortMergeJoin。
- Agg：用于描述SQL中的Group By及聚合函数，其对应的物理算子有HashAgg、SortAgg。
- Sort：用于描述SQL中的Order By及Limit，其对应的物理算子有TopN、MemSort。
- `LogicalView`：用于描述PolarDB-X下发至存储层MySQL的SQL，其内部可能包含一个或多个逻辑算子。
- `Gather`：代表从多个数据流汇集数据的操作，通常出现在LogicalView之上（若开启并行执行，则并行优化步骤会将其上拉）。

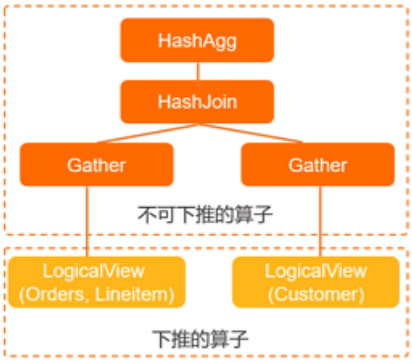
例如，对于如下查询SQL（修改自TPC-H Query 3）：

```
SELECT l_orderkey, sum(l_extendedprice *(1 - l_discount)) AS revenue
FROM CUSTOMER, ORDERS, LINEITEM
WHERE c_mktsegment = 'AUTOMOBILE'
 and c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and o_orderdate < '1995-03-13'
 and l_shipdate > '1995-03-13'
GROUP BY l_orderkey;
```

通过如下EXPLAIN命令看到PolarDB-X的执行计划：

```
HashAgg(group="l_orderkey", revenue="SUM(*)")
 HashJoin(condition="o_custkey = c_custkey", type="inner")
  Gather(concurrent=true)
   LogicalView(tables="ORDERS_[0-7],LINEITEM_[0-7]", shardCount=8, sql="SELECT `ORDERS`.`o_custkey
`, `LINEITEM`.`l_orderkey`, (`LINEITEM`.`l_extendedprice` * (? - `LINEITEM`.`l_discount`)) AS `x` FROM
`ORDERS` AS `ORDERS` INNER JOIN `LINEITEM` AS `LINEITEM` ON (((`ORDERS`.`o_orderkey` = `LINEITE
M`.`l_orderkey`) AND (`ORDERS`.`o_orderdate` < ?)) AND (`LINEITEM`.`l_shipdate` > ?))")
  Gather(concurrent=true)
   LogicalView(tables="CUSTOMER_[0-7]", shardCount=8, sql="SELECT `c_custkey` FROM `CUSTOMER` AS
`CUSTOMER` WHERE (`c_mktsegment` = ?)")
```

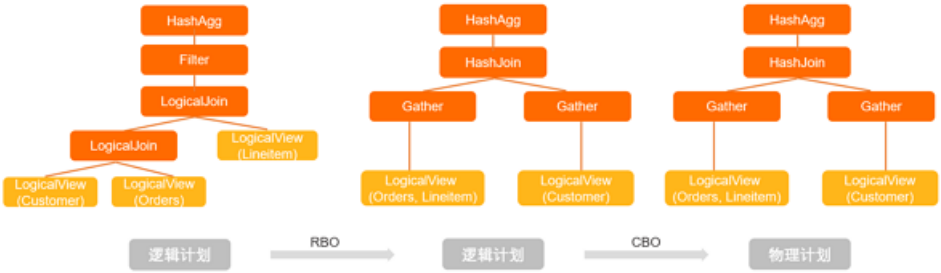
用树状图表示如下：



> **说明** 左边的LogicalView实际包含了ORDERS和LINEITEM两张表的JOIN。EXPLAIN结果中LogicalView的SQL属性也体现了这一点。

查询改写（RBO）

查询改写（SQL Rewrite）阶段输入为逻辑执行计划，输出为逻辑执行计划。这一步主要应用一些启发式规则，是基于规则的优化器（Rule-Based Optimizer，简称RBO），所以也常被称为RBO阶段。



查询改写这一步的主要有如下功能：

- 子查询去关联化（Subquery Unnesting）

  子查询去关联化是将含有关联项的子查询（关联子查询）表示为SemiJoin或类似的算子，便于后续的各种优化，例如下推到存储层MySQL或在PolarDB-X层选择某种算法执行。在如下例子中IN子查询转化为SemiJoin算子，并最终转化成SemiHashJoin物理算子由PolarDB-X进行执行：

```
> explain  select id from t1 where id in (select id from t2 where t2.name = 'hello');
SemiHashJoin(condition="id = id", type="semi")
 Gather(concurrent=true)
  LogicalView(tables="t1", shardCount=2, sql="SELECT `id` FROM `t1` AS `t1`")
 Gather(concurrent=true)
  LogicalView(tables="t2_[0-3]", shardCount=4, sql="SELECT `id` FROM `t2` AS `t2` WHERE (`name` = ?
)")
```

- 算子下推

  算子下推是非常关键的一步，PolarDB-X内置了如下算子的下推优化规则：

优化规则	描述
谓词下推或列裁剪	将Filter及Project算子下推至存储层MySQL执行，过滤掉不需要的行和列。
JOIN Clustering	将JOIN按照拆分方式及拆分键的等值条件进行重排和聚簇，方便下一步的JOIN下推。
JOIN下推	对于符合条件的JOIN，将其下推至存储层MySQL执行。
Agg下推	将聚合（Agg）拆分为FinalAgg和LocalAgg两个阶段，并将LocalAgg下推至存储层MySQL。
Sort下推	将排序（Sort）拆分为MergeSort和LocalSort两个阶段，并将LocalSort下推至存储层MySQL。

  更多关于查询下推的信息，请参见查询改写与下推。

查询计划枚举（CBO）

查询改写阶段输出的逻辑执行计划会被输入到查询计划枚举（Plan Enumerator）中，并输出一个最终的物理执行计划。查询计划枚举在多个可行的查询计划中，根据预先定义的代价模型，选择出代价最低的一个。与查询改写阶段不同，在查询计划枚举中，规则可能产生更好的执行计划，也可能产生更差的执行计划，可以根据算子经过规则优化后的前后代价对比选出较优的那个，因此这也被称为基于代价的优化（Cost-based Optimizer，简称CBO）。

其核心组件有以下几个部分：

- 统计信息（Statistics）
- 基数估计（Cardinality Estimation）
- 转化规则（Transform Rules）
- 代价模型（Cost Model）
- 计划空间搜索引擎（Plan Space Search Engine）

逻辑上，CBO的过程包括如下几个步骤：

1. 搜索引擎利用转化规则，对输入的逻辑执行计划进行变换，构造出物理执行计划的搜索空间。
2. 之后，利用代价模型对搜索空间中的每一个执行计划进行代价估计，选出代价最低的物理执行计划。
3. 而代价估计的过程离不开基数估计，它利用各个表、列的统计信息，估算出各算子的输入行数、选择率等信息，提供给算子的代价模型，从而估算出查询计划的代价。

18.4.2. 查询改写与下推

下推是查询改写的一项重要优化，利用PolarDB-X的拆分信息来优化执行计划，使得算子尽量下推以达到提前过滤数据、减少网络传输、并行计算等目的。

背景信息

根据PolarDB-X的SQL语句优化的基本原则，可以下推尽量更多的计算到存储层MySQL上执行。

可下推计算主要包括：

- JOIN连接
- 过滤条件（如 WHERE 或 HAVING 中的条件）
- 计算（如 COUNT 、 GROUP BY ）
- 排序（如 ORDER BY ）
- 去重（如 DISTINCT ）
- 函数计算（如 NOW() 函数）
- 子查询

> **说明** 通过 explain optimizer + sql 可以看到查询改写的具体过程。

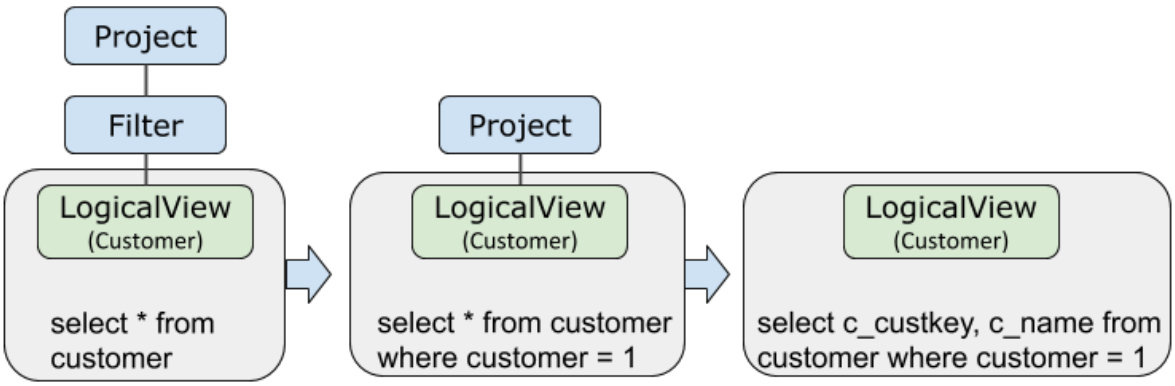
Project和Filter下推

一条SQL的执行计划在如下生成过程中，Filter和Project被先后下推到 LogicalView 算子里面。

Filter和Project下推可以达到提前过滤数据，减少网络传输等效果。

```
> explain optimizer select c_custkey,c_name from customer where c_custkey = 1;
```

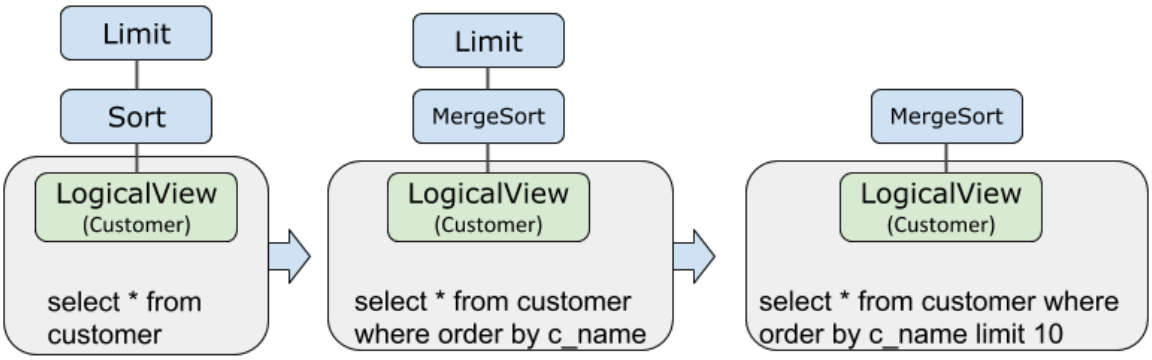
c_custkey ： customer 的拆分键。 c_name ： customer 的名字。



Limit和Sort下推

一条SQL的执行计划在如下生成过程中，Sort和Limit被先后下推到 LogicalView 算子里面。Sort和Limit下推可以达到提前过滤数据，减少网络传输、并行执行、减少PolarDB-X内存占用等效果。

```
> explain optimizer select * from customer order by c_custkey limit 10
```



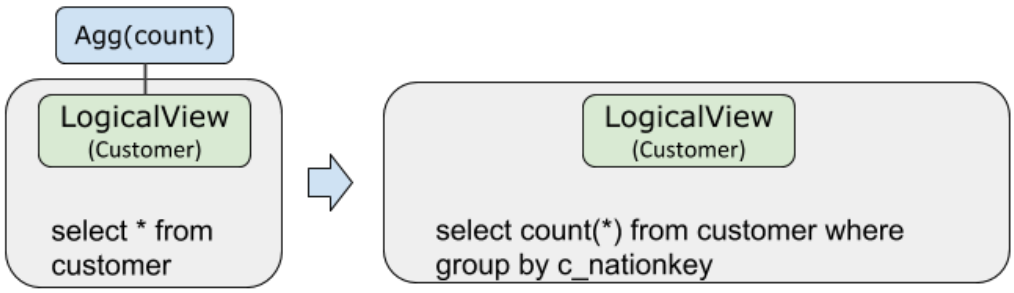
Agg下推

一条SQL的执行计划在如下生成过程中，Agg被下推到 `LogicalView` 算子里面。

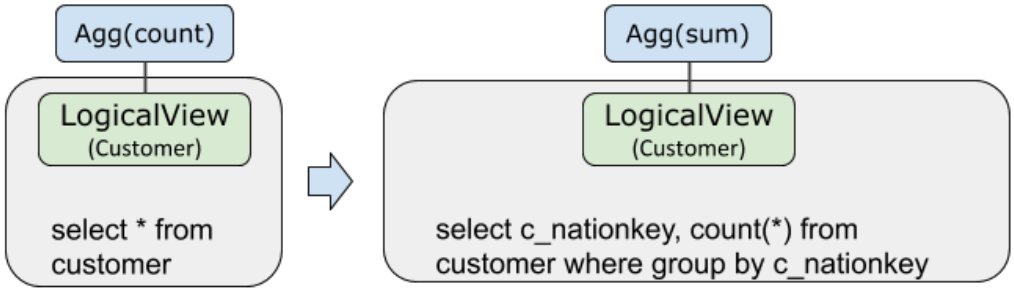
Agg下推可以达到提前过滤数据，减少网络传输，并行执行，减少PolarDB-X内存占用等效果。

```
> explain optimizer select count(*) from customer group by c_nationkey;
```

拆分键为 `c_nationkey` 情况：



拆分键不为 `c_nationkey` 情况：



JOIN下推

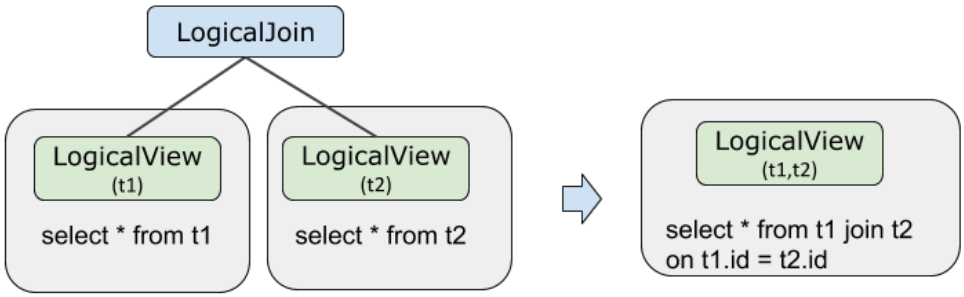
JOIN下推需要满足以下条件：

- t1与t2表的拆分方式一致（包括分库键、分表键、拆分函数、分库分表数目）。
- JOIN条件中包含t1，t2表拆分键的等值关系。

此外，任意表JOIN广播表总是可以下推。

```
> explain optimizer select * from t1, t2 where t1.id = t2.id;
```

一条SQL的执行计划在如下生成过程中，JOIN下推到 LogicalView 算子里面。JOIN下推可以达到计算离存储更近，并行执行加速的效果。



JoinClustering

当有多个表执行JOIN操作时，PolarDB-X会通过join clustering的优化技术将JOIN进行重排序，将可下推的JOIN放到相邻的位置，从而让它可以被正常下推。示例如下：

假设原JOIN顺序为t2、t1、l2, 经过重排序之后，t2和l2的JOIN操作依然能下推到 LogicalView 。

```
> explain select t2.id from t2 join t1 on t2.id = t1.id join l2 on t1.id = l2.id;
Project(id="id")
 HashJoin(condition="id = id AND id = id0", type="inner")
  Gather(concurrent=true)
   LogicalView(tables="t2_[0-3],l2_[0-3]", shardCount=4, sql="SELECT `t2`.`id`, `l2`.`id` AS `id0` FROM `t2` AS `t2` INNER JOIN `l2` AS `l2` ON (`t2`.`id` = `l2`.`id`) WHERE (`t2`.`id` = `l2`.`id`)")
  Gather(concurrent=true)
   LogicalView(tables="t1", shardCount=2, sql="SELECT `id` FROM `t1` AS `t1`")
```

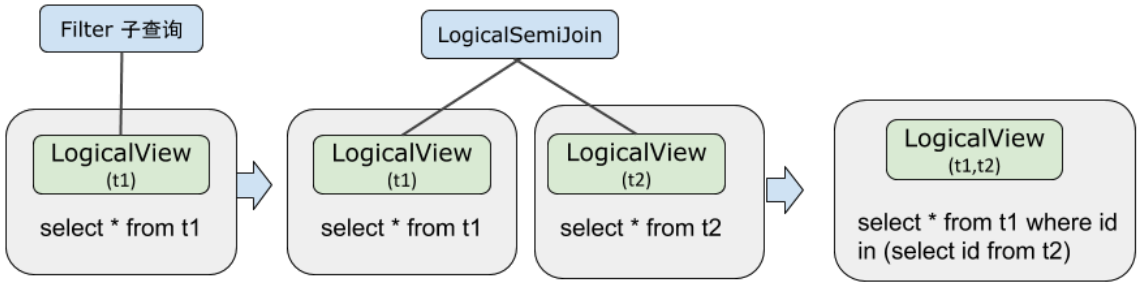
子查询下推

一条SQL的执行计划在如下生成过程中，子查询下推到 LogicalView 算子里面。

子查询下推可以达到计算离存储更近，并行执行加速的效果。

- 子查询会先被转换成 Semi Join 或 Anti Join 。
- 之后如果满足上节中JOIN下推的判断条件，就会将 Semi Join 或 Anti Join 下推至 LogicalView 。
- 下推后的 Semi Join 或 Anti Join 会被还原为子查询。

```
explain optimizer select * from t1 where id in (select id from t2);
```



18.4.3. 查询执行器介绍

本文介绍PolarDB-X的SQL执行器如何执行 SQL中无法下推的部分。

基本概念

SQL执行器是PolarDB-X中执行逻辑层算子的组件。对于简单的点查SQL，往往可以整体下推存储层MySQL执行，因而感觉不到执行器的存在，MySQL的结果经过简单的解包封包又被回传给用户。但是对于较复杂的SQL，往往无法将SQL中的算子全部下推，这时候就需要PolarDB-X执行器执行无法下推的计算。

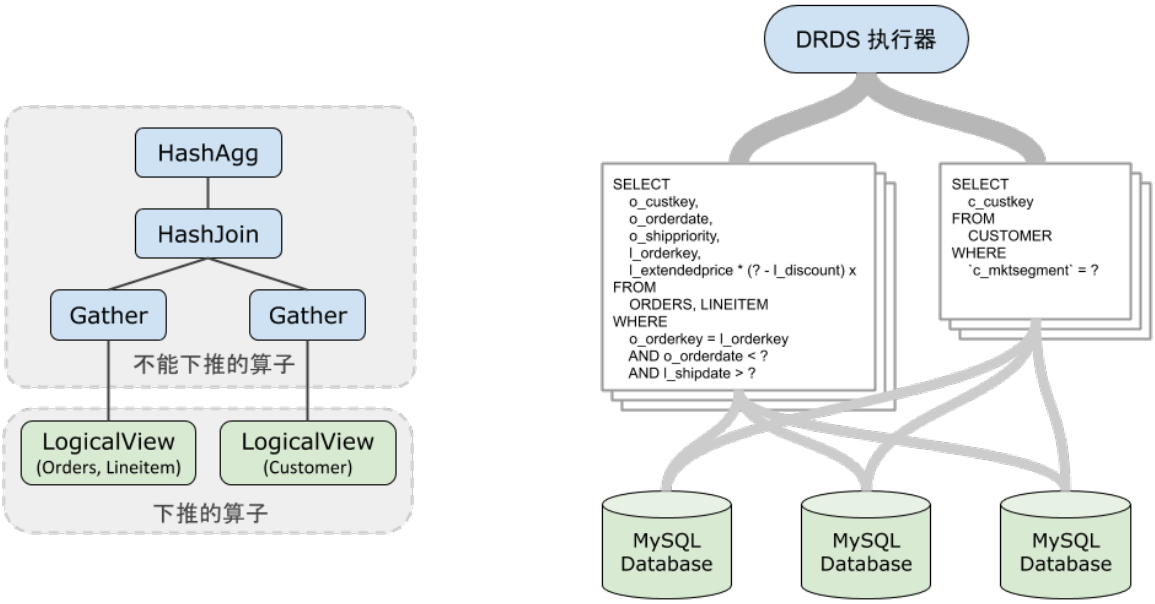
例如，对于如下查询SQL:

```
SELECT l_orderkey, sum(l_extendedprice *(1 - l_discount)) AS revenue
FROM CUSTOMER, ORDERS, LINEITEM
WHERE c_mktsegment = 'AUTOMOBILE'
 and c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and o_orderdate < '1995-03-13'
 and l_shipdate > '1995-03-13'
GROUP BY l_orderkey;
```

通过 `EXPLAIN` 命令看到PolarDB-X的执行计划如下:

```
HashAgg(group="l_orderkey", revenue="SUM(*)")
 HashJoin(condition="o_custkey = c_custkey", type="inner")
  Gather(concurrent=true)
   LogicalView(tables="ORDERS_[0-7],LINEITEM_[0-7]", shardCount=8, sql="SELECT `ORDERS`.`o_custkey
`, `LINEITEM`.`l_orderkey`, (`LINEITEM`.`l_extendedprice` * (? - `LINEITEM`.`l_discount`)) AS `x` FROM
`ORDERS` AS `ORDERS` INNER JOIN `LINEITEM` AS `LINEITEM` ON (((`ORDERS`.`o_orderkey` = `LINEITE
M`.`l_orderkey`) AND (`ORDERS`.`o_orderdate` < ?)) AND (`LINEITEM`.`l_shipdate` > ?))")
  Gather(concurrent=true)
   LogicalView(tables="CUSTOMER_[0-7]", shardCount=8, sql="SELECT `c_custkey` FROM `CUSTOMER` AS
`CUSTOMER` WHERE (`c_mktsegment` = ?)")
```

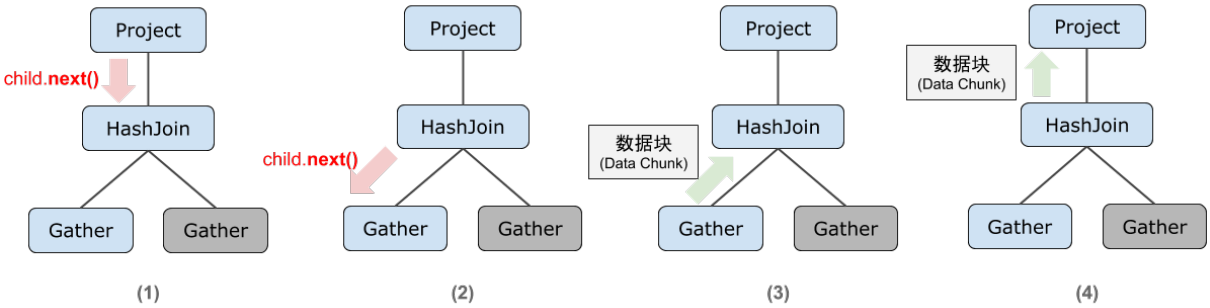
如下图所示， `LogicalView` 的SQL在执行时被下发给MySQL，而不能下推的部分（除 `LogicalView` 以外的算子）由PolarDB-X执行器进行计算，得到最终用户SQL需要的结果。



Volcano执行模型

PolarDB-X和很多数据库一样采用Volcano执行模型。所有算子都定义了 `open()` 、 `next()` 等接口，算子根据执行计划组合成一棵算子树，上层算子通过调用下层算子的 `next()` 接口的取出结果，完成该算子的计算。最终顶层算子产生用户需要的结果并返回给客户端。

下面的例子中，假设 `HashJoin` 算子已经完成构建哈希表。当上层的Project算子请求数据时， `HashJoin` 首先向下层 `Gather` 请求一批数据，然后查表得到JOIN结果，再返回给Project算子。



某些情况下，算子需要将数据全部读取并缓存在内存中，该过程被称为物化，例如，HashJoin算子需要读取内表的全部数据，并在内存中构建出哈希表。其他类似的算子还有HashAgg（聚合）、MemSort（排序）等。

由于内存资源是有限的，如果物化的数据量超出单条查询限制，或者使用的总内存超出PolarDB-X节点内存限制，将会引起内存不足（ `OUT_OF_MEMORY` ）报错。

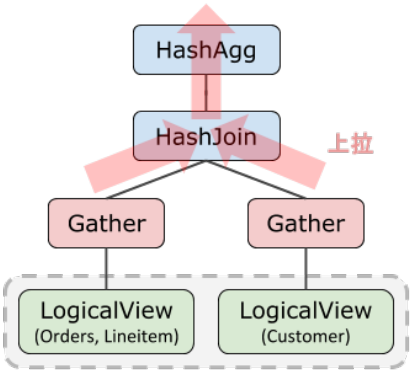
并行查询

并行查询（Parallel Query）指利用多线程并行执行用户的复杂查询。

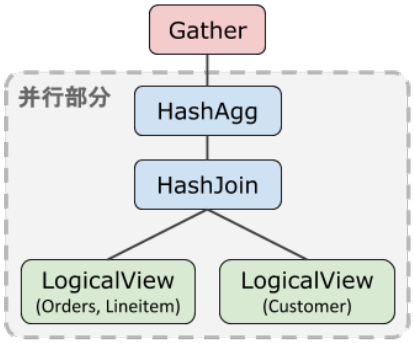
> **说明** 该功能仅在PolarDB-X标准版及企业版上提供，入门版由于硬件规格限制，不提供该项功能。

并行查询的执行计划相比原来有所改动。例如，还是以上面的查询为例，它的并行执行计划如下所示：

```
Gather(parallel=true)
 ParallelHashAgg(group="o_orderdate,o_shippriority,l_orderkey", revenue="SUM(*)")
  ParallelHashJoin(condition="o_custkey = c_custkey", type="inner")
   LogicalView(tables="ORDERS_[0-7],LINEITEM_[0-7]", shardCount=8, sql="SELECT `ORDERS`.`o_custkey`, `ORDERS`.`o_orderdate`, `ORDERS`.`o_shippriority`, `LINEITEM`.`l_orderkey`, (`LINEITEM`.`l_extendedprice` * (? - `LINEITEM`.`l_discount`)) AS `x` FROM `ORDERS` AS `ORDERS` INNER JOIN `LINEITEM` AS `LINEITEM` ON (((`ORDERS`.`o_orderkey` = `LINEITEM`.`l_orderkey`) AND (`ORDERS`.`o_orderdate` < ?)) AND (`LINEITEM`.`l_shipdate` > ?))", parallel=true)
   LogicalView(tables="CUSTOMER_[0-7]", shardCount=8, sql="SELECT `c_custkey` FROM `CUSTOMER` AS `CUSTOMER` WHERE (`c_mktsegment` = ?)", parallel=true)
```



1) 非并行执行计划



2) 并行执行计划

可以看出，并行执行计划中Gather算子的位置被拉高了，这也意味者Gather下方的算子都会以并行方式执行，直到Gather时才被汇总成在一起。

执行时，Gather下方的算子会实例化出多个执行实例，分别对应一个并行度。并行度默认等于单台机器的核心数，标准版实例默认并行度为8，企业版实例默认并行度为16。

执行过程的诊断分析

除了上文提到的EXPLAIN指令，还有如下几个指令能帮助分析性能问题：

- `EXPLAIN ANALYZE` 指令用于分析PolarDB-X Server中各算子执行的性能指标。
- `EXPLAIN EXECUTE` 指令用于输出MySQL的EXPLAIN结果（并汇总输出）。

如下是以上文提到的查询为例，介绍如何分析一条查询的性能问题。

执行 `EXPLAIN ANALYZE` 得到如下结果（删除了一些无关的信息）：

```
explain analyze select l_orderkey, sum(l_extendedprice *(1 - l_discount)) as revenue from CUSTOMER, ORDE
RS, LINEITEM where c_mktsegment = 'AUTOMOBILE' and c_custkey = o_custkey and l_orderkey = o_orderke
y and o_orderdate < '1995-03-13' and l_shipdate > '1995-03-13' group by l_orderkey;
HashAgg(group="o_orderdate,o_shippriority,l_orderkey", revenue="SUM(*)")
... actual time = 23.916 + 0.000, actual rowcount = 11479, actual memory = 1048576, instances = 1 ...
 HashJoin(condition="o_custkey = c_custkey", type="inner")
 ... actual time = 0.290 + 23.584, actual rowcount = 30266, actual memory = 1048576, instances = 1 ...
  Gather(concurrent=true)
  ... actual time = 0.000 + 23.556, actual rowcount = 151186, actual memory = 0, instances = 1 ...
   LogicalView(tables="ORDERS_[0-7],LINEITEM_[0-7]", shardCount=8, sql="SELECT `ORDERS`.`o_custkey
`, `ORDERS`.`o_orderdate`, `ORDERS`.`o_shippriority`, `LINEITEM`.`l_orderkey`, (`LINEITEM`.`l_exte
ndedprice` * (? - `LINEITEM`.`l_discount`)) AS `x` FROM `ORDERS` AS `ORDERS` INNER JOIN `LINEITEM`
AS `LINEITEM` ON (((`ORDERS`.`o_orderkey` = `LINEITEM`.`l_orderkey`) AND (`ORDERS`.`o_orderdate
` < ?)) AND (`LINEITEM`.`l_shipdate` > ?))")
   ... actual time = 0.000 + 23.556, actual rowcount = 151186, actual memory = 0, instances = 4 ...
  Gather(concurrent=true)
  ... actual time = 0.000 + 0.282, actual rowcount = 29752, actual memory = 0, instances = 1 ...
   LogicalView(tables="CUSTOMER_[0-7]", shardCount=8, sql="SELECT `c_custkey` FROM `CUSTOMER` AS
`CUSTOMER` WHERE (`c_mktsegment` = ?)")
   ... actual time = 0.000 + 0.282, actual rowcount = 29752, actual memory = 0, instances = 4 ...
```

其中：

- actual time表示实际执行耗时（其中包含子算子的耗时），加号（+）左边表示open（准备数据）耗时，右边表示next（输出数据）耗时。
- actual rowcount表示输出的行数。
- actual memory表示算子使用的内存空间大小（单位为Bytes）。
- instances表示实例数，非并行查询时始终为1，对于并行算子每个并行度对应一个实例。如果实例数不等于1，actual time，actual rowcount，actual memory代表多个实例并行执行的总实际执行耗时、总输出行数、总内存使用量。

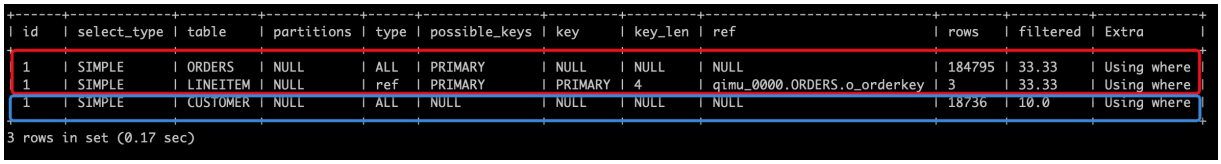
> **说明** 当使用并行查询时，上述的算子耗时、输出行数等信息均为算子多个实例的累加。例如 `actual time = 20，instances = 8` ，表示该算子有8个实例并行执行，平均耗时为2.5s。

以上面的输出为例，解读如下：

- HashAgg算子open耗时为23.916s，用于获取下层HashJoin的输出、并对输出的所有数据做分组和聚合。其中的23.601s都用在了获取了下层输出上，只有约0.3s用于分组聚合。
- HashJoin算子open耗时0.290s，用于拉取右表（下方的Gather）数据并构建哈希表；next耗时23.584s，用于拉取左表数据以及查询哈希表得到JOIN结果。
- Gather算子仅仅用于汇总多个结果集，通常代价很低。
- 左侧（上方）的LogicalView拉取数据消耗了23.556s，可判断这里是查询的性能瓶颈。
- 右侧（下方）的LogicalView拉取数据消耗了0.282s。

综上，性能瓶颈在左边的LogicalView上。从执行计划中可以看到，它是对ORDERS、LINEITEM的JOIN查询，这条查询MySQL执行速度较慢。

您可以通过如下 `EXPLAIN EXECUTE` 语句查看MySQL EXPLAIN结果：



```
+------+-------------+----------+------------+------+---------------+---------+---------+-------------------------------+--------+----------+-------------+
| id   | select_type | table    | partitions | type | possible_keys | key     | key_len | ref                           | rows   | filtered | Extra       |
+------+-------------+----------+------------+------+---------------+---------+---------+-------------------------------+--------+----------+-------------+
| 1    | SIMPLE      | ORDERS   | NULL       | ALL  | PRIMARY       | NULL    | NULL    | NULL                          | 184795 | 33.33    | Using where |
| 1    | SIMPLE      | LINEITEM | NULL       | ref  | PRIMARY       | PRIMARY | 4       | qimu_0000.ORDERS.o_orderkey | 3      | 33.33    | Using where |
| 1    | SIMPLE      | CUSTOMER | NULL       | ALL  | NULL          | NULL    | NULL    | NULL                          | 18736  | 10.0     | Using where |
+------+-------------+----------+------------+------+---------------+---------+---------+-------------------------------+--------+----------+-------------+
3 rows in set (0.17 sec)
```

上图中，红色方框对应左边的LogicalView的下推查询，蓝色方框对应右边LogicalView的下推查询。

18.4.4. 执行计划和基本算子

本文介绍如何使用 EXPLAIN 命令查询执行计划，并介绍一些基本的算子（例如 LogicalView ，Gather ，MergeSort 等）。更多算子（例如Join、Agg、Sort等）的介绍在后续的章节中单独列出。

背景信息

通常SQL调优的过程离不开以下两个步骤：

1. 分析问题，例如通过 EXPLAIN 命令查看执行计划，您也可以通过 EXPLAIN ANALYZE 查看实际执行情况来分析问题。（参见查询执行器介绍章节）。
2. 通过 Hint 控制优化器行为，修改执行计划。

执行计划与 EXPLAIN 命令

下述案例介绍如何通过 EXPLAIN 命令获取查询的执行计划。

EXPLAIN 语法如下：

> 说明 Hint 需放在 EXPLAIN 之后。

```
EXPLAIN <SQL Statement>
EXPLAIN <Hint> <SQL Statement>
```

本文中的示例均基于以下表结构：

```
CREATE TABLE `sbtest1` (
 `id` INT(10) UNSIGNED NOT NULL,
 `k` INT(10) UNSIGNED NOT NULL DEFAULT '0',
 `c` CHAR(120) NOT NULL DEFAULT '',
 `pad` CHAR(60) NOT NULL DEFAULT '',
 KEY `xid` (`id`),
 KEY `k_1` (`k`)
) dbpartition BY HASH (`id`) tbpartition BY HASH (`id`) tbpartitions 4
```

执行以下 EXPLAIN 命令，PolarDB-X将返回对应的执行计划。

```
mysql> explain select a.k, count(*) cnt from sbtest1 a, sbtest1 b where a.id = b.k and a.id > 1000 group by k h
aving cnt > 1300 order by cnt limit 5, 10;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
| LOGICAL PLAN                                                                                                                                                               |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
| MemSort(sort="cnt ASC", offset=?2, fetch=?3)                                                                                                                               |
|  Filter(condition="cnt > ?1")                                                                                                                                              |
|    HashAgg(group="k", cnt="COUNT()")                                                                                                                                       |
|     BKAJoin(id="id", k="k", c="c", pad="pad", id0="id0", k0="k0", c0="c0", pad0="pad0", condition="id = k",
type="inner")                |
|       MergeSort(sort="k ASC")                                                                                                                                              |
|        LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM `sbtest1`
WHERE (`id` > ?) ORDER BY `k`")      |
|       Gather(concurrent=true)                                                                                                                                              |
|        LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM `sbtest1`
WHERE ((`k` > ?) AND (`k` IN ('?')))") |
| HitCache:false                                                                                                                                                             |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
9 rows in set (0.01 sec)
```

除执行计划外，`EXPLAIN` 结果中还会有一些额外信息，上面的例子中仅有一项 `HitCache`（是否命中Plan Cache缓存），详细原理参见执行计划管理。

算子介绍

PolarDB-X中支持以下算子。

含义	物理算子
下发查询	`LogicalView`，`LogicalModifyView`，`PhyTableOperation`
连接（Join）	`BKAJoin`，`NLJoin`，`HashJoin`，`SortMergeJoin`，`HashSemiJoin`，`SortMergeSemiJoin`，`MaterializedSemiJoin`
排序	`MemSort`，`TopN`
聚合（Group By）	`HashAgg`，`SortAgg`
数据交换	`Gather`，`MergeSort`
其它	`Project`，`Filter`，`Limit`，`Union`，`Window`

以下介绍部分算子的含义和实现，剩余的部分在后面的章节中介绍。

LogicalView

`LogicalView` 是从存储层MySQL数据源拉取数据的算子，类似于其他数据库中的TableScan或IndexScan，但支持更多的下推。`LogicalView` 中包含下推的SQL语句和数据源信息，更像一个视图。其中下推的SQL可能包含多种算子，如Project、Filter、聚合、排序、Join和子查询等。下述示例为您展示 `EXPLAIN` 中 `LogicalView` 的输出信息及其含义：

```
> explain select * From sbtest1 where id > 1000;
Gather(concurrent=true)
  LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM `sbtest1` WH
ERE (`id` > ?)")
```

LogicalView 的信息由三部分构成：

- tables ：存储层MySQL对应的表名，以英文句号（.）分割，英文句号（.）之前是分库对应的编号，之后是表名及其编号，如 [000-127] 表示表名编号从000到127的所有表。
- shardCount ：需访问的分表总数，该示例会访问从000到127共计128张分表。
- sql ：下发至存储层MySQL的SQL模版，PolarDB-X在执行时会将表名替换为物理表名，参数化的常量问号（?）替换成实际参数，详情请参见执行计划管理。

Gather

Gather 将多份数据合并成同份数据。上面的例子中， Gather 将各个分表上查询到的数据合并成一份。

在不使用并行查询时， Gather 通常出现在 LogicalView 上方，表示收集合并各个分表的数据。如果并行查询开启， Gather 可能被上拉到执行计划中更高的地方，这时候 Gather 表示将各个Worker的计算结果收集合并。

Gather 中的 concurrent 表示是否并发执行子算子，默认为 true ，表示并发拉取数据。开启并行查询时，上拉的 Gather 属性有所变化，显示为 parallel=true 。

MergeSort

MergeSort 即归并排序算子，表示将有序的数据流进行归并排序，合并成一个有序的数据流。例如：

```
> explain select * from sbtest1 where id > 1000 order by id limit 5,10;
MergeSort(sort="id ASC", offset=?1, fetch=?2)
  LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM `sbtest1` WHE
RE (`id` > ?) ORDER BY `id` LIMIT (? + ?)")
```

MergeSort 算子包含三部分内容：

- sort ：表示排序字段以及排列顺序， id ASC 表示按照ID字段递增排序， DESC 表示递减排序。
- offset ：表示获取结果集时的偏移量，例子中被参数化了，实际值为5。
- fetch ：表示最多返回的数据行数。与offset类似，同样是参数化的表示，实际对应的值为10。

Project

Project 表示投影操作，即从输入数据中选择部分列输出，或者对某些列进行转换（通过函数或者表达式计算）后输出，当然也可以包含常量。

```
> explain select '你好, DRDS', 1 / 2, CURTIME();
Project(你好, DRDS="_UTF-16'你好, DRDS'", 1 / 2="1 / 2", CURTIME()="CURTIME()")
```

Project 的计划中包括每列的列名及其对应的列、值、函数或者表达式。

Filter

Filter 表示过滤操作，其中包含一些过滤条件。该算子对输入数据进行过滤，若满足条件，则输出，否则丢弃。如下是一个较复杂的例子，包含了以上介绍的大部分算子。

```
> explain select k, avg(id) avg_id from sbtest1 where id > 1000 group by k having avg_id > 1300;
Filter(condition="avg_id > ?1")
  Project(k="k", avg_id="sum_pushed_sum / sum_pushed_count")
    SortAgg(group="k", sum_pushed_sum="SUM(pushed_sum)", sum_pushed_count="SUM(pushed_count)"
)
      MergeSort(sort="k ASC")
        LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT `k`, SUM(`id`) AS `
pushed_sum`, COUNT(`id`) AS `pushed_count` FROM `sbtest1` WHERE (`id` > ?) GROUP BY `k` ORDER B
Y `k`")
```

`WHERE id > 1000` 中的条件没有对应的Filter算子，是因为这个算子最终被下推到了 `LogicalView` 中，可以在 `LogicalView` 的SQL中看到 `WHERE (id > ?)` 。

Union All与Union Distinct

顾名思义，Union All对应 `UNIONALL` ，Union Distinct对应 `UNIONDISTINCT` 。该算子通常有2个或更多个输入，表示将多个输入的数据合并在一起。例如：

```
> explain select * From sbtest1 where id > 1000 union distinct select * From sbtest1 where id < 200;
UnionDistinct(concurrent=true)
  Gather(concurrent=true)
    LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM `sbtest1` WH
ERE (`id` > ?)")
  Gather(concurrent=true)
    LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM `sbtest1` WH
ERE (`id` < ?)")
```

LogicalModifyView

如上文介绍， `LogicalView` 表示从底层数据源获取数据的算子，与之对应的， `LogicalModifyView` 表示对底层数据源的修改算子，其中也会记录一个SQL语句，该SQL可能是 `INSERT` 、 `UPDATE` 或者 `DELETE` 。

```
> explain update sbtest1 set c='Hello, DRDS' where id > 1000;
LogicalModifyView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="UPDATE `sbtest1` SET `
c` = ? WHERE (`id` > ?)"
```

```
> explain delete from sbtest1 where id > 1000;
LogicalModifyView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="DELETE FROM `sbtest1`
WHERE (`id` > ?)")
```

`LogicalModifyView` 查询计划的内容与 `LogicalView` 类似，包括下发的物理分表，分表数以及SQL模版。同样，由于开启了执行计划缓存，对SQL做了参数化处理，SQL模版中的常量会用 `?` 替换。

PhyTableOperation

`PhyTableOperation` 表示对某个物理分表直接执行一个操作。

> **说明** 通常情况下，该算子仅用于INSERT语句。但当路由分发分到一个分片时，该算子也会出现在SELECT语句中。

```
> explain insert into sbtest1 values(1, 1, '1', '1'),(2, 2, '2', '2');
PhyTableOperation(tables="SYSBENCH_CORONADB_1526954857179TGMMSYSBENCH_CORONADB_VGOC_0000_RDS.[sbtest1_001]", sql="INSERT INTO ? (`id`, `k`, `c`, `pad`) VALUES(?, ?, ?, ?)", params="`sbtest1_001`,1,1,1,1")
PhyTableOperation(tables="SYSBENCH_CORONADB_1526954857179TGMMSYSBENCH_CORONADB_VGOC_0000_RDS.[sbtest1_002]", sql="INSERT INTO ? (`id`, `k`, `c`, `pad`) VALUES(?, ?, ?, ?)", params="`sbtest1_002`,2,2,2,2")
```

示例中，INSERT插入两行数据，每行数据对应一个 `PhyTableOperation` 算子。 `PhyTableOperation` 算子的内容包括三部分：

- `tables` ：物理表名，仅有唯一一个物理表名。
- `sql` ：SQL模版，该SQL模版中表名和常量均被参数化，用 `?` 替换，对应的参数在随后的 `params` 中给出。
- `params` ：SQL模版对应的参数，包括表名和常量。

18.4.5. JOIN与子查询的优化和执行

本文主要介绍如何使用JOIN和子查询。JOIN将多个表以某个或某些列为条件进行连接操作而检索出关联数据的过程，多个表之间以共同列而关联在一起。子查询是指在父查询的WHERE子句或HAVING子句中嵌套另一个SELECT语句的查询。

基本概念

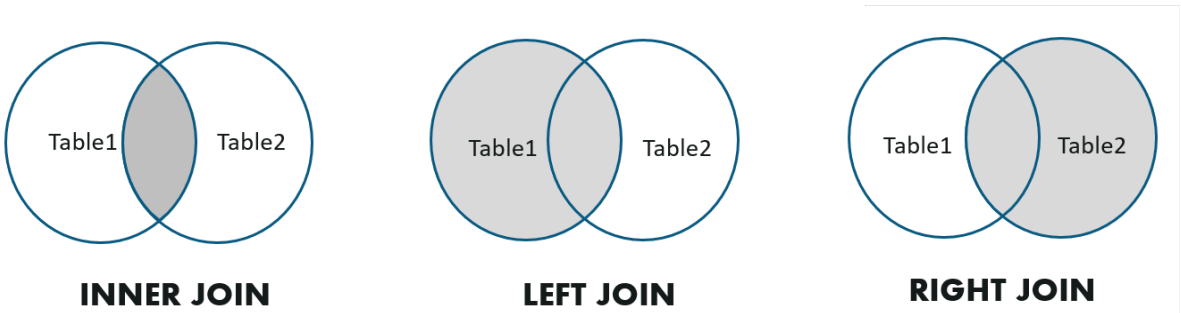
JOIN是SQL查询中常见的操作，逻辑上说，它的语义等价于将两张表做笛卡尔积，然后根据过滤条件保留满足条件的数据。JOIN多数情况下是依赖等值条件做的JOIN，即Equi-Join，用来根据某个特定列的值连接两张表的数据。

子查询是指嵌套在SQL内部的查询块，子查询的结果作为输入，填入到外层查询中，从而用于计算外层查询的结果。子查询可以出现在SQL语句的很多地方，比如在SELECT子句中作为输出的数据，在FROM子句中作为输入的一个视图，在WHERE子句中作为过滤条件等。

本文讨论的均为不下推的JOIN算子。如果JOIN被下推到LogicalView中，其执行方式由存储层MySQL自行选择。

JOIN类型

PolarDB-X支持Inner Join，Left Outer Join和Right Outer Join这3种常见的JOIN类型。



下面是几种不同类型JOIN的例子：

```
/* Inner Join */
SELECT * FROM A, B WHERE A.key = B.key;
/* Left Outer Join */
SELECT * FROM A LEFT JOIN B ON A.key = B.key;
/* Right Outer Join */
SELECT * FROM A RIGHT OUTER JOIN B ON A.key = B.key;
```

此外，PolarDB-X还支持Semi-Join和Anti-Join。Semi Join和Anti Join无法直接用SQL语句来表示，通常由包含关联项的EXISTS或IN子查询转换得到。

下面是几个Semi-Join和Anti-Join的例子：

```
/* Semi Join - 1 */
SELECT * FROM Emp WHERE Emp.DeptName IN (
  SELECT DeptName FROM Dept
)
 /* Semi Join - 2 */
SELECT * FROM Emp WHERE EXISTS (
 SELECT * FROM Dept WHERE Emp.DeptName = Dept.DeptName
)
/* Anti Join - 1 */
SELECT * FROM Emp WHERE Emp.DeptName NOT IN (
  SELECT DeptName FROM Dept
)
 /* Anti Join - 2 */
SELECT * FROM Emp WHERE NOT EXISTS (
 SELECT * FROM Dept WHERE Emp.DeptName = Dept.DeptName
)
```

JOIN算法

目前，PolarDB-X支持Nested-Loop Join、Hash Join、Sort-Merge Join和Lookup Join（BKAJoin）等JOIN算法。

Nested-Loop Join (NLJoin)

Nested-Loop Join通常用于非等值的JOIN。它的工作方式如下：

1. 拉取内表（右表，通常是数据量较小的一边）的全部数据，缓存到内存中。
2. 遍历外表数据，对于外表的每行：
   - 对于每一条缓存在内存中的内表数据。
   - 构造结果行，并检查是否满足JOIN条件，如果满足条件则输出。

如下是一个Nested-Loop Join的例子：

```
> EXPLAIN SELECT * FROM partsupp, supplier WHERE ps_suppkey < s_suppkey;
NlJoin(condition="ps_suppkey < s_suppkey", type="inner")
 Gather(concurrent=true)
  LogicalView(tables="partsupp_[0-7]", shardCount=8, sql="SELECT * FROM `partsupp` AS `partsupp`")
 Gather(concurrent=true)
  LogicalView(tables="supplier_[0-7]", shardCount=8, sql="SELECT * FROM `supplier` AS `supplier`")
```

通常来说，Nested-Loop Join是效率最低的JOIN操作，一般只有在JOIN条件不含等值（例如上面的例子）或者内表数据量极小的情况下才会使用。

通过如下Hint可以强制PolarDB-X使用Nested-Loop Join以及确定JOIN顺序：

```
/*+TDDL:NL_JOIN(outer_table, inner_table)*/ SELECT ...
```

其中inner_table 和outer_table也可以是多张表的JOIN结果，例如：

```
/*+TDDL:NL_JOIN((outer_table_a, outer_table_b), (inner_table_c, inner_table_d))*/ SELECT ...
```

下面其他的Hint也一样。

Hash Join

Hash Join是等值JOIN最常用的算法之一。它的原理如下所示：

- 拉取内表（右表，通常是数据量较小的一边）的全部数据，写进内存中的哈希表。
- 遍历外表数据，对于外表的每行：
  - 根据等值条件JOIN Key查询哈希表，取出0-N匹配的行（JOIN Key相同）。
  - 构造结果行，并检查是否满足JOIN条件，如果满足条件则输出。

以下是一个Hash Join的例子：

```
> EXPLAIN SELECT * FROM partsupp, supplier WHERE ps_suppkey = s_suppkey;
HashJoin(condition="ps_suppkey = s_suppkey", type="inner")
 Gather(concurrent=true)
  LogicalView(tables="partsupp_[0-7]", shardCount=8, sql="SELECT * FROM `partsupp` AS `partsupp`")
 Gather(concurrent=true)
  LogicalView(tables="supplier_[0-7]", shardCount=8, sql="SELECT * FROM `supplier` AS `supplier`")
```

Hash Join常出现在JOIN数据量较大的复杂查询、且无法通过索引Lookup来改善，这种情况下Hash Join是最优的选择。例如上面的例子中，partsupp表和supplier表均为全表扫描，数据量较大，适合使用HashJoin。

由于Hash Join的内表需要用于构造内存中的哈希表，内表的数据量一般小于外表。通常优化器可以自动选择出最优的JOIN顺序。如果需要手动控制，也可以通过下面的Hint。

通过如下Hint可以强制PolarDB-X使用Hash Join以及确定JOIN顺序：

```
/*+TDDL:HASH_JOIN(table_outer, table_inner)*/ SELECT ...
```

Lookup Join (BKAJoin)

Lookup Join是另一种常用的等值JOIN算法，常用于数据量较小的情况。它的原理如下：

1. 遍历外表（左表，通常是数据量较小的一边）数据，对于外表中的每批（例如1000行）数据。
2. 将这一批数据的JOIN Key拼成一个 `IN (....)` 条件，加到内表的查询中。
3. 执行内表查询，得到JOIN匹配的行。
4. 借助哈希表，为外表的每行找到匹配的内表行，组合并输出。

以下是一个Lookup Join (BKAJoin）的例子：

```
> EXPLAIN SELECT * FROM partsupp, supplier WHERE ps_suppkey = s_suppkey AND ps_partkey = 123;
BKAJoin(condition="ps_suppkey = s_suppkey", type="inner")
 LogicalView(tables="partsupp_3", sql="SELECT * FROM `partsupp` AS `partsupp` WHERE (`ps_partkey` = ?)")
 Gather(concurrent=true)
  LogicalView(tables="supplier_[0-7]", shardCount=8, sql="SELECT * FROM `supplier` AS `supplier` WHERE (`s_suppkey` IN ('?'))")
```

Lookup Join通常用于外表数据量较小的情况，例如上面的例子中，左表partsupp由于存在 `ps_partkey = 123` 的过滤条件，仅有几行数据。此外，右表的 `s_suppkey IN (...)` 查询命中了主键索引，这也使得Lookup Join的查询代价进一步降低。

通过如下Hint可以强制PolarDB-X使用LookupJoin以及确定JOIN顺序：

```
/*+TDDL:BKA_JOIN(table_outer, table_inner)*/ SELECT ...
```

> **说明** Lookup Join的内表只能是单张表，不可以是多张表JOIN的结果。

Sort-Merge Join

Sort-Merge Join是另一种等值JOIN算法，它依赖左右两边输入的顺序，必须按JOIN Key排序。它的原理如下：

1. 开始Sort-Merge Join之前，输入端必须排序（借助MergeSort或MemSort）。
2. 比较当前左右表输入的行，并按以下方式操作，不断消费左右两边的输入：
   - 如果左表的JOIN Key较小，则消费左表的下一条数据。
   - 如果右表的JOIN Key较小，则消费右表的下一条数据。
   - 如果左右表JOIN Key相等，说明获得了1条或多条匹配，检查是否满足JOIN条件并输出。

以下是一个Sort-Merge Join的例子：

```
> EXPLAIN SELECT * FROM partsupp, supplier WHERE ps_suppkey = s_suppkey ORDER BY s_suppkey;
SortMergeJoin(condition="ps_suppkey = s_suppkey", type="inner")
 MergeSort(sort="ps_suppkey ASC")
  LogicalView(tables="QIMU_0000_GROUP,QIMU_0001_GROUP.partsupp_[0-7]", shardCount=8, sql="SELECT * FROM `partsupp` AS `partsupp` ORDER BY `ps_suppkey`")
 MergeSort(sort="s_suppkey ASC")
  LogicalView(tables="QIMU_0000_GROUP,QIMU_0001_GROUP.supplier_[0-7]", shardCount=8, sql="SELECT * FROM `supplier` AS `supplier` ORDER BY `s_suppkey`")
```

注意上面执行计划中的 MergeSort算子以及下推的 `ORDER BY`，这保证了 `Sort-Merge Join` 两边的输入按JOIN Key即 `s_suppkey (ps_suppkey)` 排序。

Sort-Merge Join由于需要额外的排序步骤，通常Sort-Merge Join并不是最优的。但是，某些情况下客户端查询恰好也需要按JOIN Key排序（上面的例子），这时候使用Sort-Merge Join是较优的选择。

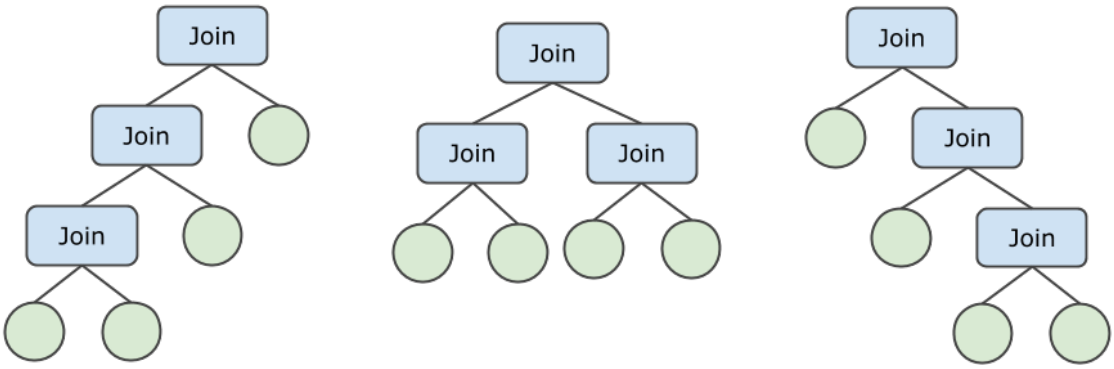
通过如下Hint可以强制PolarDB-X使用Sort-Merge Join：

```
/*+TDDL:SORT_MERGE_JOIN(table_a, table_b)*/ SELECT ...
```

JOIN顺序

在多表连接的场景中，优化器的一个很重要的任务是决定各个表之间的连接顺序，因为不同的连接顺序会影响中间结果集的大小，进而影响到计划整体的执行代价。

例如，对于4张表JOIN（暂不考虑下推的情形），JOIN Tree可以有如下3种形式，同时表的排列又有 `4! = 24` 种，一共有72种可能的JOIN顺序。



给定N个表的JOIN，PolarDB-X采用自适应的策略生成最佳JOIN计划：

- 当（未下推的）N较小时，采取Bushy枚举策略，会在所有JOIN顺序中选出最优的计划。
- 当（未下推的）表的数量较多时，采取Zig-Zag（锯齿状）或Left-Deep（左深树）的枚举策略，选出最优的Zig-Zag或Left-Deep执行计划，以减少枚举的次数和代价。

PolarDB-X使用基于代价的优化器（Cost-based Optimizer，CBO）选择出总代价最低的JOIN 顺序。详情参见查询优化器介绍。

此外，各个JOIN算法对左右输入也有不同的偏好，例如，Hash Join中右表作为内表用于构建哈希表，因此应当将较小的表置于右侧。这些也同样会在CBO中被考虑到。

子查询

根据是否存在关联项，子查询可以分为非关联子查询和关联子查询。非关联子查询是指该子查询的执行不依赖外部查询的变量，这种子查询一般只需要计算一次；而关联子查询中存在引用自外层查询的变量，逻辑上，这种子查询需要每次带入相应的变量、计算多次。

```
/* 例子：非关联子查询 */
SELECT * FROM lineitem WHERE l_partkey IN (SELECT p_partkey FROM part);
/* 例子：关联子查询（l_suppkey 是关联项） */
SELECT * FROM lineitem WHERE l_partkey IN (SELECT ps_partkey FROM partsupp WHERE ps_suppkey = l_suppkey);
```

PolarDB-X子查询支持绝大多数的子查询写法，具体参见SQL使用限制。

对于多数常见的子查询形式，PolarDB-X可以将其改写为高效的SemiJoin或类似的基于JOIN的计算方式。这样做的好处是显而易见的。当数据量较大时，无需真正带入不同参数循环迭代，大大降低了执行代价。这种查询改写技术称为子查询的去关联化（Unnesting）。

下面是2个子查询去关联化的例子，可以看到执行计划中使用JOIN代替了子查询。

```
> EXPLAIN SELECT p_partkey, (
    SELECT COUNT(ps_partkey) FROM partsupp WHERE ps_suppkey = p_partkey
    ) supplier_count FROM part;
Project(p_partkey="p_partkey", supplier_count="CASE(IS NULL($10), 0, $9)", cor=[$cor0])
 HashJoin(condition="p_partkey = ps_suppkey", type="left")
  Gather(concurrent=true)
   LogicalView(tables="part_[0-7]", shardCount=8, sql="SELECT * FROM `part` AS `part`")
  Project(count(ps_partkey)="count(ps_partkey)", ps_suppkey="ps_suppkey", count(ps_partkey)2="count
(ps_partkey)")
    HashAgg(group="ps_suppkey", count(ps_partkey)="SUM(count(ps_partkey))")
     Gather(concurrent=true)
      LogicalView(tables="partsupp_[0-7]", shardCount=8, sql="SELECT `ps_suppkey`, COUNT(`ps_partkey
`) AS `count(ps_partkey)` FROM `partsupp` AS `partsupp` GROUP BY `ps_suppkey`")
```

```
> EXPLAIN SELECT p_partkey, (
    SELECT COUNT(ps_partkey) FROM partsupp WHERE ps_suppkey = p_partkey
    ) supplier_count FROM part;
Project(p_partkey="p_partkey", supplier_count="CASE(IS NULL($10), 0, $9)", cor=[$cor0])
 HashJoin(condition="p_partkey = ps_suppkey", type="left")
  Gather(concurrent=true)
   LogicalView(tables="part_[0-7]", shardCount=8, sql="SELECT * FROM `part` AS `part`")
  Project(count(ps_partkey)="count(ps_partkey)", ps_suppkey="ps_suppkey", count(ps_partkey)2="count
(ps_partkey)")
    HashAgg(group="ps_suppkey", count(ps_partkey)="SUM(count(ps_partkey))")
     Gather(concurrent=true)
      LogicalView(tables="partsupp_[0-7]", shardCount=8, sql="SELECT `ps_suppkey`, COUNT(`ps_partkey
`) AS `count(ps_partkey)` FROM `partsupp` AS `partsupp` GROUP BY `ps_suppkey`")
```

某些少见情形下，PolarDB-X无法将子查询进行去关联化，这时候会采用迭代执行的方式。如果外层查询数据量很大，迭代执行可能会非常慢。

下面这个例子中，由于 `OR l_partkey < 50` 的存在，导致子查询无法被去关联化，因而采用了迭代执行：

```
> EXPLAIN SELECT * FROM lineitem WHERE l_partkey IN (SELECT ps_partkey FROM partsupp WHERE ps_supp
key = l_suppkey) OR l_partkey IS NOT
Filter(condition="IS(in,[$1])[29612489] OR l_partkey < ?0")
 Gather(concurrent=true)
  LogicalView(tables="QIMU_0000_GROUP,QIMU_0001_GROUP.lineitem_[0-7]", shardCount=8, sql="SELECT
* FROM `lineitem` AS `lineitem`")
>> individual correlate subquery : 29612489
Gather(concurrent=true)
 LogicalView(tables="QIMU_0000_GROUP,QIMU_0001_GROUP.partsupp_[0-7]", shardCount=8, sql="SELECT
* FROM (SELECT `ps_partkey` FROM `partsupp` AS `partsupp` WHERE (`ps_suppkey` = `l_suppkey`)) AS
`t0` WHERE (((`l_partkey` = `ps_partkey`) OR (`l_partkey` IS NULL)) OR (`ps_partkey` IS NULL))")
```

这种情形下，建议改写SQL去掉子查询的OR条件。

18.4.6. 优化聚合与排序

本文介绍如何优化器和执行器如何处理聚合（Group-by）与排序（Order-by）算子，以达到减少数据传输量和提高执行效率的效果。

基本概念

聚合操作（Aggregate，简称Agg）语义为按照 GROUP BY 指定列对输入数据进行聚合的计算，或者不分组、对所有数据进行聚合的计算。PolarDB-X支持如下聚合函数：

- COUNT
- SUM
- AVG
- MAX
- MIN
- BIT_OR
- BIT_XOR
- GROUP_CONCAT

排序操作（Sort）语义为按照指定的ORDER BY列对输入进行排序。

本文介绍均为不下推的Agg或Sort的算子的实现。如果已被下推到LogicalView中，则由存储层MySQL来选择执行方式。

聚合（Agg）

聚合（Agg）由两种主要的算子HashAgg和SortAgg实现。

HashAgg

HashAgg利用哈希表实现聚合：

1. 根据输入行的分组列的值，通过Hash找到对应的分组。
2. 按照指定的聚合函数，对该行进行聚合计算。
3. 重复以上步骤直到处理完所有的输入行，最后输出聚合结果。

```
> explain select count(*) from t1 join t2 on t1.id = t2.id group by t1.name,t2.name;
Project(count(*)="count(*)")
 HashAgg(group="name,name0", count(*)="COUNT()")
  BKAJoin(condition="id = id", type="inner")
   Gather(concurrent=true)
    LogicalView(tables="t1", shardCount=2, sql="SELECT `id`, `name` FROM `t1` AS `t1`")
   Gather(concurrent=true)
    LogicalView(tables="t2_[0-3]", shardCount=4, sql="SELECT `id`, `name` FROM `t2` AS `t2` WHERE (`id` IN ('?'))")
```

Explain结果中，HashAgg算子还包含以下关键信息：

- group：表示GROUP BY字段，示例中为 name,name0 分别引用t1,t2表的name列，当存在相同别名会通过后缀数字区分 。
- 聚合函数：等号（=） 前为聚合函数对应的输出列名，其后为对应的计算方法。示例中 count(*)="COUNT()" ，第一个 count(*) 对应输出的列名，随后的 COUNT() 表示对其输入数据进行计数。

HashAgg对应可以通过Hint来关闭： /*+TDDL:cmd_extra(ENABLE_HASH_AGG=false)*/ 。

SortAgg

SortAgg在输入数据已按分组列排序的情况，对各个分组依次完成聚合。

1. 保证输入按指定的分组列排序（例如，可能会看到 MergeSort 或 MemSort ）。

2. 逐行读入输入数据，如果分组与当前分组相同，则对其进行聚合计算。
3. 否则，如果分组与当前分组不同，则输出当前分组上的聚合结果。

相比 HashAgg，SortAgg 每次只要处理一个分组，内存消耗很小；相对的，HashAgg 需要把所有分组存储在内存中，需要消耗较多的内存。

```
> explain select count(*) from t1 join t2 on t1.id = t2.id group by t1.name,t2.name order by t1.name, t2.name;
Project(count(*)="count(*)")
 MemSort(sort="name ASC,name0 ASC")
  HashAgg(group="name,name0", count(*)="COUNT()")
   BKAJoin(condition="id = id", type="inner")
    Gather(concurrent=true)
     LogicalView(tables="t1", shardCount=2, sql="SELECT `id`, `name` FROM `t1` AS `t1`")
    Gather(concurrent=true)
     LogicalView(tables="t2_[0-3]", shardCount=4, sql="SELECT `id`, `name` FROM `t2` AS `t2` WHERE (`id` IN ('?'))")
```

SortAgg对应可以通过Hint来关闭： `/*+TDDL:cmd_extra(ENABLE_SORT_AGG=false)*/` 。

两阶段聚合优化

两阶段聚合，即通过将Agg拆分为部分聚合（Partial Agg）和最终聚合（Final Agg）的两个阶段，先对部分结果集做聚合，然后将这些部分聚合结果汇总，得到整体聚合的结果。

例如下面的 SQL 中，HashAgg 中拆分出的部分聚合（PartialAgg）会被下推至MySQL上的各个分表，而其中的 `AVG` 函数也被拆分成 `SUM` 和 `COUNT` 以实现两阶段的计算：

```
> explain select avg(age) from t2 group by name
Project(avg(age)="sum_pushed_sum / sum_pushed_count")
 HashAgg(group="name", sum_pushed_sum="SUM(pushed_sum)", sum_pushed_count="SUM(pushed_count)")
  Gather(concurrent=true)
   LogicalView(tables="t2_[0-3]", shardCount=4, sql="SELECT `name`, SUM(`age`) AS `pushed_sum`, COUNT(`age`) AS `pushed_count` FROM `t2` AS `t2` GROUP BY `name`")
```

两阶段聚合的优化能大大减少数据传输量、提高执行效率。

排序（Sort）

PolarDB-X中的排序算子主要包括 MemSort、TopN，以及 MergeSort。

- MemSort

  PolarDB-X中的通用的排序实现为MemSort算子，即内存中运行快速排序（Quick Sort）算法。

  下面是一个用到MemSort算子的例子：

```
> explain select t1.name from t1 join t2 on t1.id = t2.id order by t1.name,t2.name;
Project(name="name")
 MemSort(sort="name ASC,name0 ASC")
  Project(name="name", name0="name0")
   BKAJoin(condition="id = id", type="inner")
    Gather(concurrent=true)
     LogicalView(tables="t1", shardCount=2, sql="SELECT `id`, `name` FROM `t1` AS `t1`")
    Gather(concurrent=true)
     LogicalView(tables="t2_[0-3]", shardCount=4, sql="SELECT `id`, `name` FROM `t2` AS `t2` WHERE (`id` IN ('?'))")
```

- TopN

  当SQL中ORDER BY和LIMIT一起出现时，Sort算子和Limit算子会合并成TopN算子。

  TopN算子维护一个最大或最小堆，按照排序键的值，堆中始终保留最大或最小的N行数据。当处理完全部的输入数据时，堆中留下的N个行（或小于N个）就是需要的结果。

  下面是一个用到 TopN 算子的例子：

```
> explain select t1.name from t1 join t2 on t1.id = t2.id order by t1.name,t2.name limit 10;
Project(name="name")
 TopN(sort="name ASC,name0 ASC", offset=0, fetch=?0)
  Project(name="name", name0="name0")
   BKAJoin(condition="id = id", type="inner")
    Gather(concurrent=true)
     LogicalView(tables="t1", shardCount=2, sql="SELECT `id`, `name` FROM `t1` AS `t1`")
    Gather(concurrent=true)
     LogicalView(tables="t2_[0-3]", shardCount=4, sql="SELECT `id`, `name` FROM `t2` AS `t2` WHERE (`id` IN ('?'))")
```

- MergeSort

  通常，只要语义允许，SQL中的排序操作会被下推到MySQL上执行，而PolarDB-X执行层只做最后的归并操作，即MergeSort。严格来说，MergeSort 不仅仅是排序，更是一种数据重分布算子（类似 Gather）。

  下面的SQL是对t1表进行排序，经过PolarDB-X查询优化器的优化，Sort算子被下推至各个MySQL分片中执行，最终只在上层做归并操作。

```
> explain select name from t1 order by name;
MergeSort(sort="name ASC")
 LogicalView(tables="t1", shardCount=2, sql="SELECT `name` FROM `t1` AS `t1` ORDER BY `name`")
```

  相比 MemSort，MergeSort 算法可以减少PolarDB-X层的内存消耗，并充分利用 MySQL 层的计算能力。

优化组合的例子

下面是一个组合优化的例子，在这个例子中，用到了以下优化规则：

- Agg下推穿过Join
- Join算法选择为SortMergeJoin
- Agg算法选择为SortAgg
- SortMergeJoin中需要的排序利用了SortAgg输出的有序
- 两阶段Agg
- Agg下推

- Sort下推

```
> explain select count(*) from t1 join t2 on t1.name = t2.name group by t1.name;
Project(count(*)="count(*) * count(*)0")
 SortMergeJoin(condition="name = name", type="inner")
  SortAgg(group="name", count(*)="SUM(count(*))")
   MergeSort(sort="name ASC")
    LogicalView(tables="t1", shardCount=2, sql="SELECT `name`, COUNT(*) AS `count(*)` FROM `t1` AS `t
1` GROUP BY `name` ORDER BY `name`")
  SortAgg(group="name", count(*)="SUM(count(*))")
   MergeSort(sort="name ASC")
    LogicalView(tables="t2_[0-3]", shardCount=4, sql="SELECT `name`, COUNT(*) AS `count(*)` FROM `t2`
AS `t2` GROUP BY `name` ORDER BY `name`")
```

18.4.7. 执行计划管理

本文介绍如何管理执行计划，将重复或者复杂查询的执行计划长久地保存下来。

背景信息

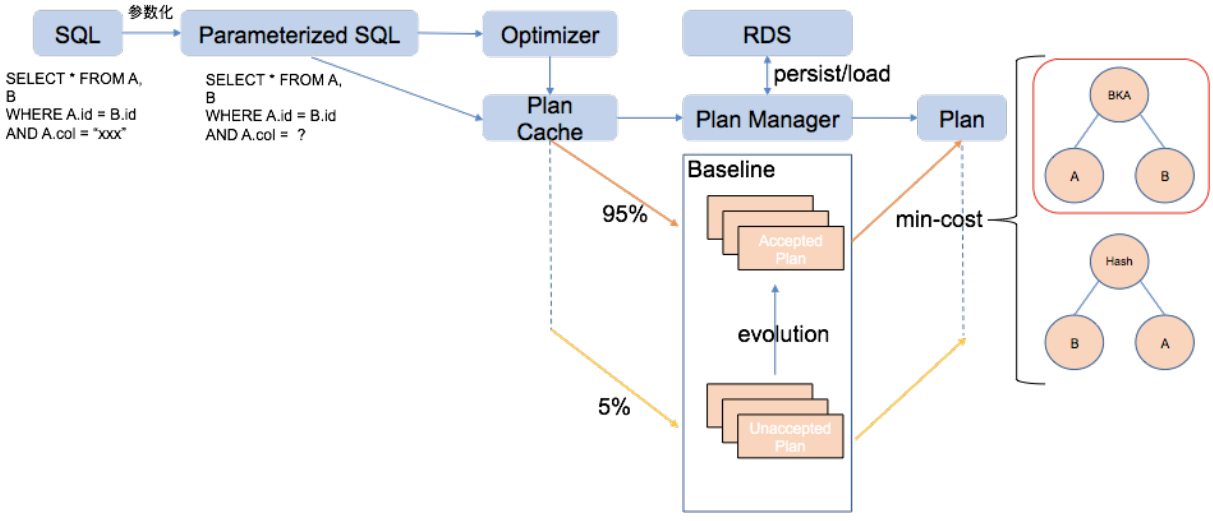
对于每一条SQL，优化器都会生成相应执行计划。但是很多情况下，应用请求的SQL都是重复的（仅参数不同），参数化之后的SQL完全相同。这时，可以按照参数化之后的SQL构造一个缓存，将除了参数以外的各种信息（比如执行计划）缓存起来，称为执行计划缓存（Plan Cache）。

另一方面，对于较复杂的查询（例如涉及到多个表的Join），为了使其执行计划能保持相对稳定，不因为版本升级等原因发生变化。执行计划管理（Plan Management）为每个SQL记录一组执行计划，该执行计划会被持久化地保存，即使版本升级也会保留。

工作流程概览

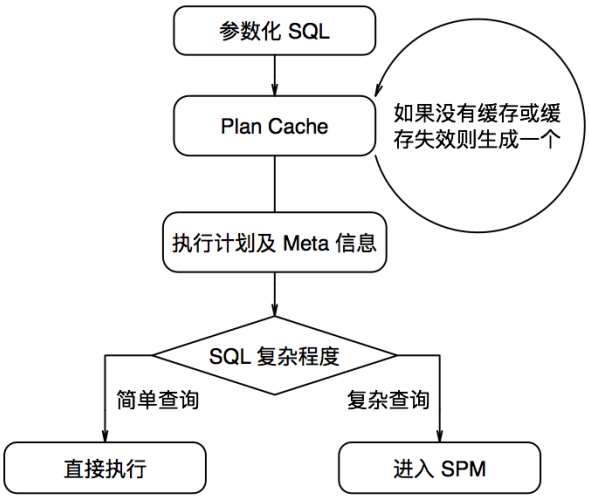
当PolarDB-X收到一条查询SQL时，会经历以下流程：

1. 对查询SQL进行参数化处理，将所有参数替换为占位符 `?` 。
2. 以参数化的SQL作为Key，查找执行计划缓存中是否有缓存；如果没有，则调用优化器进行优化。
3. 如果该SQL是简单查询，则直接执行，跳过执行计划管理相关步骤。
4. 如果该SQL是复杂查询，则使用基线（Baseline）中固化的执行计划；如果有多个，则选择代价最低的那个。



执行计划缓存

PolarDB-X默认开启执行计划缓存功能。EXPLAIN结果中的 HitCache 表示当前SQL是否命中执行计划缓存。开启执行计划缓存后，PolarDB-X会对SQL做参数化处理，参数化会将SQL中的常量用占位符 ? 替换，并构建出相应的参数列表。在执行计划中也可以看到LogicalView算子的SQL中含有 ? 。



执行计划管理

对于复杂SQL，经过执行计划缓存之后，还会经过执行计划管理流程。

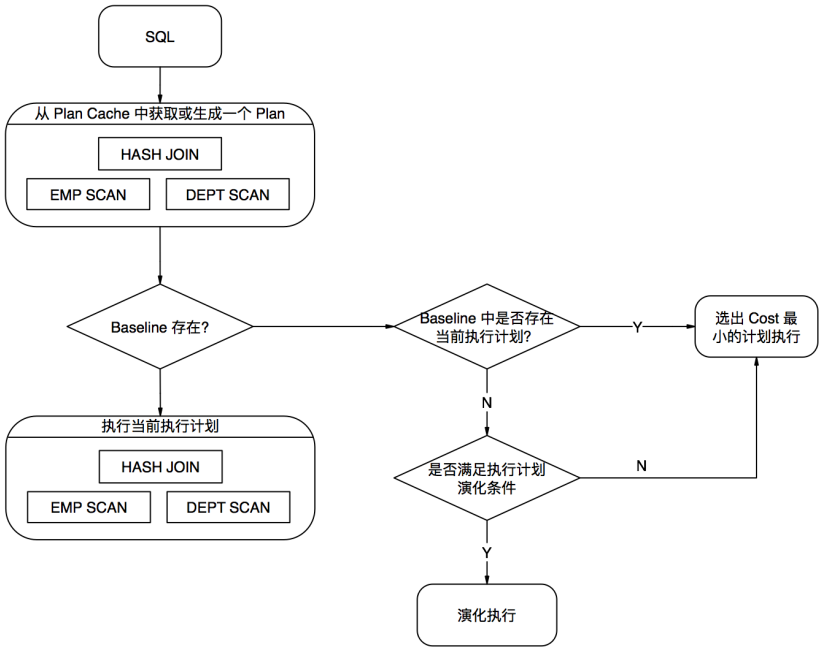
执行计划缓存和执行计划管理都是采用参数化后的SQL作为Key来执行计划。执行计划缓存中会缓存所有SQL的执行计划，而执行计划管理仅对复杂查询SQL进行处理。

由于受到具体参数的影响，SQL模版和最优的执行计划并非一一对应的。

在执行计划管理中，每一条SQL对应一个基线，每个基线中包含一个或多个执行计划。实际使用中，会根据当时的参数选择其中代价最小的执行计划来执行。

计划选择

当执行计划缓存中的执行计划走进执行计划管理时，SPM会操作一个流程判断该执行计划是否是已知的，是已知的话，是否代价是最小的；不是已知的话，是否需要执行一下以判断该执行计划的优化程度。



运维指令

PolarDB-X提供了丰富的指令集用于管理执行计划，语法如下：

```
BASELINE (LOAD|PERSIST|CLEAR|VALIDATE|LIST|DELETE) [Signed Integer,Signed Integer....]
BASELINE (ADD|FIX) SQL (HINT Select Statemtnt)
```

- BASELINE (ADD|FIX) SQL <HINT> <Select Statement>：将SQL以HINT修复过后的执行计划记录固定下来。
- BASELINE LOAD：将系统表中指定的基线信息刷新到内存并使其生效。
- BASELINE LOAD_PLAN：将系统表中指定的执行计划信息刷新到内存并使其生效。
- BASELINE LIST：列出当前所有的基线信息。BASELINE PERSIST：将指定的基线落盘。
- BASELINE PERSIST_PLAN：将指定的执行计划落盘。
- BASELINE CLEAR：内存中清理某个基线。
- BASELINE CLEAR_PLAN：内存中清理某个执行计划。
- BASELINE DELETE：磁盘中删除某个基线。
- BASELINE DELETE_PLAN：磁盘中删除某个执行计划。

执行计划调优实战

数据发生变化或PolarDB-X优化器引擎升级后，针对同一条SQL，有可能会出现更好的执行计划。SPM在自动演化时会将CBO优化自动发现的更优执行计划加入到SQL的基线中。除此以外，您也可以通过SPM的指令主动优化执行计划。

例如以下的SQL:

```
SELECT *
FROM lineitem JOIN part ON l_partkey=p_partkey
WHERE p_name LIKE '%green%';
```

正常EXPLAIN发现该SQL生成的执行计划使用的是Hash Join，并且在Baseline List的基线中，该SQL仅有这一个执行计划：

```
mysql> explain select * from lineitem join part on l_partkey=p_partkey where p_name like '%geen%';
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
| LOGICAL PLAN
|
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
| Gather(parallel=true)
|
|  ParallelHashJoin(condition="l_partkey = p_partkey", type="inner")
|
|    LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppk
ey`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus
`, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `li
neitem` AS `lineitem`", parallel=true) |
|    LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_b
rand`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE (
`p_name` LIKE ?)", parallel=true)                                                       |
| HitCache:true
|
|
|
|
|
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
7 rows in set (0.06 sec)
mysql> baseline list;
+------------+-----------------------------------------------------------------+-----------+-------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+-------+----------+
| BASELINE_ID | PARAMETERIZED_SQL                              | PLAN_ID    | EXTERNALIZED_PLAN
| FIXED | ACCEPTED |
+------------+-----------------------------------------------------------------+-----------+-------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+-------+----------+
| -399023558 | SELECT *
FROM lineitem
  JOIN part ON l_partkey = p_partkey
WHERE p_name LIKE ? | -935671684 |
Gather(parallel=true)
 ParallelHashJoin(condition="l_partkey = p_partkey", type="inner")
```

```
  LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppke
y`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus`
, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `lin
eitem` AS `lineitem`", parallel=true)
  LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_br
and`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE (
`p_name` LIKE ?)", parallel=true)
|   0|     1|
+-------------+-------------------------------------------------------------------------------+------------+-----------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+-------+----------+
1 row in set (0.02 sec)
```

假如这个SQL在某些条件下采用BKA Join（Lookup Join）会有更好的性能，那么首先需要想办法利用HINT引导PolarDB-X生成符合预期的执行计划。BKA Join的HINT格式为：

```
/*+TDDL:BKA_JOIN(lineitem, part)*/
```

通过 `EXPLAIN [HINT] [SQL]` 观察出来的执行计划是否符合预期：

```
mysql> explain /*+TDDL:bka_join(lineitem, part)*/ select * from lineitem join part on l_partkey=p_partkey w
here p_name like '%geen%';
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
| LOGICAL PLAN
|
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
| Gather(parallel=true)
|
|  ParallelBKAJoin(condition="l_partkey = p_partkey", type="inner")
|
|    LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppk
ey`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus
`, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `li
neitem` AS `lineitem`", parallel=true) |
|    Gather(concurrent=true)
|
|     LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_
brand`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE
(`p_name` LIKE ?)")                                                                       |
| HitCache:false
|
|
|
|
|
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
8 rows in set (0.14 sec)
```

注意此时由于Hint的干预，Join的算法已修正为BKA Join。但是这并不会对基线造成变动，如果想以后每次遇到这条SQL都使用上面的计划，还需要将其加入到基线中。

可以采用执行计划管理的 `Baseline Add` 指令为该SQL增加一个执行计划。这样就会同时有两套执行计划存在于该SQL的基线中，CBO优化器会根据代价选择一个执行计划执行。

```
mysql> baseline add sql /*+TDDL:bka_join(lineitem, part)*/ select * from lineitem join part on l_partkey=p_p
artkey where p_name like '%geen%';
+-------------+--------+
| BASELINE_ID | STATUS |
+-------------+--------+
|  -399023558 | OK     |
+-------------+--------+
1 row in set (0.09 sec)
mysql> baseline list;
+-------------+-----------------------------------------------------------------------------+-------------+---------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------+-------+----------+
| BASELINE_ID | PARAMETERIZED_SQL                                   | PLAN_ID    | EXTERNALIZED_PLAN
| FIXED | ACCEPTED |
+-------------+------------------------------------------------------------------------------+-------------+--------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------+-------+----------+
|  -399023558 | SELECT *
FROM lineitem
  JOIN part ON l_partkey = p_partkey
WHERE p_name LIKE ? | -1024543942 |
Gather(parallel=true)
 ParallelBKAJoin(condition="l_partkey = p_partkey", type="inner")
  LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppke
y`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus`
, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `lin
eitem` AS `lineitem`", parallel=true)
  Gather(concurrent=true)
   LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_b
rand`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE (
`p_name` LIKE ?)")
 |    0 |      1 |
|  -399023558 | SELECT *
FROM lineitem
  JOIN part ON l_partkey = p_partkey
WHERE p_name LIKE ? |  -935671684 |
Gather(parallel=true)
 ParallelHashJoin(condition="l_partkey = p_partkey", type="inner")
  LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppke
y`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus`
, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `lin
eitem` AS `lineitem`", parallel=true)
  LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_br
and`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE (
`p_name` LIKE ?)", parallel=true)
        |    0 |      1 |
+-------------+------------------------------------------------------------------------------+-------------+--------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------+-------+----------+
2 rows in set (0.03 sec)
```

通过以上 `Baseline List` 指令展示出来的结果，可以看到基于BKA_JOIN的执行计划已增加到该 SQL的基线中。此时EXPLAIN这条SQL，发现随SQL中 `p_name LIKE ?` 条件变化，PolarDB-X会选择不同的执行计划。如果想让PolarDB-X固定使用上述的执行计划（而非在两个中挑选一个），可以采用 `Baseline Fix` 指令强制PolarDB-X走指定的执行计划。

```
mysql> baseline fix sql /*+TDDL:bka_join(lineitem, part)*/ select * from lineitem join part on l_partkey=p_pa
rtkey where p_name like '%geen%';
+-------------+--------+
| BASELINE_ID | STATUS |
+-------------+--------+
| -399023558 | OK     |
+-------------+--------+
1 row in set (0.07 sec)
mysql> baseline list\G
*************************** 1. row ***************************
    BASELINE_ID: -399023558
PARAMETERIZED_SQL: SELECT *
FROM lineitem
  JOIN part ON l_partkey = p_partkey
WHERE p_name LIKE ?
       PLAN_ID: -1024543942
EXTERNALIZED_PLAN:
Gather(parallel=true)
 ParallelBKAJoin(condition="l_partkey = p_partkey", type="inner")
  LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppke
y`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus`
, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `lin
eitem` AS `lineitem`", parallel=true)
  Gather(concurrent=true)
   LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_b
rand`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE (
`p_name` LIKE ?)")
       FIXED: 1
    ACCEPTED: 1
*************************** 2. row ***************************
    BASELINE_ID: -399023558
PARAMETERIZED_SQL: SELECT *
FROM lineitem
  JOIN part ON l_partkey = p_partkey
WHERE p_name LIKE ?
       PLAN_ID: -935671684
EXTERNALIZED_PLAN:
Gather(parallel=true)
 ParallelHashJoin(condition="l_partkey = p_partkey", type="inner")
  LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppke
y`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus`
, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `lin
eitem` AS `lineitem`", parallel=true)
  LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_br
and`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE (
`p_name` LIKE ?)", parallel=true)
       FIXED: 0
    ACCEPTED: 1
2 rows in set (0.01 sec)
```

`Baseline Fix` 指令执行完后，可以看到BKA Join执行计划的 `Fix` 状态位已被置为1。此时就算不加HINT，任意条件下Explain这条SQL，都一定会采用这个执行计划。

```
mysql> explain select * from lineitem join part on l_partkey=p_partkey where p_name like '%green%';
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| LOGICAL PLAN
|
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Gather(parallel=true)
|
|  ParallelBKAJoin(condition="l_partkey = p_partkey", type="inner")
|
|   LogicalView(tables="[00-03].lineitem", shardCount=4, sql="SELECT `l_orderkey`, `l_partkey`, `l_suppkey`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_returnflag`, `l_linestatus`, `l_shipdate`, `l_commitdate`, `l_receiptdate`, `l_shipinstruct`, `l_shipmode`, `l_comment` FROM `lineitem` AS `lineitem`", parallel=true) |
|   Gather(concurrent=true)
|
|    LogicalView(tables="[00-03].part", shardCount=4, sql="SELECT `p_partkey`, `p_name`, `p_mfgr`, `p_brand`, `p_type`, `p_size`, `p_container`, `p_retailprice`, `p_comment` FROM `part` AS `part` WHERE (`p_name` LIKE ?)")                    |
| HitCache:true
|
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
8 rows in set (0.01 sec)
```

18.5. 智能索引推荐

索引优化通常需要依赖运维或开发人员对数据库引擎内部优化和执行原理的深入理解。为优化体验和降低操作门槛，PolarDB-X推出了基于代价优化器的索引推荐功能，可根据查询语句分析并推荐索引，帮助您降低查询耗时，提升数据库性能。

注意事项

索引推荐功能仅针对您当前指定的SQL查询语句进行分析与推荐。在根据推荐的信息创建索引前，您需要评估创建该索引对其它查询的影响。

环境说明

TPC-H是业界常用的基准测试方法，由TPC委员会制定发布，用于评测数据库的分析型查询能力。TPC-H基准测试方法包含8张数据表、22条复杂的SQL查询（即Q1~Q22）。下图为执行TPC-H中的Q17（小订单收入查询）的返回信息，可查看到执行该查询语句消耗的时间为28.76秒。本文将通过智能索引推荐功能，优化该查询语句的执行效率。

> **说明** 关于TCP-H中查询语句的详细介绍，请参见TPC-H的SQL语句分析。
>
> 本文的TPC-H的实现基于TPC-H的基准测试，并不能与已发布的TPC-H基准测试结果相比较，本文中的测试并不符合TPC-H基准测试的所有要求。

智能索引优化前

智能索引优化前

```
mysql> SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
    -> FROM lineitem,
    ->      part
    -> WHERE p_partkey = l_partkey
    ->   AND p_brand = 'Brand#23'
    ->   AND p_container = 'MED BOX'
    ->   AND l_quantity <
    ->     (SELECT 0.2 * avg(`l_quantity`)
    ->      FROM lineitem
    ->      WHERE l_partkey = p_partkey);
+--------------------------------------+
| avg_yearly                           |
+--------------------------------------+
| 348406.05428571428571428571428571286 |
+--------------------------------------+
1 row in set (28.76 sec)
```

步骤一：查询智能索引推荐信息

如需查询某个查询语句的智能索引推荐信息，您只需在该查询语句前增加**EXPLAIN ADVISOR**命令，示例如下：

```
EXPLAIN ADVISOR
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem,
   part
WHERE p_partkey = l_partkey
 AND p_brand = 'Brand#23'
 AND p_container = 'MED BOX'
 AND l_quantity <
  (SELECT 0.2 * avg(`l_quantity`)
   FROM lineitem
   WHERE l_partkey = p_partkey);
```

执行上述命令后，PolarDB-X将返回推荐的索引创建语句、添加索引前后的代价等信息，详细的返回信息及其注释如下所示：

> **说明**
>
> - 本案例中，预计磁盘I/O提升百分比为3024.7%，表明使用推荐的索引将带来较大的收益。
> - 当PolarDB-X无法推荐索引时，返回信息中会建议您在业务低峰期，对目标表执行Analyze Table命令刷新统计信息（该操作会消耗较大的I/O资源）。当统计信息更新后，再次执行索引推荐可获得更准确的索引。

```
IMPROVE_VALUE: 2465.3%     # 预计综合代价提升百分比
 IMPROVE_CPU: 59377.4%     # 预计CPU提升百分比
 IMPROVE_MEM: 0.4%         # 预计内存提升百分比
  IMPROVE_IO: 3024.7%      # 预计磁盘I/O提升百分比
 IMPROVE_NET: 2011.1%      # 预计网络传输提升百分比
BEFORE_VALUE: 4.711359845E8 # 添加索引前综合代价值
 BEFORE_CPU: 1.19405577E7  # 添加索引前CPU估算值
 BEFORE_MEM: 426811.2      # 添加索引前内存消耗估算值
  BEFORE_IO: 44339         # 添加索引前磁盘I/O估算值
 BEFORE_NET: 47.5          # 添加索引前网络传输估算值
AFTER_VALUE: 1.83655008E7  # 添加索引后综合代价值
  AFTER_CPU: 20075.8       # 添加索引后CPU估算值
  AFTER_MEM: 425016        # 添加索引后内存消耗估算值
   AFTER_IO: 1419          # 添加索引后磁盘I/O估算值
  AFTER_NET: 2.2           # 添加索引后网络传输估算值
ADVISE_INDEX: ALTER TABLE `lineitem` ADD INDEX `__advise_index_lineiteml_partkey`(`l_partkey`);
/* ADVISE_INDEX中的内容为推荐的索引创建语句 */
   NEW_PLAN:             # 添加索引后预计执行计划
Project(avg_yearly="$f0 / ?0")
 HashAgg($f0="SUM(l_extendedprice)")
  Filter(condition="l_quantity < $16 * f17w0$o0")
   SortWindow(p_partkey="p_partkey", l_partkey="l_partkey", l_quantity="l_quantity", l_extendedprice="
l_extendedprice", $16="$16", f5w0$o0="window#0AVG($2)", Reference Windows="window#0=window(parti
tion {1} order by [] range between UNBOUNDED PRECEDING and UNBOUNDED PRECEDING aggs [AVG($2)])")
     MemSort(sort="l_partkey ASC")
      BKAJoin(condition="l_partkey = p_partkey", type="inner")
       Gather(concurrent=true)
        LogicalView(tables="[0000,0001].part", shardCount=2, sql="SELECT `p_partkey` FROM `part` AS `pa
rt` WHERE ((`p_brand` = ?) AND (`p_container` = ?))")
       Gather(concurrent=true)
        LogicalView(tables="[0000,0001].lineitem", shardCount=2, sql="SELECT `l_partkey`, `l_quantity`, `l
_extendedprice`, ? AS `$16` FROM `lineitem` AS `lineitem` WHERE (`l_partkey` IN (...))")
     INFO: LOCAL_INDEX   # 其它信息
```

步骤二：根据推荐信息创建索引

1. 评估创建该索引带来的收益，然后根据返回结果ADVISE_INDEX中的SQL语句创建索引。

   ```
   ALTER TABLE `lineitem` ADD INDEX `__advise_index_lineiteml_partkey`(`l_partkey`);
   ```

2. 再次执行TPC-H中的Q17（小订单收入查询），耗时减少至1.41秒，查询效率得到大幅提升。

   智能索引优化后

```
mysql> SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
    -> FROM lineitem,
    ->      part
    -> WHERE p_partkey = l_partkey
    ->   AND p_brand = 'Brand#23'
    ->   AND p_container = 'MED BOX'
    ->   AND l_quantity <
    ->     (SELECT 0.2 * avg(`l_quantity`)
    ->      FROM lineitem
    ->      WHERE l_partkey = p_partkey);
+---------------------------------------+
| avg_yearly                            |
+---------------------------------------+
| 348406.05428571428571428571428571428 |
+---------------------------------------+
1 row in set (1.41 sec)
```

19.系统运维

19.1. 高危类SQL自动保护

为避免因误操作导致数据丢失，PolarDB-X默认禁止执行高危类SQL，例如全表删除（即不带WHERE或LIMIT条件的DELETE语句）和全表更新（即不带WHERE或LIMIT条件的UPDATE语句）等语句。但您也可以通过HINT语句避开上述自动保护，强制执行全表删除或更新操作。

语句

您可以在UPDATE或DELETE语句中加上如下HINT执行全表删除或更新：

```
HINT：/!TDDL:FORBID_EXECUTE_DML_ALL=false*/
```

示例

- 若DELETE语句中未加任何WHERE或LIMIT条件，执行时会被拦截且出现如下错误提示：

  ```
  mysql> delete from tt;
  ERR-CODE: [TDDL-4620][ERR_FORBID_EXECUTE_DML_ALL] Forbid execute DELETE ALL or UPDATE ALL sql.
  More: [http://middleware.alibaba-inc.com/faq/faqByFaqCode.html?faqCode=TDDL-4620]
  ```

  在上述语句中加上如下HINT则可成功执行：

  ```
  mysql> /!TDDL:FORBID_EXECUTE_DML_ALL=false*/delete from tt;
  Query OK, 10 row affected (0.21 sec)
  ```

- 若UPDATE语句中未加任何WHERE或LIMIT条件，执行时会被拦截且出现如下错误提示：

  ```
  mysql> update tt set id = 1;
  ERR-CODE: [TDDL-4620][ERR_FORBID_EXECUTE_DML_ALL] Forbid execute DELETE ALL or UPDATE ALL sql.
  More: [http://middleware.alibaba-inc.com/faq/faqByFaqCode.html?faqCode=TDDL-4620]
  ```

  在上述语句中加上如下HINT则可成功执行：

  ```
  mysql> /!  TDDL:FORBID_EXECUTE_DML_ALL=false*/update tt set id = 1;
  Query OK, 10 row affected (0.21 sec)
  ```

19.2. 待处理事件

PolarDB-X提供待处理事件服务，当出现待处理事件时，系统会通过邮件、短信和控制台等途径提醒您及时处理，本文将介绍如何在控制台上查看和管理待处理事件。

背景信息

PolarDB-X待处理事件包括：

- 运维事件：如网络变更等运维提醒，您可以自定义操作时间，维护系统会自动在预约时间进行操作。
- 通知事件：如版本升级等通知提醒，在收到通知事件消息后，您需要自行处理。

操作步骤

1. 登录云原生分布式数据库控制台。

2. 在左侧导航栏中单击**待处理事件**。

   > **说明** 若您在**待处理事件**中未看到任何提醒，说明当前您的实例中不存在任何未处理的运维事件。

3. 在**待处理事件**页，单击目标实例的事件类型页签并选择实例所在地域，您可以在下方实例列表总查看待处理事件的详细信息。
   - 针对运维事件，若您不指定运维事件的操作时间，系统会在默认事件内自动执行相应的运维操作。

     若您需要修改操作时间，请参见如下步骤：

     a. 选中对应的实例ID。

     b. 单击**自定义操作时间**，在弹出的对话框中设置操作时间。

     c. 单击**确定**。
   - 针对通知事件，需要您自行处理，处理完成后可以在控制台上确认已处理，该通知之后便不会出现在**待处理事件**中。

     > **说明** 处理完成的事件可以在**历史事件**中查看。

20.全局二级索引

20.1. 使用全局二级索引

PolarDB-X支持全局二级索引，本文将介绍如何创建、使用全局二级索引功能。

前提条件

MySQL版本需为5.7或以上，且内核小版本需为5.4.1或以上。

使用限制

关于创建和使用GSI的相关限制，请参见使用全局二级索引时的注意事项。

创建GSI

PolarDB-X对MySQL DDL语法进行了扩展，增加定义GSI的语法。使用方式与在MySQL上创建索引一致。

- 建表时定义GSI

```
GLOBAL INDEX `g_i_seller`(`seller_id`) COVERING(`order_snapshot`) DBPARTITION BY HASH(`seller_id`)
```

索引名 索引列 覆盖列 索引表分库分表子句

- 建表后添加GSI

```
GLOBAL INDEX `g_i_seller` ON `t_order`(`seller_id`) COVERING(`order_snapshot`) DBPARTITION BY HASH(`seller_id`)
```

索引名 主表名 索引列 覆盖列 索引表分库分表子句

> 说明
>
> - 索引名：作为索引表的名字，用于创建索引表。
> - 索引列：索引表的分库分表键，即索引分库分表子句中用到的所有列。
> - 覆盖列：索引表中的其他列，默认包含主键和主表的全部分库分表键。
> - 索引分库分表子句：索引表的分库分表算法，与CREATE TABLE中分库分表子句的语法一致。

示例

```
# 建表时定义 GSI
CREATE TABLE t_order (
`id` bigint(11) NOT NULL AUTO_INCREMENT,
`order_id` varchar(20) DEFAULT NULL,
`buyer_id` varchar(20) DEFAULT NULL,
`seller_id` varchar(20) DEFAULT NULL,
`order_snapshot` longtext DEFAULT NULL,
`order_detail` longtext DEFAULT NULL,
PRIMARY KEY (`id`),
GLOBAL INDEX `g_i_seller`(`seller_id`) COVERING (`id`, `order_id`, `buyer_id`, `order_snapshot`) dbpa
rtition by hash(`seller_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
# 添加 GSI
CREATE UNIQUE GLOBAL INDEX `g_i_buyer` ON `t_order`(`buyer_id`)
  COVERING(`seller_id`, `order_snapshot`)
  dbpartition by hash(`buyer_id`) tbpartition by hash(`buyer_id`) tbpartitions 3
```

> **说明** 更多关于创建全局二级索引的详情，请参见CREATE INDEX。

使用GSI

GSI创建完成后，可以通过如下方式指定查询使用索引表：

- 通过HINT指定索引

  您可以选择以下两种HINT语句中的任意一种指定使用目标索引进行查询。

  - 语句：

    ```
    FORCE INDEX({index_name})
    ```

    示例：

    ```
    SELECT a.*, b.order_id
     FROM t_seller a
      JOIN t_order b FORCE INDEX(g_i_seller) ON a.seller_id = b.seller_id
     WHERE a.seller_nick="abc";
    ```

  - 语法：

    ```
    /*+TDDL:INDEX({table_name/table_alias}, {index_name})*/
    ```

    示例：

    ```
    /*+TDDL:index(a, g_i_buyer)*/ SELECT * FROM t_order a WHERE a.buyer_id = 123
    ```

    > **说明** 如果查询需要使用索引中未包含的列，则首先查询索引表取得所有记录的主键和主表分库分表键，然后回查主表中取得缺少列的值，详细说明请参见INDEX HINT。

- 直接查询索引表

  如果索引表中包含了查询需要的所有列，可以直接查询索引表获得结果。

- 索引选择

对于带有全局二级索引的主表查询，PolarDB-X会自动选择出优化器认为代价最低的索引表（目前只支持覆盖索引选择）。

下面SQL查询的主表是 t_order ，带有 seller_id 等值过滤条件，同时涉及的 id 、 order_snapshot 和 seller_id 等列被全局二级索引 g_i_seller 覆盖。选择了覆盖索引 g_i_seller 既可以不回表，又可以明确减少分表的扫描数目（ seller_id 是 g_i_seller 的拆分键）。通过EXPLAIN可以看到PolarDB-X优化器确实选择了 g_i_seller 。

```
EXPLAIN SELECT t_order.id,t_order.order_snapshot FROM t_order WHERE t_order.seller_id = 's1';
IndexScan(tables="g_i_seller_sfL1_2", sql="SELECT `id`, `order_snapshot` FROM `g_i_seller` AS `g_i_seller` WHERE (`seller_id` = ?)")
```

- IGNORE INDEX与USE INDEX

  您可以通过以下HINT指定优化器使用或不使用某些索引。

  - 语句：

    ```
    IGNORE INDEX({index_name},...)
    ```

    示例：

    ```
    SELECT t_order.id,t_order.order_snapshot FROM t_order IGNORE INDEX(g_i_seller) WHERE t_order.seller_id = 's1';
    ```

  - 语句：

    ```
    USE INDEX({index_name},...)
    ```

    示例：

    ```
    SELECT t_order.id,t_order.order_snapshot FROM t_order USE INDEX(g_i_seller) WHERE t_order.seller_id = 's1';
    ```

20.2. 使用全局二级索引时的注意事项

本文汇总了在使用全局二级索引时的相关注意事项。

创建GSI时的注意事项

- 不支持在MySQL 5.6版本的RDS上创建GSI。
- 不支持在单表或广播表上创建GSI。
- 不支持在无主键的表上创建GSI。
- 不支持在UNIQUE GSI中通过任何方式使用前缀索引。
- 创建索引表时必须指定索引名。
- 创建索引表时必须指定分库或分库加分表组合的规则，不允许仅指定分表规则或不指定任何拆分规则。
- 索引表的INDEX列必须包含全部拆分键。
- GSI名称不可与主表上的其它局部索引名重复。
- GSI定义子句中，索引列与覆盖列不可重复。
- 索引表默认包含主表的全部主键和拆分键，如果没有显式包含在索引列中，默认添加到覆盖列。
- 对主表中的每个局部索引，如果引用的所有列均包含在索引表中，默认添加该局部索引到索引表。
- 对GSI的每个索引列，如果没有已经存在的索引，默认单独创建一个索引。

- 对包含多个索引列的GSI，默认创建一个联合局部索引，包含所有索引列。
- 索引定义中，索引列的length参数仅用于在索引表拆分键上创建局部索引。
- 建表后创建GSI时，会在GSI创建结束时自动进行数据校验，只有通过校验，创建GSI的DDL语句才能执行成功。

> **说明** 您也可以使用CHECK GLOBAL INDEX对索引数据进行校验或订正。

ALTER TABLE时的注意事项

- 下表汇总了使用ALTER TABLE语句变更列的支持情况。

语句	是否支持变更主表拆分键	是否支持变更主表主键（也即索引表主键）	是否支持变更本地唯一索引列	是否支持变更索引表拆分键	是否支持变更Unique Index列	是否支持变更Index列	是否支持变更Covering列
ADD COLUMN	无该场景	不支持	无该场景	无该场景	无该场景	无该场景	无该场景
ALTER COLUMN SET DEFAULT和ALTER COLUMN DROP DEFAULT	不支持	不支持	支持	不支持	不支持	不支持	不支持
CHANGE COLUMN	不支持	不支持	支持	不支持	不支持	不支持	不支持
DROP COLUMN	不支持	不支持	仅当唯一键中只有1列时支持	不支持	不支持	不支持	不支持
MODIFY COLUMN	不支持	不支持	支持	不支持	不支持	不支持	不支持

> **说明**
> - 考虑到全局二级索引的稳定性和性能情况，目前禁止直接使用DROP COLUMN命令删除全局二级索引中的列。如需删除全局二级索引中的某些列，您可以先使用DROP INDEX删除对应的全局二级索引，再重新创建一个新的二级索引，或提交工单联系技术支持进行删除。
> - 以上对列的分类存在重叠（如Index列包含索引表拆分键，Covering列包含主表拆分键、主键以及指定的列），若存在支持情况冲突情况，不支持的优先级高于支持。

- 下表汇总了使用ALTER TABLE语句变更索引的支持情况。

语句	是否支持
ALTER TABLE ADD PRIMARY KEY	支持
ALTER TABLE ADD [UNIQUE/FULLTEXT/SPATIAL/FOREIGN] KEY	支持，您可以同时在主表和索引表上添加局部索引，索引名称不可与GSI重复。

语句	是否支持
ALTER TABLE ALTER INDEX index_name {VISIBLE | INVISIBLE}	支持，仅在主表执行（禁止变更GSI状态）。
ALTER TABLE {DISABLE | ENABLE} KEYS	支持，仅在主表执行（禁止变更GSI状态）。
ALTER TABLE DROP PRIMARY KEY	禁止
ALTER TABLE DROP INDEX	仅支持删除普通索引或全局二级索引。
ALTER TABLE DROP FOREIGN KEY fk_symbol	支持，仅在主表执行。
ALTER TABLE RENAME INDEX	禁止

> **说明** 考虑到全局二级索引的稳定性和性能情况，目前禁止直接使用ALTER TABLE RENAME INDEX命令重命名全局二级索引。如需修改全局二级索引名，您可以先使用DROP INDEX删除全局二级索引，再重新创建使用新名称的二级索引，或提交工单联系技术支持进行修改。

变更索引表时的注意事项

- 不支持在索引表上执行DDL、DML语句。
- 不支持带有NODE HINT的DML语句更新主表、索引表。

使用其他DDL时的注意事项

语句	是否支持
DROP TABLE	支持
DROP INDEX	支持
TRUNCATE TABLE	不支持
RENAME TABLE	不支持
ALTER TABLE RENAME	不支持

> **说明**
>
> - 考虑主表与索引表的数据一致性，目前禁止执行TRUNCATE TABLE语句 。如需清空主表与索引表数据，您可以使用DELETE语句删除对应的数据，或提交工单联系技术支持进行修改。
> - 考虑到全局二级索引的稳定性和性能情况，目前禁止直接使用RENAME TABLE或ALTER TABLE RENAME命令重命名全局二级索引。如需修改全局二级索引名，您可以先使用DROP INDEX删除全局二级索引，修改表名后再重新创建新的二级索引，或提交工单联系技术支持进行修改。

使用DML语句时的注意事项

- 不支持在索引表上执行DML语句。
- 在主表上执行DML语句的限制，请参见全局二级索引对DML的限制。

21.访问控制

21.1. 注册和登录阿里云账号

本文介绍如何注册和登录阿里云账号。

注册阿里云账号

1. 您可以通过如下两种方式中的任意一种进入阿里云账号注册页面：
   - 进入阿里云官网，单击右上角的**免费注册**。

   - 直接访问注册页面。
2. 在**欢迎注册阿里云**页面填入您的账号信息并同意隐私条款、服务协议等，完成注册。

登录阿里云账号

阿里云账号的登录入口与子账号不同。

- 阿里云账号的登录入口。

- 子账号的登录入口。

21.2. RAM在PolarDB-X中的应用

本文档主要介绍阿里云的访问控制服务RAM的基本概念以及RAM在PolarDB-X中的应用场景。

背景信息

RAM是阿里云提供的用户身份管理与访问控制服务。使用RAM可以创建、管理不同账号（比如员工、系统或应用程序），并控制这些账号对云账户名下资源的操作权限。当企业或组织存在多用户协同操作资源时，RAM可以避免云账号密钥大范围传播，按需分配最小权限，从而降低企业信息安全管理风险，详情请参见什么是访问控制。

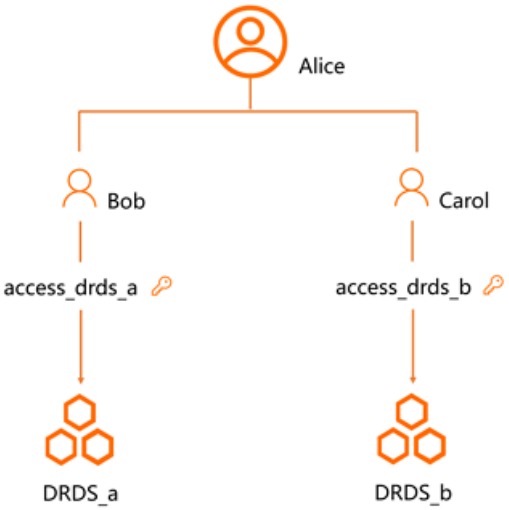
使用RAM服务前需要先了解基本概念，如云账户、RAM用户、身份凭证和角色等，了解这些基本概念能够帮助您快速上手使用RAM，更多关于RAM的基本概念，请参见基本概念。

RAM在PolarDB-X中的应用场景示例

假设某阿里云用户Alice拥有两个PolarDB-X 1.0计算资源实例（DRDS_a 和DRDS_b）。Alice对这两个实例都拥有完全的权限。

- 为了避免阿里云账号的AccessKey泄露导致安全风险，Alice使用RAM创建了两个子账号Bob和Carol。
- Alice建立了两个授权策略，access_drds_a和access_drds_b，分别表示DRDS_a和DRDS_b的读写权限。
- Alice在控制台上分别对Bob和Carol进行授权，使Bob对DRDS_a拥有读写权限，Carol对DRDS_b拥有读写权限。

Bob和Carol都拥有独立的AccessKey，若某一方出现AccessKey泄露情况也只会影响其中一个PolarDB-X 1.0计算资源实例，同时Alice也可以及时在控制台取消泄露用户的权限。



21.3. 激活PolarDB-X访问RDS服务授权

PolarDB-X的部分操作会调用RDS的OpenAPI，因此在使用RAM之前，需要先激活PolarDB-X访问RDS服务的授权，创建一个供PolarDB-X访问RDS的RAM服务角色。本文将介绍如何通过控制台和OpenAPI激活授权。

在控制台上激活RAM授权

1. 登录云原生分布式数据库控制台。
2. 在左侧导航栏，单击**资源授权**。
3. 在**云资源授权管理**页面，单击**激活授权**。

4. 在弹出的对话框中单击**同意授权**。

> 警告 激活授权的同时，RAM控制台上会默认创建一个名称为AliyunDRDSDefaultRole的RAM角色供PolarDB-X访问该账户下RDS资源，同时授权PolarDB-X访问RDS的OpenAPI，请勿随意删除该RAM角色。

通过RAM的OpenAPI激活RAM授权

若暂时无法访问控制台，您也可以通过调用相关OpenAPI创建RAM服务角色，角色创建完成即意味着PolarDB-X访问RDS资源的授权也被激活。

- 创建角色（假设要创建的RAM角色名是Jack）。
  - 调用CreateRole接口并输入以下参数创建RAM角色：

参数名	说明
Action	CreateRole
RoleName	Jack
AssumeRolePolicyDocument	信任策略，格式如下所示。

```
{
 "Statement": [
  {
   "Action": "sts:AssumeRole",
   "Effect": "Allow",
   "Principal": {
    "Service": [
     "drds.aliyuncs.com"
    ]
   }
  }
 ],
 "Version": "1"
}
```

  - 通过Java SDK创建RAM角色，Demo如下：

```
// 创建角色
String rolePolicyDoc = "{\"Statement\":[{\"Action\": \"sts:AssumeRole\", \"Effect\": \"Allow\",\"Principal
\":{\"Service\":[\"drds.aliyuncs.com\"]}}],\"Version\": \"1\"}";
String roleName = "Jack";
CreateRoleRequest request = new CreateRoleRequest();
request.setAssumeRolePolicyDocument(rolePolicyDoc);
request.setAssumeRolePolicyDocument(rolePolicyDoc);
client.getAcsResponse(request);
// 为角色授予策略
AttachPolicyToRoleRequest attachRequest = new AttachPolicyToRoleRequest();
attachRequest.setPolicyType("System");
attachRequest.setPolicyName("AliyunDRDSRolePolicy");
attachRequest.setRoleName("AliyunDRDSDefaultRole");
client.getAcsResponse(attachRequest);
```

- 查验角色是否创建成功（假设创建的RAM角色为Jack）。
  - 通过GetRole接口并输入以下参数查看角色详情：

参数名	说明
Action	CreateRole
RoleName	Jack

  - 通过Java SDK查看RAM角色详情，Demo如下：

```
String roleName = "AliyunDRDSDefaultRole";
GetRoleRequest request = new GetRoleRequest();
request.setRoleName(roleName);
GetRoleResponse resp = client.getAcsResponse(request);
GetRoleResponse.Role role = resp.getRole();
```

21.4. 通过自定义策略授权RAM用户管理PolarDB-XDRDS

如果RAM提供的系统策略无法满足您的业务需求，您可以通过创建自定义策略对PolarDB-X进行精细化权限管理。

前提条件

- 使用RAM服务前，请确保已经激活PolarDB-X对RDS的访问授权，详情请参见激活授权。
- RAM子帐户删除数据库与删除只读账户前，需开启MFA多因素认证，详情请参见为阿里云账号启用多因素认证。

背景信息

- 权限策略是用语法结构描述的一组权限的集合，可以精确地描述被授权的资源集、操作集以及授权条件，详细的语言规范请参见权限策略语法和结构。
- 使用自定义策略对PolarDB-X进行精细化权限管理前，请先了解的权限定义，详情请参见资源授权。

使用限制

RAM子帐户没有修改PolarDB-X数据库密码的权限。

步骤一：创建自定义权限策略

1. 云账号登录RAM控制台。
2. 在左侧导航栏，单击**权限管理 > 权限策略管理**。
3. 单击**创建权限策略**。
4. 在弹出的对话框中，配置自定义策略信息。

配置	说明
策略名称	建议填入具备业务意义的名称以便后续识别。
备注（可选）	填入该策略的备注信息。
配置模式	PolarDB-X仅支持**脚本配置**，此处选择为**脚本配置**。
策略内容	下拉选择系统已有的策略，将其导入至本策略中。 **说明** 本案例介绍自定义策略，此选项无需配置。
权限策略框	填入具体的权限策略信息，表格下方为您列出了一些常见的自定义权限策略供您参考。 **说明** ○ 权限策略是用语法结构描述的一组权限的集合，可以精确地描述被授权的资源集、操作集以及授权条件，详细的语言规范请参见权限策略语法和结构。 ○ 当前支持资源（Resource）和操作（Action）级别的授权。

常见的自定义权限策略：

> 说明
> - 请将下述代码中的 `1234` 替换成RAM子帐号所对应的主帐号的uid。
> - 请将下述代码中 `目标实例ID` 的替换为真实的PolarDB-X实例ID。

- 允许目标RAM子帐户操作PolarDB-X控制台。

```
{
 "Version": "1",
 "Statement": [
   {
     "Action": "drds:*",
    "Resource": "acs:drds:*:1234:instance/*",
    "Effect": "Allow"
   },
   {
     "Action": "ram:PassRole",
    "Resource": "*",
    "Effect": "Allow"
   }
 ]
}
```

- 仅允许目标RAM子账号访问华东1（杭州）地域下的所有PolarDB-X实例。

```
{
 "Version": "1",
 "Statement": [
   {
    "Action": "drds:*",
    "Resource": "acs:drds:cn-hangzhou:1234:instance/*",
    "Effect": "Allow"
   },
   {
    "Action": "ram:PassRole",
    "Resource": "*",
    "Effect": "Allow"
   }
 ]
}
```

- 不允许目标RAM子账号访问特定实例。

```
{
 "Version": "1",
 "Statement": [
   {
     "Action": "drds:*",
     "Resource": "acs:drds:*:1234:instance/*",
     "Effect": "Allow"
   },
   {
     "Action": "drds:*",
     "Resource": [
       "acs:drds:*:1234:instance/目标实例ID",
     ],
     "Effect": "Deny"
   },
   {
     "Action": "ram:PassRole",
     "Resource": "*",
     "Effect": "Allow"
   }
 ]
}
```

说明 被授予该策略的RAM子帐户可以访问主账号下除 目标实例ID 之外的所有PolarDB-X实例。

5. 单击**确定**。

步骤二：为RAM用户授权自定义策略

1. 登录RAM控制台。
2. 在左侧导航栏单击**授权管理 > 权限策略管理**。
3. 在**权限策略管理**页，找到目标自定义权限策略，单击权限策略名称。

4. 在目标权限策略的**基本信息**页，单击**引用记录**页签。
5. 单击**新增授权**，在弹出的对话框中，设置以下参数。

参数	说明
授权范围	您可以选择**授权范围**为**云账号全部资源**或**指定资源组**。
被授权主体	您可以在搜索框内输入目标RAM用户、用户组或RAM角色名称进行模糊搜索，并在搜索结果中选择目标RAM用户。

参数	说明
选择权限	目标自定义权限策略默认已加入右侧**已选择**框中，若您还需为目标RAM用户添加其他权限，可在左边待选框中选中对应的权限策略名称加入右侧**已选择**框。 **说明** 每次最多添加5条策略，如需添加更多策略，请分多次进行。

6. 单击**确定**。

说明 更多为RAM用户或用户组授权的方式，请参见为RAM用户授权和为用户组授权。

21.5. RAM资源授权

本文汇总了PolarDB-X支持的RAM资源授权规则以及已经为PolarDB-X开通了RAM服务的地域。

支持的资源授权规则

说明

- 您需要将表格中的 `$regionId` 根据目标实例所在地域替换成支持RAM的区域中对应的regionId。
- 您需要将表格中 `$` 开头的参数根据业务需求自行替换成真实参数。

Action	鉴权规则	描述
CreateDrdsInstance	acs:drds:$regionid:$accountid:instance/*	创建实例
DescribeDrdsInstanceList	acs:drds:$regionid:$accountid:instance/*	查看实例列表
UpgradeDrdsInstance	acs:drds:$regionid:$accountid:instance/$instanceid	实例变配
RemoveDRDSInstance	acs:drds:$regionid:$accountid:instance/$instanceid	释放实例
DescribeDrdsInstance	acs:drds:$regionid:$accountid:instance/$instanceid	查看实例详情
VersionChanage	acs:drds:$regionid:$accountid:instance/$instanceid	升级或回滚PolarDB-X实例版本
CreateInternetAddress	acs:drds:$regionid:$accountid:instance/$instanceid	创建PolarDB-X实例的公网地址
ReleaseInternetAddress	acs:drds:$regionid:$accountid:instance/$instanceid	释放PolarDB-X实例的公网地址

Action	鉴权规则	描述
CreateDrdsDB	acs:drds:$regionid:$accountid:instance/$instanceid/db/*	创建PolarDB-X数据库
DescribeDrdsDbList	acs:drds:$regionid:$accountid:instance/$instanceid/db/*	查看PolarDB-X实例的数据库列表
DescribeDrdsDb	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	查看PolarDB-X数据库详情
DeleteDrdsDb	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	删除PolarDB-X数据库
ModifyReadWriteWeight	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	修改读策略
DescribeLogicTableList	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	查看PolarDB-X数据库中数据表列表
ExecuteDDL	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	在PolarDB-X控制台执行DDL语句
ModifyDrdsIpWhiteList	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	修改PolarDB-X数据库的IP白名单
DrdsDataImport	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	数据导入
DrdsSmoothExpand	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	平滑扩容
CreateReadOnlyAccount	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	创建只读账户
ModifyReadOnlyAccountPassword	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	修改只读账户密码
RemoveReadOnlyAccount	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	删除只读账户

Action	鉴权规则	描述
DescribeAlarmContacts	acs:drds:$regionid:$accountid:contacts/*	查看报警联系人列表
AddAlarmContacts	acs:drds:$regionid:$accountid:contacts/*	添加报警联系人
ModifyAlarmContacts	acs:drds:$regionid:$accountid:contacts/*	修改报警联系人
RemoveAlarmContacts	acs:drds:$regionid:$accountid:contacts/*	删除报警联系人
DescribeAlarmGroup	acs:drds:$regionid:$accountid:contacts/*	查看报警联系人分组列表
AddAlarmGroup	acs:drds:$regionid:$accountid:contacts/*	添加报警联系人分组
ModifyAlarmGroup	acs:drds:$regionid:$accountid:contacts/*	修改报警联系人分组
RemoveAlarmGroup	acs:drds:$regionid:$accountid:contacts/*	删除报警联系人分组
DescribeInstanceMonitor	acs:drds:$regionid:$accountid:instance/$instanceid	查看实例监控信息
DescribeAlarmRule	acs:drds:$regionid:$accountid:instance/$instanceid	查看报警规则列表
CreateAlarmRule	acs:drds:$regionid:$accountid:instance/$instanceid	创建报警规则 **说明** 创建报警规则权限依赖查询联系人分组权限。
ModifyAlarmRule	acs:drds:$regionid:$accountid:instance/$instanceid	修改报警规则 **说明** 修改报警规则权限依赖查询联系人分组权限。
RemoveAlarmRule	acs:drds:$regionid:$accountid:instance/$instanceid	删除报警规则
DescribeAlarmHistory	acs:drds:$regionid:$accountid:instance/$instanceid	查看报警历史
DescribeSlowSql	acs:drds:$regionid:$accountid:instance/$instanceid	查看PolarDB-X慢SQL

Action	鉴权规则	描述
DrdsShardTool	acs:drds:$regionid:$accountid:instance/$instanceid/db/$dbname	使用拆分变更工具

支持RAM的区域

目前PolarDB-X已经开通RAM服务的地域以及对应的regionId如下表所示。

regionId	区域名称
cn-hangzhou	华东1（杭州）
cn-shenzhen	华南1（深圳）
cn-shanghai	华东2（上海）
cn-qingdao	华北1（青岛）
cn-beijing	华北2（北京）