ALIBABA CLOUD

阿里云

云数据库 PolarDB

云数据库 PolarDB公共云合集

文档版本：20220216

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.动态与公告 08
1.1. 新功能发布记录 08
1.1.1. 控制台发布记录 08
1.1.2. 内核发布记录 20
1.1.3. 数据库代理发布记录 43
1.2. 产品公告 48
1.2.1. 数据库代理企业版发布说明 48
1.2.2. 多主架构灰度发布说明 50
1.2.3. 列存索引（IMCI）灰度发布说明 53
1.3. 安全公告 56
1.3.1. Apache Log4j2漏洞对PolarDB无影响 56
2.快速入门 57
3.数据迁移或同步 58
3.1. 数据迁移或同步方案概览 58
3.2. 【注意事项】MySQL 5.7迁移到PolarDB MySQL引擎8.0版本 59
3.3. 从RDS迁移至PolarDB 60
3.3.1. 一键升级RDS MySQL至PolarDB MySQL引擎 60
3.3.2. 包年包月RDS迁移至PolarDB后申请转单优惠退款 69
3.3.3. 一键克隆RDS MySQL至PolarDB MySQL引擎 69
3.3.4. RDS MySQL迁移至PolarDB MySQL 73
3.4. PolarDB间的数据迁移 81
3.4.1. PolarDB MySQL间迁移 81
3.5. 从其它数据库迁移至PolarDB 90
3.5.1. 自建MySQL迁移至PolarDB MySQL 90
3.5.2. 从自建MySQL迁移至PolarDB MySQL引擎（mysqldump工具... 100
3.5.3. 从Amazon Aurora MySQL迁移至PolarDB MySQL 102

3.6. 从PolarDB迁移至其它数据库 110
3.6.1. PolarDB MySQL迁移至RDS MySQL 110
3.7. PolarDB间的数据同步 118
3.7.1. PolarDB MySQL集群间的单向同步 118
3.7.2. PolarDB MySQL集群间的双向同步 124
3.8. PolarDB与其它数据库的数据同步 132
3.8.1. RDS MySQL同步至PolarDB MySQL集群 132
3.8.2. PolarDB MySQL同步至RDS MySQL 140
3.8.3. 从PolarDB MySQL同步至云原生数据仓库AnalyticDB MySQL 147
3.8.4. 从PolarDB MySQL同步至云原生数据仓库AnalyticDB Postgr... 154
3.8.5. 从PolarDB MySQL同步至Datahub 160
3.8.6. 从PolarDB MySQL同步到Kafka 166
3.8.7. 从ECS上的自建MySQL同步至PolarDB MySQL 172
3.8.8. 从通过专线、VPN网关或智能接入网关接入的自建MySQL同步至.. 178
4.内核功能 184
4.1. 内核说明 184
4.1.1. 兼容性说明 184
4.1.2. 内核发布记录 184
4.1.3. 内核版本说明 207
4.1.4. 内核高级功能 207
4.2. DDL优化 209
4.2.1. 秒级加字段 209
4.2.2. 并行DDL 210
4.2.3. DDL物理复制优化 214
4.2.4. 并行元数据锁同步 219
4.2.5. 防止只读节点上长事务阻塞DDL操作 221
4.2.6. 查看DDL执行状态和MDL锁状态 223
4.3. 大查询优化（Parallel Query） 226

4.3.1. 并行查询（Parallel Query） 227
4.3.2. 并行执行EXPLAIN 232
4.3.3. 并行度控制策略 236
4.3.4. Hash Join的并行执行 237
4.3.5. Semi-Join的并行执行 238
4.3.6. ROLLUP性能增强 244
4.3.7. 通过Hint控制 246
4.3.8. 并行查询使用限制与串行执行结果兼容问题 249
4.3.9. 并行查询使用示例 250
4.4. 高并发优化 264
4.4.1. Statement Concurrency Control 264
4.4.2. Inventory Hint 269
4.4.3. Statement Queue 271
4.4.4. 热点行优化 276
4.4.5. Thread Pool 281
4.5. 性能监控 289
4.5.1. Performance Agent 289
4.6. 其它 296
4.6.1. Fast Query Cache 296
4.6.2. Statement Outline 303
4.6.3. 表回收站 309
4.6.4. Returning 312
5.SDK下载 316
6.常见问题 317
7.视频专区 325
7.1. 全球数据库网络GDN 325
7.2. 历史库 325
7.3. Instant DDL和Parallel DDL 325

7.4. 快速恢复 325
7.5. 并行查询和Hash Join的并行执行 325
8.服务等级协议 326

1.动态与公告

1.1. 新功能发布记录

1.1.1. 控制台发布记录

本章节介绍了云原生关系型数据库PolarDB的产品功能动态和对应的文档。

2021年12月

功能名称	功能描述	发布时间	相关文档
数据库代理升级为数据库代理企业版	数据库代理企业版提供两种版本：**企业通用版**和**企业独享版**。 • **企业通用版**：配套集群子系列的通用规格，它可以共享CPU物理资源，可根据业务负载，提供智能秒级资源弹性扩展能力。 • **企业独享版**：配套集群子系列的独享规格，它可以独占CPU物理资源，具有更好的性能稳定性。 相对于数据库代理历史版本（免费版），数据库代理企业版将支持多主架构、计算节点秒级弹性扩展、Proxy限流保护等更多高级功能。 **说明** 数据库代理企业版暂定于2022年4月1日起付费使用，当前您可以免费使用。	2021-12-09	数据库代理企业版概述
新增产品系列：历史库集群版	历史库集群版在历史库单节点版的基础上，基于共享存储实现了一写多读，集群中有一个主节点（可读可写）和至少一个只读节点。历史库集群版多节点架构不仅继承了历史库单节点版的优势，还可用于保障集群的高可用，当系统发生故障时，可读写的主节点和只读节点之间会自动进行故障切换（Failover），保证了服务可用性不低于99.99%。 存量的历史库单节点版可升级至历史库集群版。	2021-12-01	历史库

2021年11月

功能名称	功能描述	发布时间	相关文档
新增闪回查询功能	通过闪回查询（Flashback Query）功能，您可以高效查询实例、数据库、数据表在过去某个时间点的信息。	2021-11-11	闪回查询
包年包月的集群支持自动变配	包年包月的PolarDB MySQL引擎集群支持自动扩容和自动回缩。	2021-11-05	结合数据库自治服务DAS自动变配和PolarDB MySQL引擎集群版自动变配

2021年10月

功能名称	功能描述	发布时间	相关文档
参数模板中新增多个Thread Pool控制参数	PolarDB MySQL引擎8.0版本自8.0.1.1.9起新增**bypass_thread_pool_ips**、**bypass_thread_pool_check_ignore_proxy**等多个Thread Pool控制参数。	2021-10-25	Thread Pool

2021年08月

功能名称	功能描述	发布时间	相关文档
新增**polar_replica_work_on_nonblock_mdl_mode**参数	PolarDB支持**polar_replica_work_on_nonblock_mdl_mode**参数，开启该参数后，可以防止PolarDB只读节点上的长事务阻塞主节点上的DDL操作。	2021-08-26	防止只读节点上长事务阻塞DDL操作
新增计划内事件功能	PolarDB计划内的运维事件（例如数据库软件升级、硬件维护与升级）除了会通过短信、语音、邮件或站内信通知您，还会在控制台上进行通知。您可以在计划内事件中，查看具体的事件类型、任务ID、集群名称、切换时间等，也可以手动修改切换时间。	2021-08-06	查看并管理计划内事件

2021年07月

功能名称	功能描述	发布时间	相关文档
GDN网络中的集群规格变更	GDN网络中的集群不再支持2C规格。	2021-07-02	计算节点规格
MySQL 5.7新增库表恢复功能	PolarDB MySQL引擎5.7版本自5.7.1.0.8起支持按备份集和时间点两种库表恢复方式。	2021-07-02	库表恢复方式1：从备份集恢复和库表恢复方式2：恢复到过去时间点

2021年06月

功能名称	功能描述	发布时间	相关文档
TDE加密支持新建表自动加密	PolarDB MySQL引擎8.0版本自8.0.1.1.15起支持所有新建的表将自动加密。	2021-06-15	高级选项

2021年05月

功能名称	功能描述	发布时间	相关文档
SSL配置支持SSL证书自动轮转	开启证书自动轮转后，在证书即将过期的10天内，PolarDB会在集群的可维护时间窗口内自动更新证书。	2021-05-28	开启证书自动轮换

2021年03月

功能名称	功能描述	发布时间	相关文档
新增参数模板功能，方便批量化管理数据库集群。	参数模板可以批量管理数据库集群的参数，您可以使用参数模板功能，快速应用模板到PolarDB集群上。	2021-03-09	使用参数模板
新增增强备份功能，提升数据恢复速度。	PolarDB支持增强备份功能，能够对最近24小时的备份进行增强保护，您可以根据业务选择开启每2、3或4小时进行一次增强备份。	2021-03-09	增强备份
计算节点支持通用规格	PolarDB MySQL引擎的集群版支持通用规格，能够使同一服务器上的不同集群，互相充分利用彼此空闲的计算资源（如CPU）。通过复用计算资源享受规模红利，性价比更高。	2021-03-02	计算节点规格

2021年02月

功能名称	功能描述	发布时间	相关文档
MySQL 5.7新增支持秒级加字段和Statement Queue机制	PolarDB MySQL引擎5.7版本自5.7.1.0.6版本起，支持秒级加字段和Statement Queue机制。您可以通过秒级加字段功能快速完成对大表的加列操作，或通过Statement Queue机制减少冲突开销、提高数据库性能。	2021-02-09	秒级加字段 Statement Queue
提升数据恢复速度	PolarDB集群备份和恢复功能均采用多线程并行处理，并通过其它技术创新，10分钟内即可完成从备份集（快照）恢复到一个新的集群。	2021-02-09	备份和恢复数据

2021年01月

功能名称	功能描述	发布时间	相关文档
历史库正式上线商用	PolarDB历史库使用了高数据压缩率的X-Engine引擎，最高可节省70%存储成本，非常适合对计算诉求不高但需要存储一些归档类数据（如钉钉消息等数据）的业务。PolarDB历史库高度兼容MySQL，最大支持200 TB存储容量。	2021-01-13	历史库
支持定时升配或增加节点的功能	在升级集群配置或增加节点的时候，您可以选择在未来24小时内的业务低谷期进行升级，减少升级闪断对业务的影响。	2021-01-12	手动变配 增加或删除节点

2020年12月

功能名称	功能描述	发布时间	相关文档
提高部分规格节点的存储容量上限	PolarDB MySQL引擎8核32 GB的存储上限提升至20 TB，16核128 GB、32核256 GB和64核512 GB规格的存储上限提升至100 TB。	2020-12-24	计算节点规格
PolarDB控制台支持展示任务进度条	您可以在PolarDB控制台右上角单击  图标查看备份恢复等任务的进度。	2020-12-03	无
MySQL 5.6支持库表恢复功能	PolarDB MySQL引擎5.6版本支持库表恢复功能。	2020-12-01	• 库表恢复方式1：从备份集恢复 • 库表恢复方式2：恢复到过去时间点
5.6支持透明数据加密（TDE）	PolarDB MySQL引擎5.6版本支持透明数据加密（TDE）功能。	2020-12-01	设置透明数据加密TDE

2020年11月

功能名称	功能描述	发布时间	相关文档
存储包新增支持抵扣一级备份费用	除支持抵扣PolarDB集群中存储空间用量，存储包现新增支持抵扣一级备份中超过免费额度的空间用量。	2020-11-27	存储包规则

2020年10月

功能名称	功能描述	发布时间	相关文档
MySQL 8.0.2开启公测	PolarDB MySQL引擎8.0.2版本开启公测。	2020-10-26	• 购买按量付费集群 • 购买包年包月集群 • 内核发布记录

2020年9月

功能名称	功能描述	发布时间	相关文档
新增节点规格64核512 GB	PolarDB支持新规格64核512 GB。	2020-09-25	计算节点规格
计算资源包	PolarDB MySQL引擎 支持计算包，推荐您将计算包配合DAS提供的自动扩、缩容服务一起使用，轻松应对业务量波动。	2020-09-25	购买计算包 自动扩、缩容
单节点普惠版集群系列	PolarDB MySQL引擎新推出超高性价比的单节点普惠版集群系列。	2020-09-25	概述

2020年8月

功能名称	功能描述	发布时间	相关文档
澳大利亚（悉尼）和华北1（青岛）开服	PolarDB MySQL引擎在澳大利亚（悉尼）和华北1（青岛）正式开服。	2020-08-21	• 购买按量付费集群 • 购买包年包月集群
MySQL 5.7正式上线商业	PolarDB MySQL引擎5.7版本正式商业化，原公测期间使用PolarDB MySQL引擎5.7版本的用户可购买用于生产环境。	2020-08-21	• 购买按量付费集群 • 购买包年包月集群
英国（伦敦）开服	PolarDB MySQL引擎在英国（伦敦）正式开服。	2020-08-15	• 购买按量付费集群 • 购买包年包月集群

2020年7月

功能名称	功能描述	发布时间	相关文档
新建集群时支持区分表名大小写、设置时区和删除集群时的数据保护策略	您可以创建PolarDB MySQL引擎集群时自定义是否区分表名大小写，以及设置集群所在时区和删除集群时的数据保护策略。	2020-07-29	• 购买按量付费集群 • 购买包年包月集群
变配页支持一次增加多个只读节点	您可以在变更配置时为PolarDB MySQL引擎集群同时增加多个只读节点，只需5分钟左右时间即可完成多个节点的同时添加。	2020-07-28	增加或删除节点
MySQL 5.7开启公测	PolarDB MySQL引擎5.7版本开启公测。	2020-07-28	• 购买按量付费集群 • 购买包年包月集群
全球数据库支持全局读写分离	PolarDB MySQL引擎全球数据库网络中的所有集群地址均可读可写，读请求将被直接发送到本地从集群，而写请求则自动转发到中心地域的主集群。	2020-07-01	创建与删除全球数据库网络

2020年6月

功能名称	功能描述	发布时间	相关文档

功能名称	功能描述	发布时间	相关文档
带地址切换	您可以将原RDS MySQL实例升级为PolarDB MySQL引擎，并且保留原来的RDS地址，无需修改应用配置和代码。	2020-06-10	一键升级RDS MySQL至PolarDB MySQL引擎
服务协议等级	云原生关系型数据库PolarDB通过技术升级可用性得到了大幅提升，SLA承诺从99.95%提升至99.99%。	2020-06-02	服务等级协议

2020年5月

功能名称	功能描述	发布时间	相关文档
备份功能升级	PolarDB备份功能开放更多高级配置和能力，满足企业级数据备份的需求，进一步保障数据安全。	2020-05-12	备份与恢复数据
备份商业化	PolarDB将开始收取备份文件和日志文件的存储空间费用，同时会赠送一定额度的免费空间。	2020-05-12	备份和恢复费用说明

2020年4月

功能名称	功能描述	发布时间	相关文档
日本（东京）开服	PolarDB MySQL引擎在日本（东京）正式开服。	2020-04-30	• 购买按量付费集群 • 购买包年包月集群
西南1（成都）开服	PolarDB MySQL引擎在西南1（成都）正式开服。	2020-04-13	• 购买按量付费集群 • 购买包年包月集群

2020年3月

功能名称	功能描述	发布时间	相关文档
美国东部（弗吉尼亚）开服	PolarDB MySQL引擎5.6和8.0版本在美国东部（弗吉尼亚）正式开服。	2020-03-09	• 购买按量付费集群 • 购买包年包月集群
全局一致性	PolarDB MySQL引擎集群地址支持全局一致性，确保在主备复制延迟的情况下，100%可以查到最新数据。	2020-03-03	• 配置数据库代理 • 概述 • 一致性级别

2020年2月

功能名称	功能描述	发布时间	相关文档
SSL加密传输	PolarDB MySQL引擎主地址支持SSL加密传输，SSL在传输层对网络连接进行加密，能提升通信数据的安全性和完整性，但会同时增加网络连接响应时间。您可以按需开启SSL加密功能。	2020-02-11	设置SSL加密
主库不接受读	PolarDB MySQL引擎上线主库不接受读功能，在确保一致性的前提下，将查询SQL发送到只读节点，来降低主节点的负载，确保主节点稳定。	2020-02-07	配置数据库代理

2020年1月

功能名称	功能描述	发布时间	相关文档
新版报警规则	云监控上线新的报警规则“云数据库PolarDB MySQL引擎（新版）”，支持更多的报警指标以及更友好的通知体验。 为了更好的通知体验，建议您使用新版报警规则。	2020-01-09	创建报警规则

2019年11月

功能名称	功能描述	发布时间	相关文档
诊断与优化功能增加新地域支持	PolarDB MySQL引擎诊断与优化功能新增美国（硅谷）、德国（法兰克福）地域支持。 诊断与优化功能主要包括： • 一键诊断：该功能包括会话管理和实时性能展示，提供可视化的诊断结果供您查看。 • 慢SQL：慢SQL分析功能，能够查看慢日志趋势和统计信息，并且提供SQL建议和诊断分析。	2019-11-19	• 一键诊断 • 慢SQL
马来西亚（吉隆坡）开服	PolarDB MySQL引擎5.6和8.0版本在马来西亚（吉隆坡）正式开服。	2019-11-14	• 购买按量付费集群 • 购买包年包月集群
支持迁移可用区	PolarDB MySQL引擎支持更换主可用区，您可以通过该功能将数据库集群计算节点迁移到其他可用区，适用于灾难恢复或者让ECS就近访问的场景。	2019-11-13	多可用区部署

2019年10月

功能名称	功能描述	发布时间	相关文档
性能监控API	• 开放查询PolarDB集群的性能数据接口：DescribeDBClusterPerformance • 开放查询PolarDB集群节点的性能数据接口：DescribeDBNodePerformance	2019-10-21	• DescribeDBClusterPerformance • DescribeDBNodePerformance • 性能指标监控
存储包在新零售云（聚石塔）上线	存储包支持从100 GB到100 TB，共9个规格，容量不够可以随时升级。 • 中国内地通用：该存储包可被中国内地所有地域内的PolarDB集群使用，抵扣存储空间费用。 • 中国香港及海外通用：该存储包可被中国香港及海外所有地域内的PolarDB集群使用，抵扣存储空间费用。	2019-10-16	购买存储包
支持设置节点故障切换的优先级	PolarDB支持设置节点的故障切换优先级。每个节点的默认优先级为1，您可以修改为1-15的任意数值，修改优先级不影响数据库正常使用，实时生效。	2019-10-14	无
故障切换（主备切换）	PolarDB新增计划内的故障切换（主备切换）功能，您可以根据需要把主节点切换到指定的只读节点上。	2019-10-14	自动/手动主备切换

2019年9月

功能名称	功能描述	发布时间	相关文档
PolarDB MySQL引擎 8.0 正式上线商用	原生支持并行查询，特定场景下性能提升十倍。另外，还支持不锁表快速添加字段，大大降低添加字段对业务稳定性的影响，同时增强了NoSQL的部分功能，可以方便地使用JSON数据类型，在SQL方面增加了窗口函数等高级特性。	2019-09-12	并行查询（Parallel Query）

2019年8月

功能名称	功能描述	发布时间	相关文档
性能洞察	PolarDB推出性能洞察功能，以简单直观的方式帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。	2019-08-27	性能洞察

2019年7月

功能名称	功能描述	发布时间	相关文档

PolarDB API	• 从RDS迁移功能，开放共计3个接口：
	  ◦ 查询迁移状态：DescribeDBClusterMigration
	  ◦ 修改迁移任务：ModifyDBClusterMigration
	  ◦ 关闭迁移任务：CloseDBClusterMigration
	• 备份恢复功能，开放共计5个接口：
	  ◦ 创建备份：CreateBackup
	  ◦ 删除备份：DeleteBackup
	  ◦ 查看备份集：DescribeBackups
	  ◦ 配置备份策略：ModifyBackupPolicy
	  ◦ 查看备份策略：DescribeBackupPolicy
		2019-07-31	• DescribeDBClusterMigration
			• ModifyDBClusterMigration
			• CloseDBClusterMigration
			• CreateBackup
			• DeleteBackup
			• DescribeBackups
			• ModifyBackupPolicy
			• DescribeBackupPolicy

存储资源包	PolarDB推出了预付费形式的存储包。支持从100 GB到100 TB，共9个规格，容量不够可以随时升级。相比按量付费，包年包月的存储包有折扣，购买的容量越大，折扣力度就越大。	2019-07-30	购买存储包
数据库管理	表名长度限制与开源MySQL保持一致。 • 字母或数字表名长度限制从59个字符提升到64个字符。 • 汉字表名长度限制从11个字符提升到50个字符。	2019-07-12	使用限制
并行查询	PolarDB MySQL引擎8.0版本重磅推出并行查询（Parallel Query）框架，当您的查询数据量到达一定阈值，就会自动启动并行查询框架。并行查询利用多核CPU的并行处理能力，大幅提高查询效率。	2019-07-01	并行查询（Parallel Query）

2019年6月

功能名称	功能描述	发布时间	相关文档
批量购买	PolarDB批量购买功能发布，创建集群时支持选择集群数量，批量创建多个集群，最多可一次性购买50个集群。	2019-06-20	• 购买按量付费集群 • 购买包年包月集群

功能名称	功能描述	发布时间	相关文档
多可用区部署	PolarDB支持多可用区部署，满足机房级容灾要求。PolarDB是计算和存储分离架构，在购买时可以选择计算节点所在可用区（主可用区），底层数据会自动分布在多个可用区内。同时，PolarDB会在备可用区内预留足够的计算资源用于主可用区故障时进行容灾切换。您可以在控制台上集群的详情页看到数据分布的可用区。	2019-06-17	多可用区部署
一键升级RDS For MySQL到PolarDB	在创建PolarDB集群时，除了创建全新的PolarDB集群，阿里云还提供如下两种方式： • 从RDS克隆：从RDS备份文件恢复全量数据到PolarDB新集群上，新集群创建后，两者完全独立，互不影响。 • 从RDS迁移：从RDS备份文件恢复全量数据到PolarDB新集群，然后利用源RDS实例的Binlog单向同步数据到PolarDB集群。	2019-06-12	• 一键克隆RDS MySQL至PolarDB MySQL引擎 • 一键升级RDS MySQL至PolarDB MySQL引擎

2019年5月

功能名称	功能描述	发布时间	相关文档
跨数据库查询	支持PolarDB MySQL引擎与其它数据库之间进行跨数据库查询。	2019-05-30	跨数据库查询
PolarDB API	• 开放新增节点接口：CreateDBNodes • 开放删除节点接口：DeleteDBNodes • 开放修改节点规格配置接口：ModifyDBNodeClass	2019-05-10	• CreateDBNodes • DeleteDBNodes • ModifyDBNodeClass
自定义集群地址	PolarDB MySQL引擎推出自定义集群地址，可以指定哪些节点提供服务，同时还可以创建多个集群地址，分别调度到不同的节点，以达到资源隔离或独占的效果。	2019-05-06	配置数据库代理

2019年4月

功能名称	功能描述	发布时间	相关文档
新增IOPS等8项性能监控指标	PolarDB新增对IOPS、I/O吞吐量、Redo日志大小、Binlog日志大小的监控指标，便于排查性能和空间问题。	2019-04-11	查看性能监控指标

2019年3月

功能名称	功能描述	发布时间	相关文档
标签管理	PolarDB支持用户对PolarDB资源进行标签化管理，支持对PolarDB资源添加标签和删除标签。该功能目前仅支持OpenAPI、SDK。	2019-03-12	TagResources

2019年1月

功能名称	功能描述	发布时间	相关文档
支持开启Binlog	PolarDB支持开启二进制日志Binlog，可以更加方便地与MySQL生态融合。通过修改集群参数（ `loose_polar_log_bin = ON_WITH_GTID` ）的方式来手动开启，同时Binlog的保留时长受参数loose_expire_logs_hours控制。一般情况下，开启后会有大概30%~40%的写性能损耗，查询性能不受影响。	2019-01-15	开启Binlog

2018年

功能名称	功能描述	发布时间	相关文档
OpenAPI支持	开放OpenAPI，创建集群、删除集群、管理连接地址、管理账号和数据库、管理参数等功能可以通过API操作。您也可以使用阿里云命令行工具CLI或 Web端命令行工具CloudShell来管理您的PolarDB集群。	2018-12-28	API概览
SQL洞察	通过采集、分析数据库原始SQL日志，帮您洞察潜在的安全和性能风险。 • 增强了搜索功能，提供按照DB、User、客户端IP、线程ID、执行时长、扫描行数等多维度检索能力，并支持搜索结果的导出和下载。 • 新增了分析能力，可以对指定时间段的SQL日志进行可视化交互式分析，找出异常SQL，定位性能问题。	2018-12-21	SQL洞察
默认提供集群地址并支持会话读一致性	PolarDB集群默认提供一个集群地址，通过连接该地址，即可连接到所有节点。带有读写分离功能，写请求会自动发往主节点，读请求会自动根据各节点的负载发往主节点或只读节点。同时为了确保延迟情况下数据查询一致性，PolarDB提供了会话读一致性保障。	2018-12-20	• 申请集群地址和主地址 • 一致性级别 • 概述
监控报警	云原生关系型数据库PolarDB与云监控打通，支持监控报警等功能。您可以在云监控配置对PolarDB数据库的CPU、内存、连接数、网络带宽等指标进行监控并设置报警。	2018-10-26	查看性能监控指标

功能名称	功能描述	发布时间	相关文档
账号和数据库管理	支持在控制台创建、删除数据库；支持创建、删除账号；支持管理账号权限、重置账号等操作。	2018-10-19	• 管理数据库 • 管理数据库账号
自定义时间点恢复数据	通过创建一个新集群的方式，支持用户自定义时间点来恢复数据。	2018-08-13	库表恢复方式2：恢复到过去时间点
Snapshot备份和恢复	支持系统自动创建和用户手动创建Snapshot备份，并通过Snapshot备份创建新集群用于恢复数据。 • 支持自动备份时间设置。 • 支持手动Snapshot备份，以及从备份恢复。	2018-08-13	备份与恢复数据
手动重启节点	支持PolarDB集群主节点、只读节点等的手动重启操作。	2018-07-31	重启节点
全量迁移	PolarDB接入DTS，支持从RDS（MySQL）全量迁移到PolarDB。	2018-07-13	从RDS MySQL迁移至PolarDB MySQL
读写分离	PolarDB支持通过读写分离数据库连接地址，访问PolarDB集群的能力。提供读写请求自动路由，以及并发请求的自适应负载均衡的能力。	2018-07-07	概述
按量计费售卖模式	PolarDB增加全规格按量计费售卖，包括新购、升级、降级、增加只读节点、减少只读节点以及按量转包年包月功能。	2018-06-25	购买方式12：按量付费
在线规格变配	PolarDB增加计算规格在线升级能力。	2018-05-25	• 手动变配 • 结合数据库自治服务DAS自动变配 • PolarDB MySQL引擎集群版自动变配
在线增减只读节点	PolarDB支持在线增加只读节点或者减少只读节点。	2018-05-25	增加或删除节点
PolarDB全规格放开售卖	云原生关系型数据库PolarDB支持更多产品规格售卖，在已有16核128 GB和32核220 GB的基础上，增加支持更多规格售卖，如：2核4 GB、2核16 GB、4核32 GB和8核64 GB。	2018-05-24	计算节点规格

功能名称	功能描述	发布时间	相关文档
PolarDB商用上线	云原生关系型数据库PolarDB是阿里云自研的能满足高吞吐在线事务处理的关系型云数据库。既融合了商业数据库稳定可靠、高性能、可扩展的特征，又具有开源数据库简洁开放的优势。 • SQL：100%兼容MySQL 5.6 。 • 性能：100万QPS OLTP处理能力，10ms内数据延迟，10s内高可用切换时间。 • 存储：支持高达100TB的数据存储容量。Serverless 按量数据收费。 • 备份：支持Snapshot秒级备份。 • 弹性：5分钟内规格升级或增加只读节点。	2018-03-27	什么是PolarDB

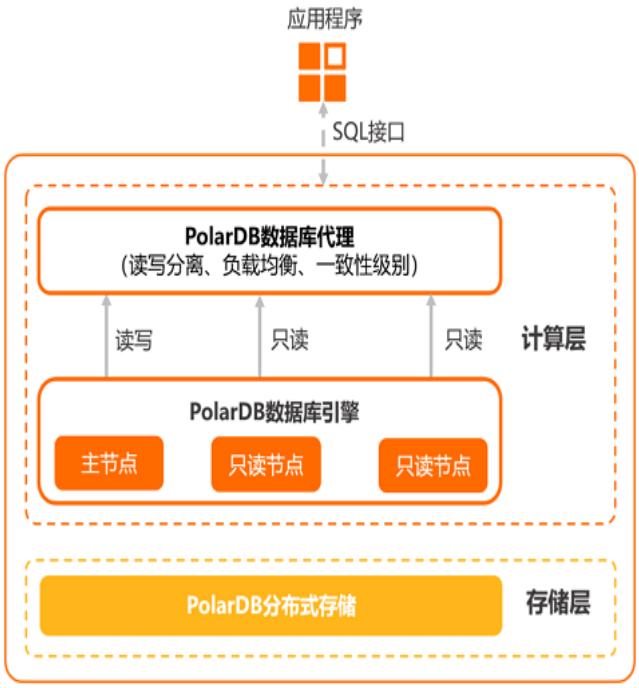
2017年

功能名称	功能描述	发布时间	相关文档
PolarDB（公测）发布	PolarDB是与MySQL100%兼容的，提供高性能在线事务处理能力的全托管数据库。它融合了商业数据库稳定可靠、高可用、可扩展的优势，又具有开源数据库简洁开放、成本低廉的特征。 • 100%兼容MySQL。 • 高性能（高达100万QPS）。 • 大容量（上百TB级数据量支持）、高可靠（多数据副本）。 • 高可用、可扩展（具备计算规格Scale Up/Down，以及只读节点Scale Out/In弹性能力）。 • 支持Snapshot备份、恢复。	2017-09-26	什么是PolarDB

1.1.2. 内核发布记录

本文将详细介绍PolarDB MySQL引擎内核模块的更新日志和全部的功能特性。

PolarDB数据库引擎在PolarDB架构中的位置如下图所示。



更多关于PolarDB产品架构及特点，请参见产品架构。

查询版本号

您可以通过如下方式查看集群的内核版本信息：

- 登录PolarDB控制台，在目标集群的**基本信息**页直接查看**内核版本**信息。

- 若在控制台上未看见**内核版本**信息，您可以通过 `show variables like "%polardb_version%";` 或 `show variables like '%rds_release_date%';` 命令查询具体的版本号。

> **说明**
> - PolarDB MySQL引擎5.6版本集群仅支持通过 `show variables like '%rds_release_date%';` 命令查询具体的版本号。
> - 关于PolarDB数据库引擎版本号的详细说明，请参见数据库引擎版本号说明。

PolarDB MySQL与社区MySQL版本的兼容关系

内核版本	完全兼容的MySQL小版本
8.0.2	8.0.18
8.0.1	8.0.13

内核版本	完全兼容的MySQL小版本
5.7	5.7.28
5.6	5.6.16

PolarDB MySQL引擎8.0版本发布日志

PolarDB MySQL引擎8.0支持如下版本：

- 8.0.2.1.4.1（发布日期：20211026）

类别	说明
问题修复	支持修改数据库缓存大小。

- 8.0.2.1.4（发布日期：20201201）

类别	说明
新增功能和性能优化	支持官方MySQL 8.0.20的Index Hints功能。
问题修复	○ 修复Hash Join并行的一个稳定性问题。 ○ 修复统计信息中关于每个索引key对应行数估算错误的问题。 ○ 修复并行扫描共享临时表时，处理空数据出现错误的问题。 ○ 修复官方REGEXP_REPLACE函数处理数据出现错误的问题。 ○ 修复官方对于复杂查询中包含子查询常量的崩溃问题。 ○ 修复filesort返回错误数据的问题。

- 8.0.2.1.3（发布日期：20201026）

类别	说明
新增功能和性能优化	○ 增强Hash Join的代价估算模型性能。 ○ 支持并行Hash Join，即Build和Probe阶段的并行。
问题修复	○ 修复并行执行计划中并行分区数量错误显示的问题。 ○ 修复并行子查询的一个异常崩溃问题。 ○ 修复在并行查询中使用RAND() 函数时，多个工作线程不能产生随机结果的问题。

- 8.0.2.1.2（发布日期：20200927）

类别	说明
新增功能和性能优化	PolarDB并行全面支持最新的火山模型迭代执行器。

- 8.0.2.1.1（发布日期：20200826）

类别	说明
新增功能和性能优化	优化集群的写性能。
问题修复	修复内存泄漏问题。

- 8.0.2.1.0（发布日期：20200722）

类别	说明
新增功能和性能优化	○ EXPLAIN支持并行计划的展示，支持通过FORMAT=TREE查看并行的执行计划树。 ○ 并行查询支持哈希连接，增强了无索引的JOIN性能，详情请参见Hash Join的并行执行。 ○ 支持Resource Manager功能，能够实现对CPU和内存资源的监控，详情请参见Resource Manager。 ○ 支持Performance Agent，能够实现PolarDB MySQL引擎集群中的节点内部各项性能数据的采集与统计，您可以通过直接查询内存表PERF_STATISTICS获取相关指标的性能数据，详情请参见Performance Agent。 ○ 支持热重启（Warm buffer pool），在Crash重启和升级时，Buffer Pool中的数据依然存在，大幅度加快重启速度，并保持重启后性能无衰减。

- 8.0.1.1.23（发布日期：20220120）

类别	说明
新增功能和性能优化	○ 支持Strict Consistency Cluster（RO节点强一致性读）功能。 ○ Statement Outline支持PrepareStatement。 ○ 支持热备节点功能，进一步优化高可用效率。
问题修复	○ 修复Fast Query Cache在RO节点获取MDL锁导致Redo日志同步阻塞的问题。 ○ 支持SELECT FOR UPDATE/SHARE WAIT N语法。 其中，N表示等待超时的秒数，针对单个行锁。如果一个查询需要锁定多行，不会对时间进行累计，仅对单行进行超时检测。等待时间超过N秒，则返回锁等待超时错误 `Lock wait timeout exceeded; try restarting transaction` 。

- 8.0.1.1.22（发布日期：20211222）

类别	说明

类别	说明
问题修复	○ 优化并行计划在分区表索引等值访问时的并行度估算精度。 ○ 修复特定场景下，优化器进行统计信息估算的代价耗时长的问题。 ○ 修复优化器针对部分GROUP BY语句未选择更优的索引范围路径的问题。 ○ 修复Standby节点通过HA切换为主节点后，创建新的Redo日志文件失败的问题。

- 8.0.1.1.21（发布日期：20211126）

类别	说明
问题修复	○ 修复group_concat_max_len参数取值较大时，出现算术溢出导致GROUP_CONCAT函数结果错误的问题。 ○ 修复备可用区的Standby节点，从故障中恢复后出现数据错误的问题。 ○ 修复备可用区的Standby节点切换为主节点（RW节点）时，出现的数据异常问题。 ○ 修复优化器在选定了多列组合索引过滤条件时，只能进行单列索引的过滤，不能进行组合索引过滤的问题。

- 8.0.1.1.19（发布日期：20210918）

类别	说明
新增功能和性能优化	○ 针对DDL操作增加了新的并发控制CCL规则。 ○ 增加参数restrict_on_limit_without_order，控制存在limit子句的并行查询中在没有order by子句的情况下，是否允许执行并行查询。
问题修复	○ 修复并行查询中的Group by隐式排序，在选择group by列的索引的时候，并发执行结果无序的问题。 ○ 修复在使用线程池（Thread Pool）时，审核日志中事务ID字段始终为0的错误。

- 8.0.1.1.18（发布日期：20210814）

类别	说明
新增功能和性能优化	数据库内核支持事务断点续传和只读节点读取未提交的事务。

类别	说明
问题修复	优化 `master_key_id_mutex` ，使得DDL操作可以并行执行。

- 8.0.1.1.17（发布日期：20210723）

类别	说明
新增功能和性能优化	○ 新增polar_replica_work_on_nonblock_mdl_mode参数。开启该参数时，只读节点上的RU/RC隔离级别的未提交事务将不再阻塞主节点上的DDL操作，同时只读节点上将不在保证表定义的事务特性。 ○ 优化了海量表场景下（如需要提供SaaS服务的场景）对统计信息的维护机制，大幅提升高并发下的查询表结构（desc table）和读写性能。
问题修复	○ 修复了只读节点在进行物理复制，高并发压力很大时，崩溃在m_recv_bits.is_set(fold)的问题。 ○ 修复了只读节点在高并发压力很大时，replay log崩溃的问题。

- 8.0.1.1.16（发布日期：20210624）

类别	说明
问题修复	在ACL Cache Lock请求发生等待的时候，在master error log中打印请求线程和最早的锁持有线程的信息来辅助诊断问题。

- 8.0.1.1.15（发布日期：20210525）

类别	说明
新增和优化功能	○ TDE支持对集群中MySQL的新建表自动加密。 ○ MySQL数据表支持utf8mb4_0900_bin字符集。
问题修复	○ 修复了instant add column后回滚Update操作时产生的记录过长，进而导致数据库崩溃的问题。 ○ 修复了避免mysql.slow_log、mysql.general_log被DDL语句强制使用innodb引擎造成混乱的问题。 ○ 修复了REGEXP函数的元数据信息错误导致结果集错误的问题。 ○ 修复了在虚拟列上回滚Update操作导致数据库崩溃的问题。 ○ 调整RO节点在首次注册到主节点时是否要立即触发checkpoint的策略。当LSN差值小于规定的阈值时，可不触发checkpoint。

- 8.0.1.1.14（发布日期：20210423）

类别	说明
新增功能和性能优化	○ 优化了海量表场景（如需要提供SaaS服务的场景）下对内部索引信息的维护，来提升只读节点的启动速度。 ○ 优化GDN同步连接，减少同步线程对CPU资源的消耗，提升了小规格（即8核以下）从集群同步Redo日志的速度。 ○ 优化了并行度控制参数中的AutoDop策略，以避免对PARALLEL HINT和 `force_parallel_mode` 的使用造成影响。
问题修复	○ 修复当RANGE查询范围较广时，由于 `records_in_range` 统计信息不准确，导致使用了错误索引的问题。 ○ 修复当进行按时间点恢复全量数据时，由于缓冲池过小导致Standby节点崩溃问题。 ○ 修复X-Engine引擎内部元信息内存占用过高的问题。 ○ 修复官方MySQL中2个关于ACL DDL的问题，避免由于ACL DDL操作带来的死锁导致集群不可用的问题。 ○ 修复当并行查询中包含了SQL_BUFFER_RESULT关键字，且使用了聚集函数但没有GROUP BY时，会返回错误结果集的问题。

- 8.0.1.1.13.2（发布日期：20210419）

类别	说明
问题修复	修复主备切换时，TDE加密表中的加密信息可能丢失导致解密失败的问题。

- 8.0.1.1.13.1（发布日期：20210408）

类别	说明
问题修复	修复并行查询中，由于未将Block Nested-Loop Join（BNL）算法中的常量过滤条件推到单表上，导致查询速度变慢的问题。

- 8.0.1.1.13（发布日期：20210330）

类别	说明
新增功能和性能优化	○ 只读节点支持polar_use_statement_mdl_on_replica参数，避免只读节点上的事务（RC隔离级别）堵塞主节点的DDL操作。同时，当只读节点上的读事务与主节点上的DDL并发时，读事务将会看到不同的表定义（例如在只读节点读事务的两条语句中间，主节点上存在ADD COLUMN操作，那么只读节点上的第二条语句将读到比第一条语句更多的列）。 ○ 移除索引等值查找路径中不必要的等值条件，便于在ORDER BY LIMIT场景下将Limit offset下推执行。 ○ 新增 `dbms_stats.gather_stats(timeout, workers)` 命令，支持通过事件调度或手动执行该命令来更新过时的直方图。 ○ `mysql.slow_log` 新增支持查看 `log_version` 、 `log_id` 、 `origin_start_time` 和 `rds_ext` 字段。

类别	说明
问题修复	○ 修复当对历史库的X-Engine表执行CHECK TABLE、COUNT(*)或DDL等命令时，无法终止查询的问题。 ○ 将KICKOUT修改为非保留关键字。 ○ 修复特定情况下，由于生成查询计划时评估扫描行数少于实际扫描行数，导致未充分进行并行查询的问题。

- 8.0.1.1.12.2（发布日期：20210312）

类别	说明
问题修复	○ 修复在打开 `session_track_temporary_tables` 系统变量的情况下，在存储过程中创建或删除临时表会导致集群不可用的问题。 ○ 引入官方MySQL 8.0.14的补丁，用于解决执行CREATE USER命令时，由于无法获取MySQL数据库系统表的元数据锁（MDL），导致CREATE USER语句被阻塞的问题。

- 8.0.1.1.12.1（发布日期：20210302）

类别	说明
新增功能和性能优化	优化在多表场景下持续导入数据时，PolarDB历史库X-Engine的写性能。
问题修复	修复并行查询中，由于Leader线程缺失互斥锁（Mutex）保护，导致与Worker线程上的元数据锁（MDL）状态可能不一致的问题。

- 8.0.1.1.12（发布日期：20210220）

类别	说明
新增功能和性能优化	○ 并行度控制策略新增auto_dop_low_degree_cost参数，用于设置并行查询的并行度选择策略，详情请参见并行度控制策略。 ○ 新增支持通过 `restore_table` 命令，快速恢复回收站内的表，详情请参见表回收站。 ○ 支持从只读节点上获取Binlog，详情请参见远程获取并解析PolarDB MySQL引擎Binlog日志。 ○ 支持在 `opt trace` 中打印 `in_memory` 等关键信息，便于在执行计划出现问题时，定位问题原因。
问题修复	○ 引入Port Percona补丁，用于解决高并发场景下，ACL CACHE 元数据锁冲突检测较慢的问题。 ○ 在 `optimizer_switch` 系统变量中新增 `preferred_ordering_index` 参数，用于修复某些场景下（如使用了LIMIT子句的ORDER BY或GROUP BY查询），由于使用了有序索引，导致没有选择最优计划的问题。 ○ 修复部分场景下SHOW PROCESSLIST结果不正确的问题。 ○ 修复8.0.1.1.10之前的版本在执行小版本升级后，由于未更新系统表 `information_schame.KEY_COLUMN_USAGE` 的定义，导致系统表访问较慢的问题。

- 8.0.1.1.11（发布日期：20210129）

类别	说明
新增功能和性能优化	○ 新增支持通过parallel_degree_policy参数来设置并行查询中并行度的配置策略，详情请参见并行度控制策略。 ○ 新增支持通过SET GLOBAL语句设置max_digest_length参数值来限制可识别语句的长度。 **说明** max_digest_length参数值修改后，客户端需要重新连接集群，新参数值才会生效。
问题修复	○ 修复主节点和只读节点权限不一致的问题。 ○ 修复主备切换后，只读节点无法连接到主节点的问题。 ○ 修复当特定执行计划失效时，SPM的处理逻辑不正确的问题。

- 8.0.1.1.10（发布日期：20210112）

类别	说明
新增功能和性能优化	○ 新增支持 `Group By` 的隐式排序功能，与在PolarDB MySQL引擎5.7版本上的用法一致。 ○ 新增当存在Blob字段时禁用并行查询的功能。 ○ 新增只读节点上自动更新语句级并发控制缓存信息的功能。 ○ 新增热点行优化功能，详情请参见热点行优化。 ○ 新增支持DDL物理复制优化，详情请参见DDL物理复制优化。 ○ 新增支持并行元数据锁同步，详情请参见并行元数据锁同步。 ○ 新增当计算下推时快速反向遍历的功能。 ○ 优化了文件系统，加快多表场景下表的打开速度。 ○ 缩短了多表场景下的主备切换时间，加速新主节点的恢复。

类别	说明
问题修复	○ 修复当只读节点升级为主节点时出现的系统表丢失的问题。 ○ 修复开启并行查询后，使用范围查询时会导致评估扫描行数过多的问题。 ○ 修复当字段类型为BIT时，聚合查询的结果为整型的问题。 ○ 修复使用枚举字段后，通过SELECT DISTINCT查询返回的结果不正确的问题。 ○ 修复使用EXISTS条件的并行查询结果出现异常的问题。 ○ 修复某些情况下只读节点重启失败的问题。 ○ 修复当只读节点执行DDL时，因外键关联表在打开表时把正在执行的DDL表也重新打开，导致数据字典中表信息异常的问题。 ○ 修复因主备切换后未正确设置节点重启标志，导致全文索引查询失败的问题。 ○ 修复因元数据锁（MDL）导致只读节点上日志应用线程被阻塞的问题。 ○ 修复因释放的内存被复用导致主备切换后，新的主节点不可用的问题。 ○ 修复因 `polar.info` 数据问题导致所有节点不可用的问题。 ○ 修复分区表自增列异常的问题。 ○ 修复主节点出现Redo log被覆盖写导致数据出错的问题。 ○ 修复当主节点等待元数据锁（MDL）时，主节点不可用的问题。 ○ 修复在使用透明数据加密（TDE）时的相关问题。 ○ 修复在执行Lock Table并开启表回收站功能时出现集群不可用的问题。 ○ 修复主节点在执行DDL时出现死锁的问题。 ○ 修复线程池和连接控制无法同时生效的问题。

- 8.0.1.1.9（发布日期：20201218）

类别	说明
新增功能和性能优化	取消将 `SPM` 和 `PLAN` 设置为关键字，避免出现因表名包含这两个词而无法操作的问题。

- 8.0.1.1.8（发布日期：20201209）

类别	说明
新增功能和性能优化	○ 执行计划管理器新增多计划模式。 ○ 新增系统变量rds_ap_threshold参数，来阻拦优化器评估扫描记录数太多的请求。 ○ 提升了主节点的脏页落盘效率。 ○ 新增Redo多分片写入机制。
问题修复	○ 修复并行查询执行过程中出现元数据锁（MDL） key空指针的问题。 ○ 修复当创建并线线程缓存时出现查询失败的问题。 ○ 修复并行查询MRR（Multi-Range Read）返回结果异常的问题。

- 8.0.1.1.7（发布日期：20201116）

类别	说明
新增功能和性能优化	○ 提升了JOIN查询等场景中，被驱动表并行扫描的效率。 ○ 新增支持在关闭Binglog情况下，依然能够清理残留的Binglog文件。 ○ 新增Slave节点物理复制断开自动检查重连机制，避免出现长时间的物理复制断开。 ○ 提升了主节点和只读节点间的切换效率。 ○ 支持快速启动包含大量表的集群，方便快速扫描表数据文件。
问题修复	○ 修复在获取 `trx->wait_lock` 的类型时出现集群崩溃的问题。 ○ 修复在打开多队列Simulated AIO时，AIO线程存在数量上限的问题。 ○ 修复当查询索引遇到初始化查询失败时，无法直接结束查询的问题。 ○ 修复Slave节点在SMO（Split Merge Operation）过程中，当前游标的Next Page指向了一个不存在的Page的问题。 ○ 修复只读节点会读取到已被主节点覆盖的日志信息的问题。 ○ 修复因Redo日志中时间戳间隔过大导致清理Redo文件失败的问题。 ○ 修复释放元数据锁（MDL）时未清除相关缓存中表缓存信息的问题。

- 8.0.1.1.6（发布日期：20200921）

类别	说明
新增功能和性能优化	○ 提升了SPM（SQL plan manager）和并行查询的兼容性。 ○ 提高了并行查询归并排序的效率。 ○ 支持删除操作的计算下推工作。 ○ 支持PolarDB Commit Timestamp（CTS）功能。
问题修复	○ 修复 `pq_optimize_switch` 描述不正确的问题。 ○ 修复子查询不能被稳定执行的问题。

- 8.0.1.1.5（发布日期：20200819）

类别	说明
新增功能和性能优化	○ 当只读节点和主节点建立复制关系时，新增主节点是否需要立即执行checkpoint策略的功能。 ○ 支持简单的范围查询和计算下推工作。 ○ PFS（Polar file system）文件系统新增pfs_remount功能，避免了因文件未关闭导致无法挂载PFS文件的问题。 ○ 解决了只读节点上由于Parse线程强制停顿造成的性能瓶颈问题，提升了物理复制过程中数据同步的效率。 ○ 优化了多连接场景下的Early Lock Release性能，多连接场景下的集群性能提升至原来的10倍。

类别	说明
问题修复	○ 修复只读节点在连接主节点失败后的不可用问题。 ○ 修复在使用全文索引并执行DDL查询语句后，会导致的只读节点在主备切换后不可用问题。 ○ 修复执行UNDO TRUNCATE命令后导致的无法进行purge binlog问题。 ○ 修复只读节点和主节点间统计信息不一致的问题。

- 8.0.1.1.4（发布日期：20200704）

类别	说明
新增功能和性能优化	○ 支持并行DDL，提升DDL执行效率。 ○ 支持Simulated AIO动态调整多队列的长度。 ○ 支持全文搜索（Full Text Search，FTS）缓存一致性。 ○ WHERE条件中支持含有聚集函数的子查询，且若子查询支持基于索引的扫描，那么该子查询还可以支持并行执行。 ○ 临时表和普通表一样支持进行lock mode检查。
问题修复	○ 修复当主节点降为备节点时，因部分DDL仍在复制中而导致的集群不可用问题。 ○ 修复因为开启线程池导致的性能下降问题。 ○ 修复purge binlog导致死锁的问题。 ○ 修复若干内存泄漏的问题。 ○ 修复若干在高可用场景中出现的问题。

- 8.0.1.1.3（发布日期：20200529）

类别	说明
新增功能和性能优化	○ 增强安全性（如密码管理）。 ○ 提升如下场景中并行查询（Parallel Query）的性能： ■ 增强GROUP BY、UNION、SELECT COUNT(*) FROM <table>场景中的并行查询性能。 ■ 并行子查询中，执行计划使用了共享InnoDB临时表的场景。 ■ 计划中使用VIEW/DERIVED TEMP TABLE的场景。 ■ 并行查询支持定义临时表的场景， 但有如下限制： ■ 不支持临时表上不带条件的SELECT COUNT(*) ■ 不支持在Memory Engine临时表的并行。 ○ 支持新版本的审计日志格式，增加了VIP信息。 ○ 支持索引页面空闲比率控制，减少SMO概率和latch竞争，提升写入性能。 ○ 支持多队列的模拟AIO，增强刷脏和写入性能。 ○ 支持在core文件中不记录buffer pool的内容，降低core文件的大小，避免影响线上服务。

类别	说明
问题修复	○ 修复当达到TempTable最大内存限制的时候，原来逻辑中TempTable存储引擎会误报out-of-memory的错误，而不是回退到基于磁盘存储上的问题。 ○ 修复当排序缓存（sort buffer size）参数设置过小时，在InnoDB全文搜索中使用ORDER BY会导致集群不可用的问题。 ○ 修复在临时表出现同名列情况下，无法找到对应的正确Field的问题。 ○ 修复并行查询中，当使用MAX/MIN函数结合GROUP BY并使用松散扫描数据时，无法被Kill的问题。 ○ 修复故障切换时的若干问题。 ○ 修复并行查询下的若干问题。 ○ 修复使用SHOW BINARY LOGS命令可能会阻碍事务提交的问题。

- 8.0.1.1.2（发布日期：20200409）

类别	说明
新增功能和性能优化	○ 支持基于字符串前缀的排序优化方法（即优先使用字符串前缀进行排序，如果前缀相同，再使用字符串的全长进行排序）。对长字符类型的列进行排序时，可以通过指定该列值最大不同的前缀长度来加速比较，减少排序时间。 ○ 新增在如下场景中支持并行查询的能力： ■ 支持Range Cost的估算模型的并行。 ■ 支持Temporary Table表的并行。 ■ 支持Semijoin物化Lookup和Scan的策略下的并行。 ○ 增加了3类可以被PolarDB智能路由用来支持连接保持功能的会话状态tracker；同时，tracker打开后可以跟踪会话中user variable的改变、临时表的创建和删除、SQL语句中的prepare和dealloction操作等。 ○ 优化DDL过程中Drop AHI的性能，降低DDL对集群性能的影响。 ○ 增加表回收站的功能，避免出现因误删导致数据丢失的情况。 ○ 优化在大缓冲池临时表空间truncate的表现，降低临时表操作对集群性能的影响。
问题修复	○ 修复当聚合函数存在于IF函数中的场景下执行ROLLUP的问题。 ○ 修复Blob类型在排序过程中的问题。 ○ 修复并行中使用Prepare中包含聚合函数SQL的特定问题。 ○ 修复并行查询下的若干问题。 ○ 修复可能会清除过多Redo日志的问题。 ○ 修复RO节点上Redo日志的相关问题。

- 8.0.1.1.1（发布日期：20200328）

类别	说明

类别	说明
新增功能和性能优化	○ 支持在子查询含有ROLLUP的场景中使用并行。 ○ 支持语句并发控制功能。 ○ 增加 `POLARDB_INDEX` 的Hint。 ○ 优化主节点和只读节点同步延迟。 ○ 支持线程池（Thread Pool）。 ○ 支持TDE keyring_rds插件。 ○ 支持全球数据库（GDN）。 ○ 优化无锁事务系统， 优化读写性能。
问题修复	○ 修复并行查询下的若干问题。 ○ 修复ONLINE DDL过程中统计信息可能为0的问题。 ○ 优化用户态文件系统，加速集群启动。 ○ 修复innodb_flush_method被设置为all_o_direct时可能会导致集群不可用的问题。 ○ 修复事务提交放锁时可能会导致集群不可用的问题。 ○ 修复慢日志truncate时可能会阻塞用户请求的问题。 ○ 修复压缩页在RO上可能会导致集群不可用的问题。 ○ 修复线程池可能会错误断开杀复制连接的问题。

- 8.0.1.1.0（发布日期：20200205）

类别	说明
新增功能和性能优化	○ 增强并行查询的能力，支持企业级分析ROLLUP函数的并行计算。 ○ 增加优化器的估算模型能力（如增强条件过滤的选择率和对并行查询的代价估算模型），被执行的SQL可以根据Selectivity更准确地选择并行计划还是串行计划。 ○ 支持对按照FIFO模式分配工作线程的并行工作线程进行统一管理，避免大量并行查询造成系统资源耗尽的问题。
问题修复	○ 修复并行查询的内存相关系列问题。 ○ 修复并行查询下的若干不稳定问题。

- 8.0.1.0.6（发布日期：20200101）

类别	说明
问题修复	○ 修复在主节点降为备节点时Binlog Index文件未关闭的问题。 ○ 修复RO节点在访问已经被清除的Undo页时会出现集群不可用的问题。 ○ 修复在RO节点进行主备切换时，后台线程访问到不存在表空间页面的问题。 ○ 修复在集群关闭时由于日志线程已经退出之后还在写Redo日志导致的集群不可用问题。

- 8.0.1.0.5（发布日期：20191203）

类别	说明
新增功能和性能优化	○ Optimizer trace中支持并行查询的相关信息（例如您可以通过Optimizer trace的工具分析为何使用并行或者未使用并行的原因）。 ○ 增加并行查询相关的Hint，支持通过SQL Hint的方式显性启用并行和指定并行度。 ○ 支持在READ COMMITTED下INSERT..SELECT的并行扫描，您可以使用INSERT..SELECT语句将导入的数据加入到另一个表。
问题修复	○ 修复并行查询下的若干问题。 ○ 修复在主备切换时，备节点升级为主节点时出现的节点不可用的问题。 ○ 修复在主备切换时因使用部分DDL语句引起故障的问题。 ○ 修复锁限制导致报错too many connection error的问题。

PolarDB MySQL引擎5.7版本发布日志

PolarDB MySQL引擎 5.7支持如下版本：

- 5.7.1.0.17（发布日期：20220114）

类别	说明
新增功能和性能优化	○ 支持热点行优化功能。具体请参见热点行优化。 ○ 支持热备节点功能，进一步优化高可用效率。

- 5.7.1.0.16（发布日期：20211214）

类别	说明
问题修复	优化了手动触发checkpoint的策略。

- 5.7.1.0.15（发布日期：20211117）

类别	说明
新增功能和性能优化	新增Fast Query Cache功能。具体请参见Fast Query Cache。

- 5.7.1.0.14（发布日期：20211019）

类别	说明

类别	说明
问题修复	○ 提供一个optimizer hint使得用户可以控制是否强制视图进行merge。具体请参考MySQL官方文档。 ○ 支持 `SELECT FOR UPDATE WAIT N` 语法。 其中N表示等待超时的秒数，仅针对单个行锁。如果一个查询需要锁定多行时，系统不会对多个行进行时间累计，仅对单行进行超时检测。等待时间超过N秒，系统则返回锁等待超时错误： `Lock wait timeout exceeded; try restarting transaction` 。

- 5.7.1.0.13（发布日期：20210910）

类别	说明
新增功能和性能优化	GDN从集群中的只读节点支持通过alter polar to slave命令切换成主节点，从而实现从集群的高可用。

- 5.7.1.0.12（发布日期：20210827）

类别	说明
新增功能和性能优化	○ GDN中的主集群与同步节点之间支持多连接，提高物理复制的吞吐量。 ○ 支持从只读节点上获取Binlog，详情请参见远程获取并解析PolarDB MySQL引擎Binlog日志。
问题修复	○ 修复Statement Concurrency Control指定库表的规则无法匹配的问题。 ○ 加快只读节点和从集群应用redo log，提升主节点的同步效率。

- 5.7.1.0.11（发布日期：20210708）

类别	说明
新增功能和性能优化	新增polar_replica_work_on_nonblock_mdl_mode参数。开启该参数时，只读节点上的RU/RC隔离级别的未提交事务将不再阻塞主节点上的DDL操作，同时只读节点上将不在保证表定义的事务特性。

- 5.7.1.0.10（发布日期：20210615）

类别	说明
新增功能和性能优化	兼容MySQL 8.0，支持SELECT FOR UPDATE/SHARE SKIP LOCKED/NOWAIT语法。具体请参见MySQL 8.0。
问题修复	RO节点支持将innodb temp table数据落盘。

- 5.7.1.0.9（发布日期：20210513）

类别	说明
问题修复	○ 多表场景下，支持存储引擎快速启动。 ○ 修复了在虚拟列上回滚Update操作导致数据库崩溃的问题。

- 5.7.1.0.8（发布日期：20210419）

类别	说明
新增功能和性能优化	当只读节点和主节点建立复制关系时，新增主节点是否需要立即执行checkpoint策略的功能。
问题修复	○ 将KICKOUT修改为非保留关键字。 ○ 修复主备切换时，TDE加密表中的加密信息可能丢失导致解密失败的问题。

- 5.7.1.0.7（发布日期：20210310）

类别	说明
新增功能和性能优化	○ 新增并行DDL功能，提升DDL性能，详情请参见并行DDL。 ○ 新增innodb_buffer_pool_in_core_file参数，以便将buffer pool从core file中移除。 ○ Index Hints新增支持如下关键字，以便优化器处理查询时使用或忽略指定的索引： ■ `GROUP_INDEX` 或 `NO_GROUP_INDEX` ：使用或忽略指定的索引以进行带有GROUP BY操作的索引扫描。 ■ `INDEX` 或 `NO_INDEX` ：强制服务器使用或忽略指定索引。 ■ `JOIN_INDEX` 或 `NO_JOIN_INDEX` ：强制MySQL对任何访问方法使用或忽略指定的索引。 ○ Join Order Hint新增支持如下关键字，以便优化器选择合适的表连接顺序： ■ `JOIN_FIXED_ORDER` ：强制优化器使用FROM子句中出现的顺序来连接表。 ■ `JOIN_ORDER` ：指导优化器使用指定的表顺序连接表。 该Hint适用于命名表。 优化器可以将未命名的表放在连接顺序中的任何位置，包括指定表之间。 ■ `JOIN_PREFIX` ：指导优化器为连接执行计划的前几个表使用指定的表顺序来连接表。 该Hint适用于命名表。 优化器会将所有的其他表放在命名表之后。 ■ `JOIN_SUFFIX` ：指导优化器为连接执行计划的最后几张表使用指定的表顺序来连接表。 该Hint适用于命名表。 优化器会将所有的其他表放在命名表之前。

类别	说明
问题修复	○ 将token number的对照表进行分区并预留部分token，以修复插入新token后，同一语句digest hash值不一致的问题。 ○ 修复变配过程中，由于只读节点无法在主节点上成功注册，导致变配失败的问题。 ○ 修复在打开 `session_track_temporary_tables` 系统变量的情况下，在存储过程中创建或删除临时表会导致集群异常退出的问题。

- 5.7.1.0.6.3（发布日期：20210222）

类别	说明
问题修复	修复部分场景下SHOW PROCESSLIST结果不正确的问题。

- 5.7.1.0.6.2（发布日期：20210210）

类别	说明
问题修复	修复在只读节点上执行使用了Index Merge优化的查询时，偶发性地出现1032错误码 `Can't find record in TABLE` 的问题。

- 5.7.1.0.6.1（发布日期：20210202）

类别	说明
问题修复	○ 修复读写节点上刷脏异常的问题。 ○ 修复在已执行过主备切换的集群上，可能无法再进行更换主可用区操作的问题。 ○ 修复由于执行SHOW PROCESSLIST时错误调用了THD（Thread Descriptor），导致的数据库崩溃问题。

- 5.7.1.0.6（发布日期：20210129）

类别	说明
新增功能和性能优化	○ 新增支持Statement Queue功能，详情请参见Statement Queue。 ○ 新增支持秒级加字段（ `Instant add column` ）功能，详情请参见秒级加字段。 ○ 新增支持Returning功能，详情请参见Returning。
问题修复	修复升级PolarDB内核版本时，系统表丢失的问题。

- 5.7.1.0.5（发布日期：20201231）

类别	说明

类别	说明
问题修复	○ 修复主节点修改密钥后，在只读节点上进行查询时，只读节点会崩溃的问题。 ○ 修复在只读节点上分析分区表时，只读节点会崩溃的问题。 ○ 修复主备切换后，由于表空间不一致导致新的主节点会崩溃的问题。

- 5.7.1.0.4（发布日期：20201117）

类别	说明
问题修复	○ 修复主备切换后，无法将数据插入临时表的问题。 ○ 修复当在只读节点上扩展临时表空间时，只读节点不可用的问题。 ○ 修复Simulated AIO不能正常工作的问题。

- 5.7.1.0.3（发布日期：20201021）

类别	说明
新增功能和性能优化	支持透明数据加密TDE功能。
问题修复	○ 修复只读节点和主节点间统计信息不一致的问题。 ○ 修复SHOW PROCESSLIST返回结果会显示错误信息的问题。

- 5.7.1.0.2（发布日期：20200828）

类别	说明
问题修复	○ 修复从RDS MySQL 5.7迁移后不能扩展.IDB文件的问题。 ○ 禁止在主备切换时执行CF_STATUS_COMMAND相关命令，以保证主备切换的正常运行。 ○ 修复因后台更新统计信息线程和Truncate逻辑出现Page争用，导致主节点不可用的问题。

- 5.7.1.0.1（发布日期：20200730）

类别	说明
问题修复	○ 修复当部分PolarDB专属命令缺少对应命令数字时，导致maxscale不能正常访问的问题。 ○ 修复通过从回收站恢复方式创建的集群由于找不到表空间，导致集群不可用的问题。 ○ 修复执行 `promote replica` 命令时，有文件未关闭导致unmount PFS文件崩溃的问题。

PolarDB MySQL引擎5.6版本发布日志

PolarDB MySQL引擎5.6支持如下版本：

- 5.6.1.0.30（发布日期：20211110）

类别	说明
问题修复	○ 修复分区表统计信息不稳定的问题。 ○ 修复报文长度为251时，数字长度编码错误的问题。

- 5.6.1.0.29（发布日期：20210909）

类别	说明
新增功能和性能优化	○ 数据库内核支持事务断点续传。 ○ 支持Fast Query Cache。具体请参见Fast Query Cache。
问题修复	加快只读节点和从集群应用redo log，提升主节点的同步效率。

- 5.6.1.0.28（发布日期：20210723）

类别	说明
新增功能和性能优化	新增polar_replica_work_on_nonblock_mdl_mode参数。开启该参数时，只读节点上的RU/RC隔离级别的未提交事务将不再阻塞主节点上的DDL操作，同时只读节点上将不在保证表定义的事务特性。
问题修复	优化表空间元信息的加载速度。对于拥有百万级以上表文件的数据库实例，能大幅缩短主节点崩溃的恢复时间以及从节点的启动时间。

- 5.6.1.0.27（发布日期：20210601）

类别	说明
问题修复	○ 优化Standby节点在执行truncate polar logs lsn命令删除文件时按照4K对齐。 ○ 将KICKOUT修改为非保留关键字。 ○ 调整只读节点在首次注册到主节点时是否要立即触发checkpoint策略。当LSN差值小于特定的阈值时，可不触发checkpoint策略。 ○ load polar logs支持添加条件语句。 ○ 修复autoinc重复的问题。

- 5.6.1.0.26（发布日期：20210319）

类别	说明

类别	说明
问题修复	○ 修复当执行FLUSH PRIVILEGES或FLUSH GRANT命令来批量授权时，可能出现连接失败的问题。 ○ 修复部分情况下由于分区表估计逻辑提前中止，导致的分区表估计错误的问题。 ○ 修复部分场景下SHOW PROCESSLIST结果不正确的问题。 ○ 修复在打开 `session_track_temporary_tables` 系统变量的情况下，在存储过程中创建或删除临时表会导致集群不可用的问题。

- 5.6.1.0.25（发布日期：20210205）

类别	说明
新增功能和性能优化	优化库表级恢复功能，提升数据恢复速度。
问题修复	○ 修复只读节点读取到已经TRUNCATE的undo page后会导致节点不可用的问题。 ○ 修复在已执行过主备切换的集群上，可能无法再进行更换主可用区操作的问题。

- 5.6.1.0.24（发布日期：20210122）

类别	说明
新增功能和性能优化	○ 优化了PolarDB引擎初始化进程，缩短大表场景下引擎的启动时间。 ○ 新增支持在**基本信息**页查看集群的**内核版本**信息。
问题修复	○ 修复从RDS迁移至PolarDB过程中，无法TRUNCATE Undo Log的问题。 ○ 修复无法新增系统表的问题。 ○ 修复库表恢复时主节点不可用的问题。 ○ 修复当查询结果为DECIMAL类型时，排序不正确的问题。 ○ 修复若干在特殊情况下可能出现的MySQL服务进程崩溃的问题。

- 5.6.1.0.23（发布日期：20210104）

类别	说明
问题修复	修复只读节点上的内存泄漏问题。

- 5.6.1.0.22（发布日期：20201225）

类别	说明
新增功能和性能优化	PFS新增支持目录索引，以提升海量表场景下的集群性能。

类别	说明
问题修复	○ 修复某些情况下，主备切换后节点角色不对导致集群不可用的问题。 ○ 修复Statement Queue未初始化导致只读节点崩溃的问题。 ○ 修复新增系统表在主备切换后没有初始化的问题。 ○ 修复线程池和连接控制（Connection Control ）功能会同时开启的问题。 ○ 修复全文索引存在重复ID导致集群不可用的问题。 ○ 修复某些情况下只读节点查询失败的问题。 ○ 修复日志复制线程退出异常造成复制中断的问题。

- 5.6.1.0.21（发布日期：20201112）

类别	说明
新增功能和性能优化	○ 支持全量恢复方式2：恢复到过去时间点功能。 ○ 支持透明数据加密TDE功能。 ○ PFS支持单表恢复所需的多盘挂载能力。 ○ 优化在物理复制中AHI（Adaptive Hash Index）锁的资源占用问题。
问题修复	○ 修复SELECT语句在使用DYNAMIC RANGE AND INDEX MERGE情况下出现OOM（Out Of Memory）的问题。 ○ 修复某些情况下创建或删除账号导致集群崩溃的问题。 ○ 修复某些情况下无法重连STANDBY节点的问题。 ○ 修复当主备切换发生异常时导致集群无法启动的问题。 ○ 修复某些情况下Binlog线程状态不正确的问题。

- 5.6.1.0.20（发布日期：20201027）

类别	说明
新增功能和性能优化	提升某些情况下物理复制的效率。
问题修复	○ 修复某些情况下执行 `CREATE TABLE... SELECT` 命令会导致集群崩溃的问题。 ○ 修复存储过程中，因派生表使用次数过多导致内存泄露的问题。 ○ 修复使用按时间点恢复数据时，恢复时间不准或恢复失败的问题。 ○ 修复PolarFS异常日志输出过多的问题。 ○ 修复关闭外键检查后，执行DDL导致表丢失的问题。 ○ 修复同时TRUNCATE多个临时表导致只读节点崩溃的问题。

- 20200831（发布日期：20200922）

类别	说明
新增功能和性能优化	PFS支持本地盘，支持挂载可写快照及性能优化。

类别	说明
问题修复	○ 修复某些情况下使用Statement Queue功能会导致集群崩溃的问题。 ○ 修复Corefiles占用过多空间的问题。 ○ 修复只读节点和主节点间统计信息不一致的问题。 ○ 修复只读节点切换为主节点后，其它只读节点无法连接新主节点的问题。 ○ 修复只读节点和主节点间全文索引缓存不一致的问题。

- 20200616（发布日期：20200701）

类别	说明
新增功能和性能优化	○ 支持Statement Queue功能，详情请参见Statement Queue。 ○ 优化RO和RW之间的复制延迟。 ○ 支持复制LOCK TABLE时的MDL锁。
问题修复	○ 修复MDL锁复制中的问题。 ○ 修复FTS INDEX创建在系统表的问题。 ○ 修复热点更新优化的一些问题。

- 20200601（发布日期：20200605）

类别	说明
新增功能和性能优化	○ 支持热点行更新优化，详情请参见热点行优化。 ○ 支持Statement Concurrency Control功能，详情请参见Statement Concurrency Control。
问题修复	修复线程池（Thread Pool）带来的写性能下降的问题。

- 20200507（发布日期：20200513）

类别	说明
新增功能和性能优化	○ 增加并发控制功能。 ○ 增加一个参数控制索引页的空闲空间。 ○ 优化Simulate AIO。
问题修复	○ 修复bool flag类导致的性能退化问题。 ○ 修复pfs_umount表没有关闭导致集群不可用的问题。

版本升级

- DB Version和Minor Version升级

PolarDB暂时不提供直接升级DB Version和Minor Version版本（如您无法将PolarDB MySQL引擎 5.6版本直接升级到PolarDB MySQL引擎 8.0版本，同时您也无法将PolarDB MySQL引擎 8.0.1版本直接升级到8.0.2版本），但您可以通过DTS将数据从源版本迁移或同步到目标版本的集群完成升级，关于如何迁移或同步数据，请参见数据迁移或同步方案概览。

- Revision Version升级

  关于如何升级Revision Version版本，请参见Revision Version升级。

1.1.3. 数据库代理发布记录

本文主要介绍了PolarDB MySQL引擎数据库代理的版本更新记录。

查询版本号

您可以登录PolarDB控制台，在目标集群的**配置与管理 > 版本管理**页直接查看数据库代理版本信息。

当前PolarDB MySQL引擎数据库代理包含1.x.x和2.x.x两大版本，两个版本的区别如下：

- **1.x.x**

  2021年2月1日前创建的集群下的数据代理版本，该版本不再进行新功能迭代开发，只进行问题修复。

- **2.x.x**

  2021年2月1日（包含）后新创建的集群下的数据代理版本，该版本属于当前的主流版本，所有的新增功能都在该版本上进行迭代开发。新增的功能包括连接保持、数据脱敏等。

> **说明**
> - 您只能将1.x.x版本升级到1.x.x系列版本的最新版本，将2.x.x版本升级到2.x.x系列版本的最新版本。
> - 您无法跨版本将1.x.x版本升级到2.x.x版本，或者将2.x.x版本回退到1.x.x版本。如需跨版本升级或回退操作，请提交工单联系技术支持。

数据库代理1.x.x版本发布日志

> **说明** 以下版本为数据库代理的主流版本，并不包含全部的数据库代理版本。您查询到的版本号，可能并不包含在下列的版本列表中。

- 1.13.30（发布日期：20211230）

类别	说明
问题修复	修复提交会话一致性的事务后，读取不到最新的已提交数据的问题。

- 1.13.27（发布日期：20211116）

类别	说明
问题修复	○ 修复了某些客户端SSL不兼容的问题。 ○ 优化了insert语句代理parse的性能。

- 1.13.25（发布日期：20210818）

类别	说明
问题修复	○ 修复MySQL账号认证失败导致的代理内存泄漏问题。 ○ 修复多Endpoint场景下可能导致代理异常崩溃的问题。

- 1.13.22（发布日期：20210721）

类别	说明
新增功能&性能优化	○ 开启事务级连接池后支持select last_insert_id()的用法。 ○ 开启事务级连接池后支持FOUND_ROWS函数。 ○ COM_STATISTICS协议支持路由到只读节点。 ○ 优化事务级连接池。 ○ 优化全局一致性：只要有一个只读节点满足一致性要求就可以将请求路由到只读节点。 ○ geo函数支持路由到只读节点。 ○ 增加部分内部监控指标。
问题修复	○ 修复部分SQL语句解析不正确导致路由错误的问题。 ○ 修复特定场景下执行stmt_long_data()后，stmt_exec()执行失败的问题。 ○ 修复load data infile执行失败的问题。

- 1.13.5（发布日期：20201201）

类别	说明

类别	说明
新增功能&性能优化	○ 最终一致性支持事务拆分。 ○ 支持如下HINT语法： ■ `force node connection /*force_proxy_internal*/set force_node = 'pi-aaaaaaaaa';` 该连接之后的所有请求只发往节点pi-aaaaaaaaa。如果这个节点不健康的话，则报错 `set force node 'pi-aaaaaaaaa' is not found, please check.` 。 ■ `force node query /*force_node='pi-aaaaaaaa'*/ show processlist;` 该条请求只在pi-aaaaaaaa节点上执行，如果这个节点不健康的话，则报错 `'force hint server node is not found, please check.'` 。 ○ 增加部分内部监控指标。
问题修复	○ 修复 `select type, status, mode, where gtx_id = '4' FOR UPDATE;` 语句包含mode关键字导致路由到只读节点的问题。 ○ 修复特定条件下负载不均衡的问题。 ○ 修复prepare场景下stmt_close可能失败的问题。

- 1.12.10（发布日期：20201019）

类别	说明
问题修复	○ 修复MySQL 8.0版本SSL加密建立连接异常问题。 ○ 修复数据库节点的状态从DOWN变成RUNNING时，数据库代理负载的新请求到该节点异常的问题。

- 1.12.7（发布日期：20200806）

类别	说明
新增功能&性能优化	○ 支持show full processlist语法。 ○ 支持XA事务语法。

类别	说明
问题修复	○ 修复事务级连接池存在的若干问题。 ○ 修复GDN中访问只读节点存在的问题。 ○ 修复MySQL 8.0执行show processlist命令报错的问题。 ○ 修复若干建立连接失败问题。

- 1.11.12（发布日期：20200622）

类别	说明
新增功能&性能优化	○ 支持事务级连接池。 ○ PolarDB MySQL引擎8.0版本只读Endpoint支持并行查询。

- 1.10.7（发布日期：20200318）

类别	说明
新增功能&性能优化	支持全局一致性功能。
问题修复	修复会话级连接池初始化系统环境变量异常的问题。

- 1.9.23（发布日期：20200221）

类别	说明
新增功能&性能优化	○ 支持通过root账号连接集群。 ○ 支持SSL证书加密。
问题修复	○ 修复change user失败的问题。 ○ 修复load file失败的问题。 ○ 修复用户侧收到sequence错误的报文，导致应用报 `Exception: Packets out of order` 的问题。 ○ 修复主节点异常时只读模式的EndPoint被断开的问题。

- 1.9.14（发布日期：20191224）

类别	说明
新增功能&性能优化	○ 支持HINT语法： `/*FORCE_SLAVE*/` 和 `/*FORCE_MASTE*/` 。 ○ 支持主库是否接受读。
问题修复	○ 修复charset默认值获取错误导致乱码的问题。 ○ 修复返回的mysql version string不正确的问题。

数据库代理2.x.x版本发布日志

> **说明** 以下版本为数据库代理的主流版本，并不包含全部的数据库代理版本。您查询到的版本号，可能并不包含在下列的版本列表中。

- 2.4.27（发布日期：20211230）

类别	说明
新增功能&性能优化	支持行存储和列存储自动引流功能。
问题修复	○ 修复kill session失败的问题。 ○ 修复频繁修改参数类型时，执行COM_STMT_EXECUTE失败的问题。 ○ 修复事务内执行Select for update后的读请求被路由到只读节点的问题。 ○ 修复主节点重启期间新建的连接，执行插入utf8mb4类型的特殊字符失败的问题。 ○ 修复提交会话一致性的事务后，读取不到最新的已提交数据的问题。

- 2.4.22（发布日期：20210910）

类别	说明
问题修复	○ 修复数据库代理未正常关闭prepare，导致数据库内存使用量高的问题。 ○ 修复特定场景下连接保持失败的问题。 ○ 修复默认集群地址改为只读模式时，导致数据库代理异常的问题。 ○ 修复特定场景下数据库代理异常崩溃的问题。

- 2.4.18（发布日期：20210812）

类别	说明
问题修复	○ 修复jdbc应用没有指定字体集时，数据库节点重启的瞬间或者账号认证失败的瞬间导致数据乱码的问题。 ○ 修复flink客户建连失败的问题。 ○ 修复部分临时表路由错误的问题。

- 2.4.17（发布日期：20210714）

类别	说明
新增功能&性能优化	○ PolarDB MySQL引擎5.7版本支持故障切换场景下的事务连接保持。 ○ 增加内部监控指标。

类别	说明
问题修复	○ 优化MySQL账号认证失败导致的RT响应变长的问题。 ○ 优化只读节点异常后新建连接快速跳过该节点的问题。 ○ 修复只读Endpoint开启并行计算后建连失败的问题。 ○ 修复current timestamp路由出错的问题。 ○ 修复for update parse路由不正确的问题。 ○ 修复@a在join子句里语法分析不正确，导致路由错误的问题。 ○ 修复MySQL 8.0客户端通过空密码认证失败的问题。 ○ 修复部分name prepare执行失败的问题。

- 2.4.12（发布日期：20210520）

类别	说明
新增功能&性能优化	支持动态脱敏。
问题修复	修复特定场景下的代理异常问题。

- 2.4.7（发布日期：20210315）

类别	说明
新增功能&性能优化	支持连接保持。
问题修复	修复lock in shared mode路由不正确的问题。

升级版本

若集群当前数据库代理Proxy的版本不是最新版本，则可以根据实际需要进行升级操作。具体操作请参见升级版本。

> **说明** 高版本的问题可能也存在于您当前的版本，可以尝试进行版本升级操作。如果数据库代理的版本升级后，问题仍未解决，建议您提交工单，联系技术支持。

1.2. 产品公告

1.2.1. 数据库代理企业版发布说明

随着PolarDB MySQL引擎的广泛应用，越来越多的企业应用场景对数据库代理提出了更高的要求，比如快速自动弹性扩展、独占物理资源等。为了更好的满足企业应用场景的要求，阿里云发布了全新的数据库代理功能——**数据库代理企业版（付费使用）**。

发布时间

数据库代理企业版于2021年12月9日正式发布。PolarDB MySQL引擎集群版和历史库集群版均支持数据库代理企业版。

费用

数据库代理企业版暂定于2022年4月1日起付费使用。

系列	购买类型	策略
单节点和历史库单节点版	不论是新购还是存量，都不支持数据库代理企业版。	
集群版和历史库集群版	新购集群	在2021年12月9日后，新购集群仅提供数据库代理企业版，并将于2022年4月1日正式开始收费。
	存量按量付费集群	对于存量的按量付费集群，数据库代理于2021年12月9日已自动切换到数据库代理企业版，并将于2022年4月1日开始收费。
	存量包年包月集群	对于存量的包年包月集群，数据库代理于2021年12月9日已自动切换到数据库代理企业版，且： • 如果集群在2022年4月1日已到期，则数据库代理企业版将于2022年4月1日开始收费。 • 如果集群在2022年4月1日未到期，则数据库代理企业版将在集群订单到期日开始收费。

数据库代理企业版功能特点

数据库代理企业版提供两种版本：**企业通用版**和**企业独享版**。

- **企业通用版**：配套集群子系列的通用规格，它可以共享CPU物理资源，可根据业务负载，提供智能秒级资源弹性扩展能力。
- **企业独享版**：配套集群子系列的独享规格，它可以独占CPU物理资源，具有更好的性能稳定性。

下表将为您介绍数据库代理历史版本与当前数据库代理企业版之间的差异。

对比项	历史版本	当前全新版本：数据库代理企业版	
		企业通用版	企业独享版
计费类型	免费	付费（约占计算节点总费用的5~10%） 当前免费使用，暂定于2022年4月1日起付费	
资源类型	共享CPU物理资源，无弹性扩展能力	共享CPU物理资源，可根据业务负载，提供智能秒级资源弹性扩展能力 计算节点秒级弹性扩展功能预计于2022年1月上线	独占物理资源，具有更好的性能稳定性
部署架构	高可用冗余架构		
实例规格	最低配置：2核		

对比项	历史版本	当前全新版本：数据库代理企业版	
		企业通用版	企业独享版
性能	数据库代理企业版下，集群存储最大IOPS相比历史版本可提升50%，不同规格集群的最大IOPS可参见计算节点规格 性能提升已覆盖所有2021年12月9日后新购的集群；预计于2022年1月覆盖所有存量集群。		
只读节点配置	只读节点与主节点配置保持一致	只读节点无需与主节点配置保持一致，可以根据业务负载降配从而节省成本 该功能预计于2022年2月底前后上线	
只读节点数量	支持1~15个只读节点		
多主架构	不支持	支持，该功能预计于2022年3月底前后上线 关于多主架构，可参见多主架构发布预告	
计算节点秒级弹性扩展	不支持	支持 该功能预计于2022年1月上线	独占资源，无需支持
Proxy限流保护	不支持	支持 该功能预计于2022年1月底前后上线	
数据脱敏（安全）	支持		
变配业务无感	支持		
Endpoint	1个主地址+7个集群地址		
主从切换	闪断20~30秒	连接/事务不中断，短暂阻塞5~10秒 该功能预计于2022年1月底前后上线	
一致性	• 最终一致性 • 会话一致性 • 全局一致性		
连接池	支持		
事务拆分	支持		
防闪断（连接保护）	支持		

1.2.2. 多主架构灰度发布说明

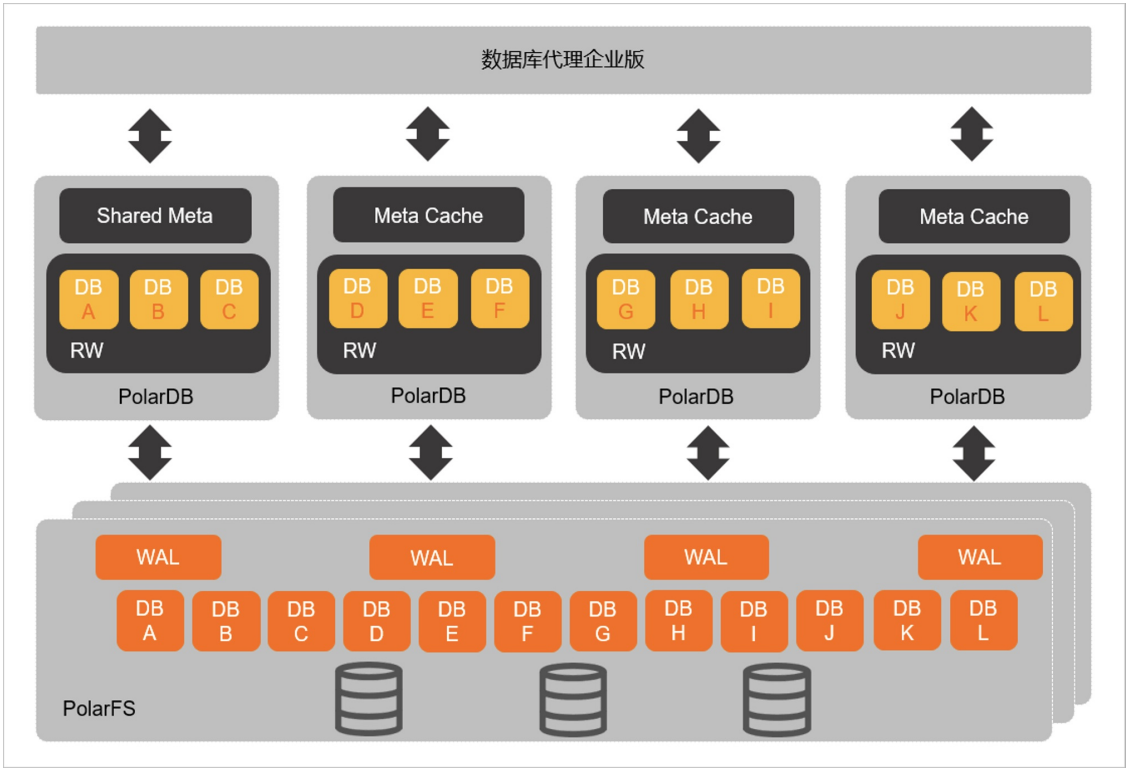
PolarDB MySQL引擎重磅推出了多主架构集群版。

简介

随着PolarDB MySQL客户的不断增加，大规模头部客户不断涌入，部分头部客户业务体量规模庞大，使得目前PolarDB MySQL引擎的单写（一写多读）架构在特定场景下，写性能出现瓶颈。

PolarDB MySQL引擎全新推出多主架构集群版，实现从一写多读架构到多写多读架构的升级，主要面向多租户、游戏、电商等高并发读写的应用场景。

多主架构的架构图如下：



发布时间

2022年1月20日。

多主架构以多主架构集群版的形态呈现，当前处于灰度发布阶段。如有需求，请提交工单申请试用。

版本要求

多主架构集群版目前仅支持PolarDB MySQL引擎8.0内核版本。

核心优势和能力

- 支持不同数据库在不同计算节点并发写入。
- 集群版支持1个写节点和最多15个只读节点，多主架构集群版最多可支持32个节点同时写入。
- 支持数据库跨节点动态调度，秒级完成切换，极大提升集群整体并发读写能力。
- 若某个计算节点发生故障，可秒级完成切换。

适用场景

多主架构主要面向SaaS多租户、游戏、电商等高并发读写的应用场景。

- **SaaS多租户场景：满足高并发性能需求，实现租户间负载均衡**

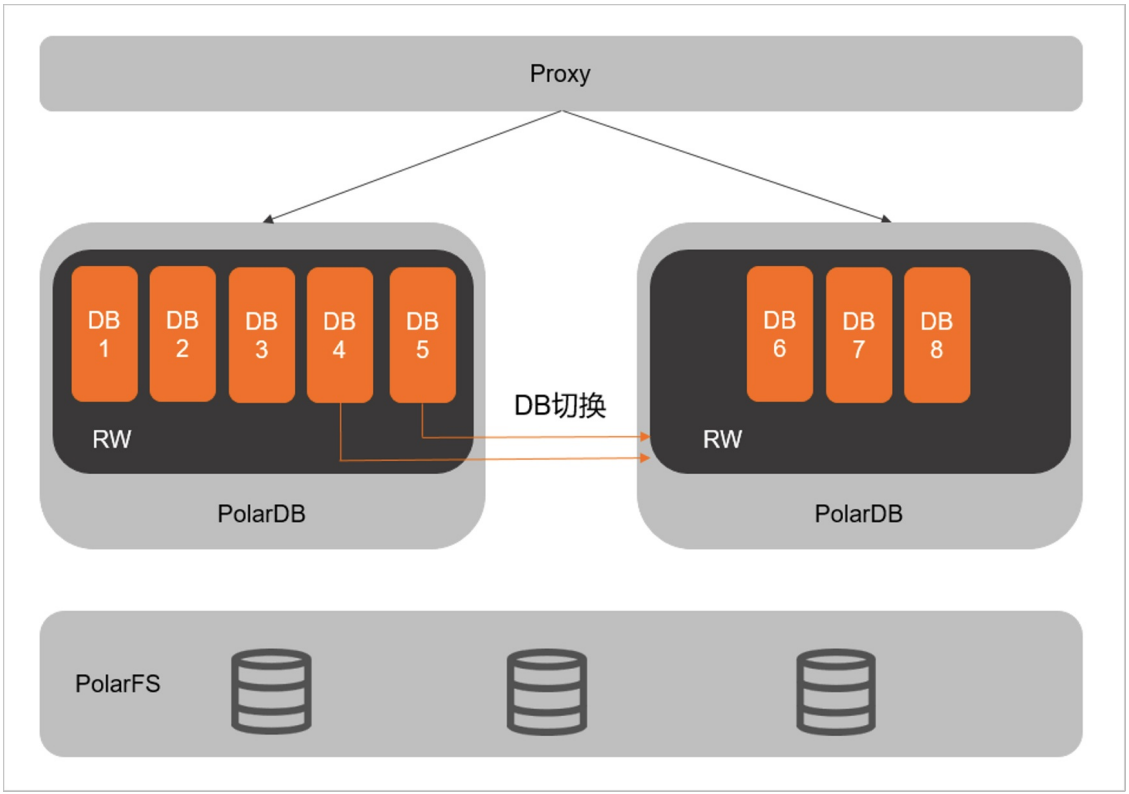
场景特点：租户的数据库数量变化较快，负载变化较大，需要经常在不同的实例之间调配数据库资源，以便达到最佳用户体验。

解决方案：多主架构可帮助客户快速将租户的数据库在不同RW节点间进行切换，从而实现负载均衡。

- **分服游戏场景：更好的性能和扩展能力，支持世界服架构**

场景特点：在游戏成长期，数据库负载较大，且呈现为不断增长的趋势特点。通常表现为在游戏成长期期间，会不断增加数据库，导致RW节点负荷也不断增加。而在游戏衰退期，数据库负载逐渐减少，数据库会不断合并，导致RW节点的负荷也呈减少趋势。

解决方案：游戏成长期，可快速将部分数据库切换到新的RW节点，实现负载均衡；游戏衰退期，可快速将数据库聚合到少量RW节点，快速降低运作成本。



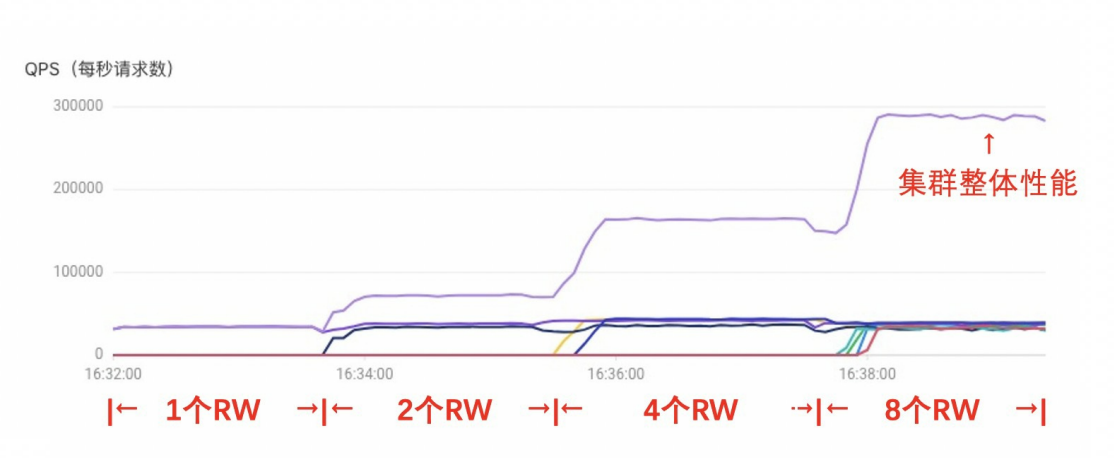
性能提升情况

经测试，随着一个集群中的数据库切换至更多的主节点（RW）上，集群整体并发读写能力几乎呈线性提升。

如下为实际测试数据。

- 测试背景：集群包含8个数据库，8个RW节点。
- 测试过程：初始情况下，8个数据库全部负载在其中一个RW节点上，然后对所有数据库同步执行相同的压力测试。压测期间，将8个数据库分别平均切换到2个RW节点、4个RW节点、8个RW节点上，观察集群整体的性能变化趋势。

- 性能变化趋势如下，以QPS为例：



从上图可以看出，随着数据库切换至更多的RW上时，集群整体并发读写能力得到了极大的提升，几乎呈现为线性提升。

相关文档

- 多主架构集群版
- 使用说明

1.2.3. 列存索引（IMCI）灰度发布说明

PolarDB MySQL引擎重磅推出了列存索引（IMCI）特性。

简介

当前PolarDB MySQL引擎主要面向OLTP场景，广泛应用于在线业务，日常产生大量的数据。但是，PolarDB MySQL基于行存的查询性能并不能满足所有应用场景的需求。通常情况下，为了实现复杂分析型查询，需要将数据从PolarDB MySQL导出，然后导入到外部专用的OLAP系统中再进行分析查询。这样一来，需要使用两套数据库系统，架构复杂性、运维工作量和成本都会大大增加。

PolarDB MySQL引擎重磅推出的列存索引（In-Memory Column Index，简称IMCI）面向OLAP场景大数据量复杂查询。通过列存索引，PolarDB MySQL实现了一体化的实时事务处理和实时数据分析的能力，成为一站式HTAP数据库产品解决方案。通过一套数据库系统，即可满足业务的OLTP及OLAP需求。

发布时间

2022年1月20日。

列存索引当前处于灰度发布阶段。如有需求，请提交工单申请试用。

版本要求

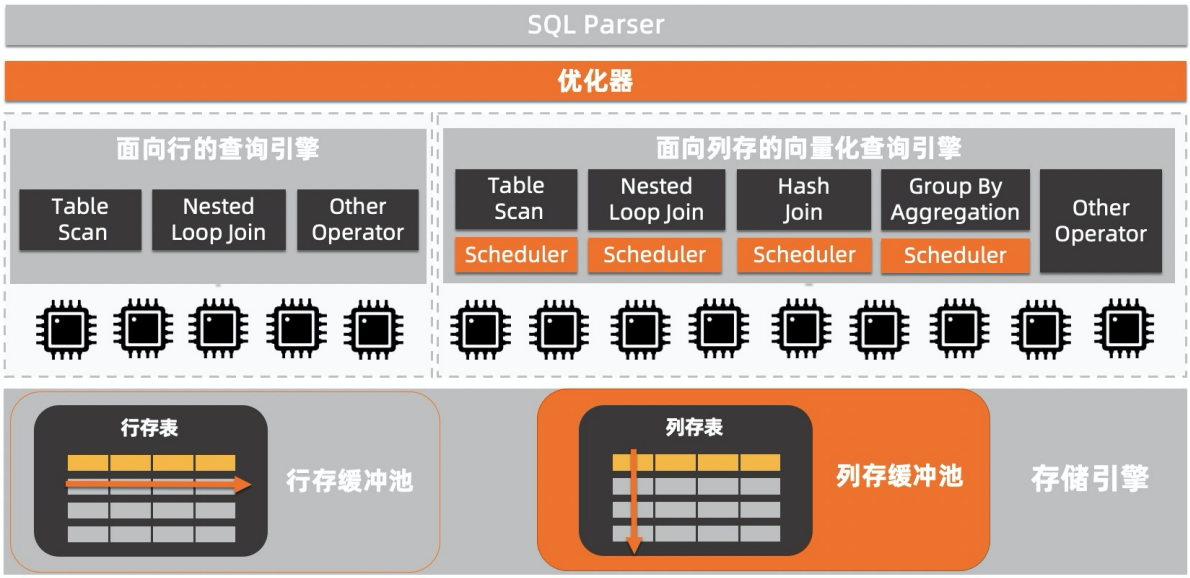
仅PolarDB MySQL集群版，内核版本为8.0.1.1.22及以上的集群支持列存索引特性。单节点、历史库单节点版、历史库集群版均不支持列存索引。

实现形态

为了实现OLAP和OLTP的计算资源隔离，当前仅支持在只读节点上实现列存索引，即OLAP查询请求只发给只读节点，而不会发给主节点。至于OLAP查询请求是发给只读行存节点，还是只读列存节点，可根据行存/列存分流方案进行配置。

技术原理

列存索引特性在PolarDB MySQL引擎中的功能架构图如下：



从以上架构图可以看到，PolarDB MySQL引擎从存储引擎、查询引擎、优化器三个层面设计了列存索引的特性：

- 存储引擎：支持实时事务级别一致性的行列混合存储；
- 查询引擎：面向列存的向量化并行查询引擎，支持极速的单表和多表查询；
- SQL Parser/优化器：面向行列混合存储的CBO优化器，可以根据代价自动选择行存或者列存执行查询请求；

在此架构下，PolarDB MySQL引擎实现了100%兼容MySQL协议的基础上，同时获得数个数量级的查询加速效果。

核心优势

PolarDB MySQL引擎依托列存索引特性，具备如下优势：

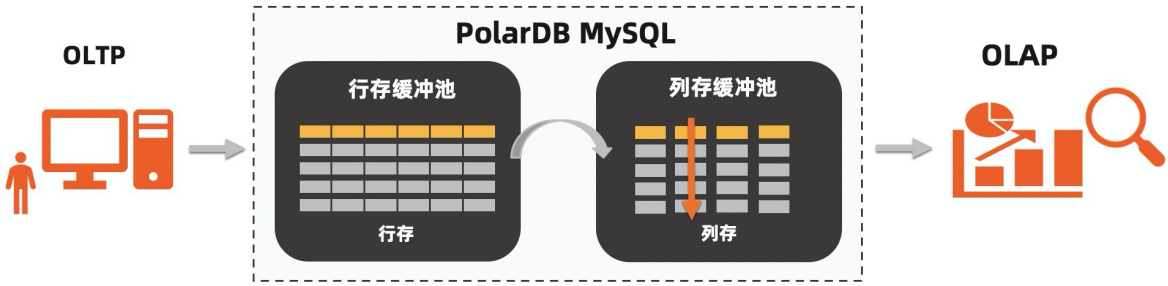
- 100%兼容MySQL：列存具有与MySQL一致的数据类型系统，支持灵活的类型转换，100%兼容MySQL协议；
- 极致的HTAP性能：PolarDB在OLTP方面本身具备极致性能。列存索引使其OLAP性能也与专用的OLAP数据库系统处于同一水平；
- 行列混合存储，降低成本：同时支持行存储和列存储两种格式，且实时保证行列的事务级一致。列存更具有低成本的优势。

适用场景

PolarDB MySQL引擎的列存索引特性提供了一站式HTAP产品体验，可以应用于多种业务场景：

- 对在线数据有实时数据分析需求的场景，如实时报表；
- 专用数据仓库场景：依托PolarDB提供的海量数据存储能力，汇聚多个上游数据源，将其作为专用数据仓库使用；

- ETL数据加速计算场景：依托PolarDB基于列存索引提供的强大而灵活的计算能力，在PolarDB中使用SQL来实现ETL功能。



性能提升情况

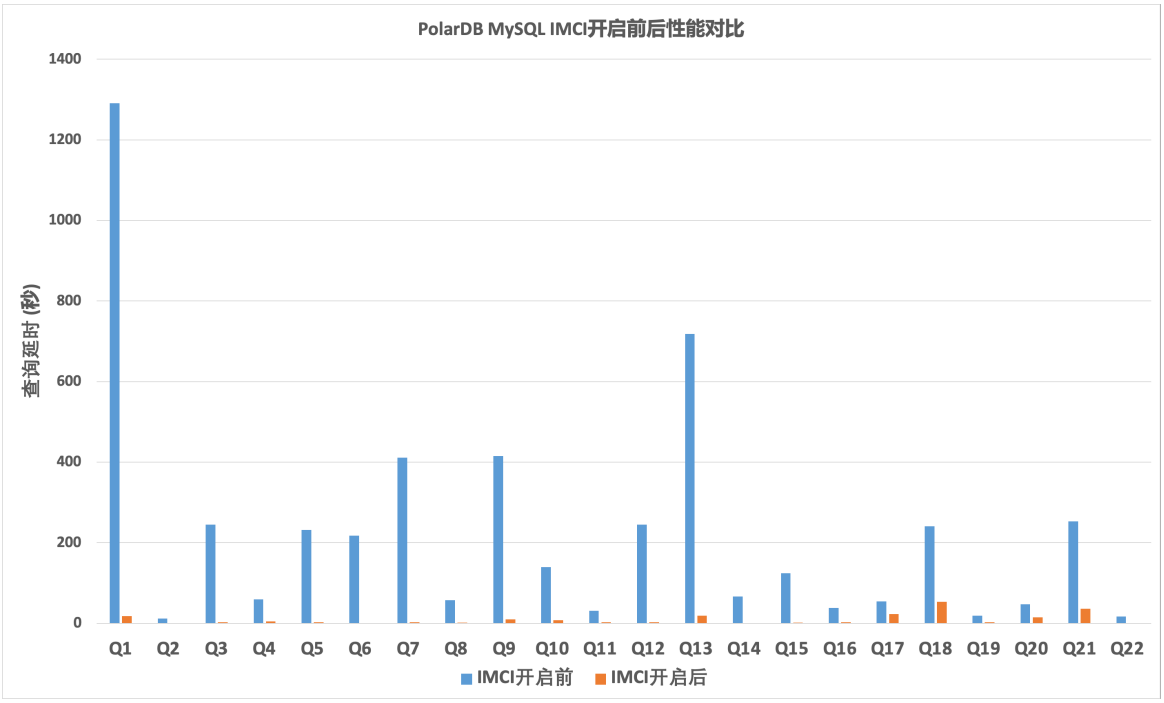
列存索引功能对SQL查询操作有明显的加速作用，查询性能甚至可以提升百倍。接下来我们以标准TPC-H测试的数据表和SQL为例，体验列存索引对查询的加速效果。

- **测试方法**：TPC-H是业界常用的一套基准，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查询包含8张数据表、22条复杂的SQL查询，大多数查询包含若干表Join、子查询和Group by聚合等。

> 说明
>
> 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。

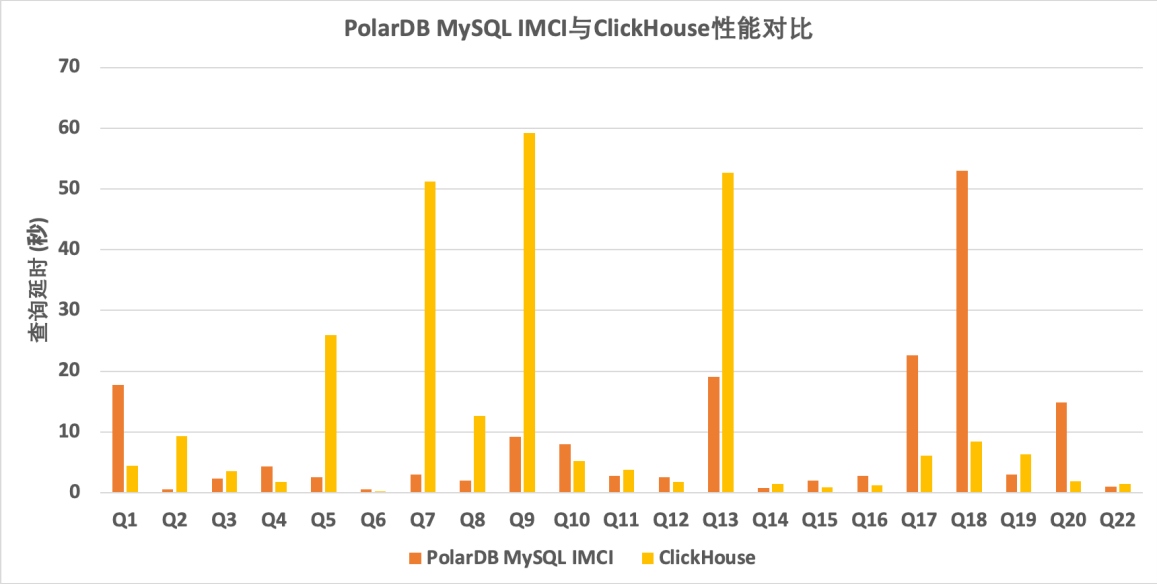
- **数据量**：100 GB。
- **测试结果**：
  - PolarDB MySQL开启列存索引前后性能对比

    下图展示了PolarDB MySQL开启列存索引前后分别执行TPC-H的22条查询语句的查询响应时间的对比。



- PolarDB MySQL开启列存索引后与ClickHouse性能对比

  下图展示了PolarDB MySQL开启了列存索引后，执行TPC-H的21条查询语句（Q21除外，ClickHouse不支持）的查询响应时间，与相同数据量和数据结构的ClickHouse数据库的对比。



- **结论**：
  - 列存索引对大多数的复杂查询操作都有加速作用，查询性能提升非常明显，甚至可达到百倍。
  - 与传统OLAP数据库ClickHouse相比：PolarDB MySQL开启列存索引后，与ClickHouse性能相比各有优劣。其中PolarDB MySQL在单表Scan/AGG、Join等场景中表现突出。未来PolarDB MySQL的列存索引特性将在聚合加速、窗口函数等方面持续优化和突破。

相关文档

- 列存索引概述
- 添加只读列存节点
- 行存/列存分流
- 列存索引语法说明

1.3. 安全公告

1.3.1. Apache Log4j2漏洞对PolarDB无影响

近期，阿里云安全团队向Apache官方报告了Apache Log4j2远程代码执行漏洞（CVE-2021-44228）。云原生关系型数据库PolarDB技术团队确认该漏洞对云原生关系型数据库PolarDB无影响。

更多信息，请参见【漏洞预警】Apache Log4j2 远程代码执行漏洞（CVE-2021-44228）。

2.快速入门

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

使用流程

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

1. 购买按量付费集群或购买包年包月集群
2. 设置白名单
3. 创建数据库账号
4. 申请集群地址和主地址
5. 连接数据库集群

3.数据迁移或同步

3.1. 数据迁移或同步方案概览

云原生关系型数据库PolarDB提供了多种数据迁移同步方案，可满足不同上云、迁云、同步的业务需求，使您可以在不影响业务的情况下平滑将数据库迁移、同步至阿里云云原生关系型数据库PolarDB上面。

通过使用阿里云数据传输服务（DTS），您可以实现PolarDB的结构迁移、全量迁移和实时同步。

数据迁移

使用场景	文档链接
从RDS迁移至PolarDB	• 一键升级RDS MySQL至PolarDB MySQL引擎（平滑迁移，推荐） • 一键克隆RDS MySQL至PolarDB MySQL引擎 • RDS MySQL迁移至PolarDB MySQL
从PolarDB迁移至RDS	PolarDB MySQL迁移至RDS MySQL
从自建数据库迁移至PolarDB	从自建MySQL迁移至PolarDB MySQL
从第三方云数据库迁移至PolarDB	从Amazon Aurora MySQL迁移至PolarDB MySQL
PolarDB之间的数据迁移	PolarDB MySQL集群间的数据迁移

数据同步

使用场景	文档链接
从RDS同步至PolarDB	RDS MySQL同步至PolarDB MySQL集群
PolarDB间的同步	• PolarDB MySQL集群间的单向同步 • PolarDB MySQL集群间的双向同步
从自建数据库同步至PolarDB	从ECS上的自建MySQL同步至PolarDB MySQL
从PolarDB同步至RDS	从PolarDB MySQL同步至RDS MySQL
从PolarDB同步至分析型数据库	• 从PolarDB MySQL同步至云原生数据仓库AnalyticDB MySQL • 从PolarDB MySQL同步至云原生数据仓库AnalyticDB PostgreSQL
从PolarDB同步至Datahub	从PolarDB MySQL同步至Datahub
从PolarDB同步至Kafka	从PolarDB MySQL同步到Kafka

3.2. 【注意事项】MySQL 5.7迁移到PolarDB MySQL引擎8.0版本

PolarDB MySQL引擎 8.0版本完全兼容MySQL 5.7，您可以将MySQL 5.7数据库迁移到PolarDB MySQL引擎 8.0版本使用，数据不会丢失，但需要注意客户端版本和PolarDB MySQL引擎 8.0版本的兼容性问题。

如何将MySQL 5.7迁移至PolarDB MySQL引擎 8.0版本请参见如下文档：

- 从RDS MySQL迁移至PolarDB MySQL
- 从Amazon Aurora MySQL迁移至PolarDB MySQL
- PolarDB MySQL集群间的数据迁移
- 从自建MySQL迁移至PolarDB MySQL

客户端版本

您需要将MySQL客户端程序升级到如下版本：

- Java：MySQL Connector/J 8.0及以上版本。
- ODBC：MySQL Connector/ODBC 8.0及以上版本。
- CPP：MySQL Connector/CPP 8.0及以上版本。
- .NET：MySQL Connector/NET 8.0及以上版本。
- Nodejs：MySQL Connector/Nodejs 8.0及以上版本。
- Python：MySQL Connector/Python 8.0及以上版本。
- Python：mysql-connector-Python 8.0.5及以上版本。
- Golang：go-sql-driver/mysql 1.4.0及以上版本。
- PHP：mysqlnd 7.4及以上版本。
- C/CPP：libmysqlclient 8.0及以上版本。

已知客户端问题

- 问题现象：MySQL数据库连接异常，`query_cache_size`无法识别。
  - Driver版本：mysql-connector-java:5.1.42
  - 数据库版本：mysql 8.0.13
  - 解决办法：使用mysql-connector-java:5.1.42以上的版本，该版本相关更新日志请参见Changes in MySQL Connector/J 5.1.43。
- 问题现象：mysql python driver由于`COM_STMT_EXECUTE`的flag没有设置正确且没有发送`com_stmt_fetch`获取结果集，在MySQL 8.0版本上会导致无法正常获取返回结果，在MySQL 5.6/5.7版本可以正常返回结果。
  - Driver版本：mysql-connector-2.2.9。
  - 数据库版本：mysql 8.0.13。
  - 解决办法：安装8.0的python driver。
- 问题现象：在PolarDB MySQL引擎 8.0版本数据库集群上执行带有`kickout`的SQL语句时，会出现如下语法报错：

```
ERROR 1064 (42000): You have an error in your SQL syntax;
```

- 解决办法：建议升级PolarDB MySQL引擎 8.0版本集群到最新修订版本。详细升级操作步骤，请参见版本管理。

3.3. 从RDS迁移至PolarDB

3.3.1. 一键升级RDS MySQL至PolarDB MySQL引擎

PolarDB支持将RDS MySQL一键升级至PolarDB MySQL引擎，升级后PolarDB集群包含源RDS实例的账号、数据库、IP白名单和必要的参数。

方案优势

- 可保留数据库原连接地址，无需应用程序修改任何连接配置即可切换至PolarDB。
- 无需DTS等数据迁移工具，仅需PolarDB控制台即可完成整个迁移流程。
- 迁移完全免费。
- 迁移过程数据0丢失。
- 支持增量迁移，停机时间小于10分钟。
- 支持在线热迁移，迁移过程仅闪断一次（即当业务从RDS切换至PolarDB时）。
- 支持回滚，迁移失败可以在10分钟内恢复。
- 对于包年包月的RDS实例，数据从RDS迁移到PolarDB后，若业务已在PolarDB上稳定运行且不再需要RDS时，您可以申请转单优惠退款，避免浪费闲置的RDS资源，详情请参见包年包月RDS迁移至PolarDB后申请转单优惠退款。

前提条件

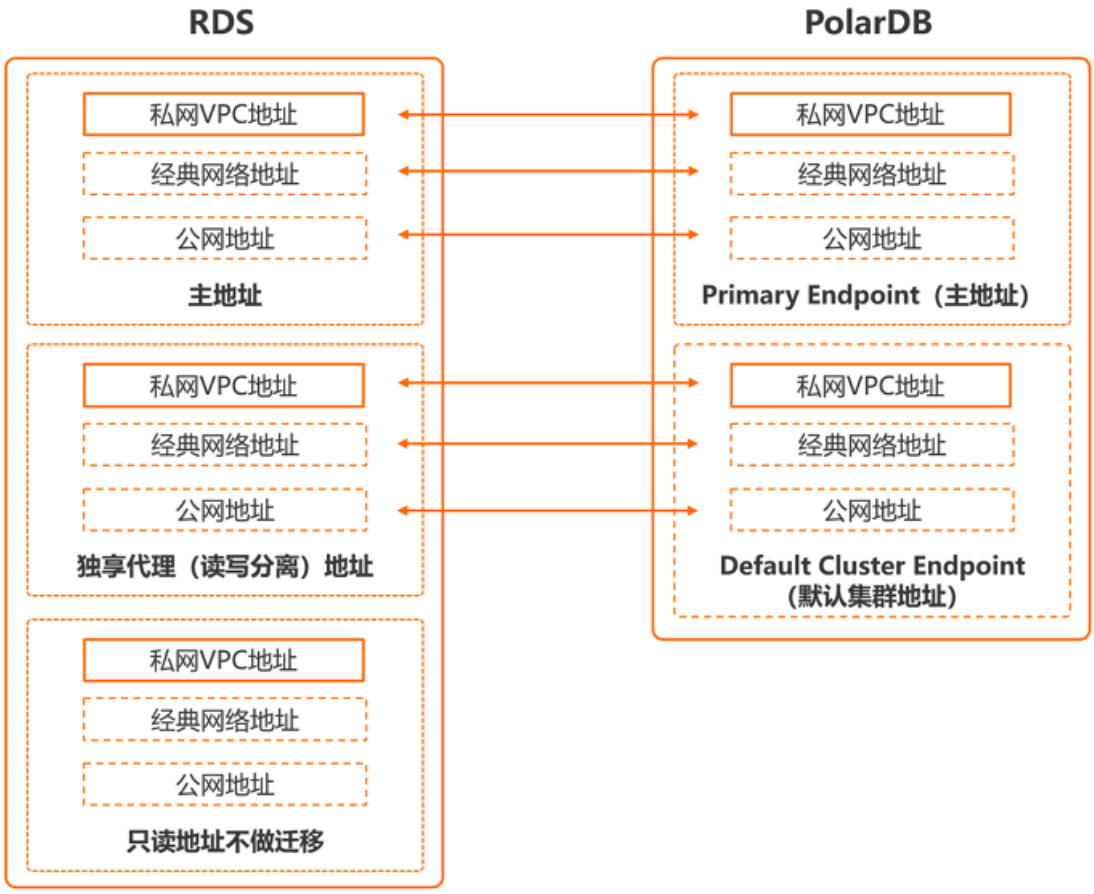
- 源RDS实例版本需为RDS MySQL 5.6或5.7高可用版，且存储类型为本地SSD盘。
- 针对RDS MySQL 5.6，内核小版本需为20190815或以上版本。
- 针对RDS MySQL 5.7，内核小版本需为20200331或以上版本。

> **说明** 您可以执行 `show variables like '%rds_release_date%';` 命令查看源RDS实例的内核小版本。如果源RDS实例的内核小版本低于上述指定版本，您可以将内核小版本升级到最新版。关于如何升级内核小版本，请参见升级内核小版本。

- 源RDS实例未开启TDE和SSL。
- 源RDS实例的表存储引擎为InnoDB。
- 如果RDS处于高安全模式（数据库代理模式），需要创建有高权限账号（请参见创建账号），或者切换到高性能模式（参见【重要】RDS网络链路升级说明），才能进行一键升级。

带地址切换

一键升级RDS至PolarDB时支持带地址切换，系统会自动交换RDS和PolarDB上的连接地址，您无需在应用程序端修改任何配置即可自动连接到PolarDB。选择该切换方式后，RDS连接地址对应的PolarDB连接地址如下图所示。



使用带地址切换功能时，需注意以下几点：

- 带地址切换只会切换RDS和PolarDB的域名，Vswitch、Vip等配置不会切换。
- 仅当源RDS和目标PolarDB集群同时存在的连接地址才支持相互切换，默认情况下仅私网主地址支持带地址切换。
- 如需切换其他连接地址，您需在切换前创建好对应的连接地址，否则不会切换。关于如何为PolarDB集群和RDS实例创建连接地址，请参见申请集群地址和主地址和设置连接地址。
- 带地址切换不会切换端口，请确保RDS和PolarDB的连接端口一致（PolarDB和RDS默认使用的端口号均为3306），如需修改端口，请参见修改内外网地址和端口（RDS）或修改连接地址和端口（PolarDB）。
- 切换域名后，可能会存在DNS解析缓存问题，在缓存过期时间内可能会出现连接不上数据库或数据库只支持读操作无法执行写入操作等情况，建议您刷新一下服务器的DNS缓存。
- 切换域名后，如果您需要使用DMS登录PolarDB数据库，必须使用新版本的DMS并且使用集群ID来进行登录，连接串无法登录。

功能限制

- 暂不支持跨地域迁移。
- 迁移期间不允许对源RDS实例执行参数设置的操作。

费用

- 从RDS迁移到PolarDB的操作完全免费，您只需承担购买PolarDB集群的费用。关于PolarDB集群的费用，详情请参见计费项概览。
- 对于包年包月的RDS实例，从RDS迁移到PolarDB完成后，若确定业务已在PolarDB上稳定运行且不再需要RDS时，您可以申请转单优惠退款，避免浪费闲置的RDS资源，详情请参见包年包月RDS迁移至PolarDB后申请转单优惠退款。

迁移流程介绍

步骤	说明
1、从RDS迁移	本操作将创建一个与源RDS实例数据相同的PolarDB集群，源RDS实例的增量数据会实时同步到该PolarDB集群。
2、迁移切换	• 在执行迁移切换时，您可以选择带连接切换，系统会自动交换RDS和PolarDB上的连接地址（PolarDB集群上需具备对应的连接地址，例如经典网络或公网地址），您无需在应用程序端修改任何配置即可自动连接到PolarDB。 • 执行本操作后，源RDS实例为只读状态，PolarDB集群为可读可写状态，同时会将PolarDB集群的新增数据同步到源RDS实例。 **说明** 迁移切换完成后，如果您发现数据存在异常等问题，可以执行迁移回滚，快速恢复至迁移前的状态。
3、完成迁移	• 您可以观察一段时间后确认业务运行正常，再执行本操作（需要在7天内完成）。 • 执行本操作后，PolarDB集群和源RDS实例间的数据同步关系将中断，此时RDS为可读可写状态。

步骤一：从RDS迁移

本操作将创建一个与源RDS实例数据相同的PolarDB集群，源RDS实例的增量数据会实时同步到该PolarDB集群。

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击**创建新集群**。
4. 选择**商品类型**为**包年包月**或**按量付费**。
   - **包年包月**：在创建集群时支付计算节点的费用，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。
   - **按量付费**：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。

   说明 PolarDB MySQL引擎已推出**计算包**。推荐购买**按量付费**的集群，因为**按量付费**的集群支持配合**计算包**使用，比**包年包月**付费方式更划算、更灵活。详情请参见购买方式1：按量付费+计算包（推荐）。
5. 设置如下参数。

参数	说明
地域	选择源RDS MySQL实例所在地域。 **说明** 新建的PolarDB集群也在此地域。
创建方式	选择**从RDS迁移**。 即先从RDS复制全量数据，然后保持增量同步，主要用于迁移。在正式迁移切换前PolarDB的读写状态为只读，且默认开启Binlog。迁移完成后，若原RDS为包年包月实例，可申请转单优惠退款。
源RDS引擎	源RDS实例的引擎类型，固定为**MySQL**，不可变更。
源RDS版本	源RDS实例的版本，您可以选择**5.6**或**5.7**。
源RDS实例	选择源RDS实例，不包括只读实例。
主可用区	可用区是地域中的一个独立物理区域，不同可用区之间没有实质性区别。 您可以选择将PolarDB集群与ECS创建在同一可用区或不同的可用区。 您只需要选择主可用区，系统会自动选择备可用区。
网络类型	PolarDB集群的网络类型，不可变更。
VPC网络 **VPC交换机**	PolarDB集群所属的VPC和虚拟交换机。请确保PolarDB集群与需要连接的ECS创建于同一个VPC，否则它们无法通过内网互通，无法发挥最佳性能。
兼容性	PolarDB集群的数据库引擎版本，默认与源RDS引擎版本保持一致，不可变更。
系列	固定为**集群版（2-16个节点）【推荐】**，无需选择。
子系列	PolarDB MySQL引擎集群版支持**通用规格**和**独享规格**两种子系列，其中： 。 关于两种类型的详细对比，请参见如何选择通用规格和独享规格。
节点规格	按需选择，建议不低于源RDS实例规格。关于PolarDB节点规格，详情请参见计算节点规格。
节点个数	无需选择。系统将自动创建一个与主节点规格相同的只读节点。
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**。

参数	说明
表名大小写	设置集群表名是否区分大小写，您可以选择**不区分大小写（默认）**或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。关于数据备份的计费规则，详情请参见备份存储（超出免费额度）计费规则。
集群名称	○ 集群名称长度为2~128个字符，以大小写字母或中文开头，可包含数字、英文句号（.）、下划线（_）或短划线（-）。 ○ 如果留空，系统将为您自动生成一个集群名称。创建集群后还可以修改集群名称。
资源组	从已创建资源组中选择一个目标资源组。 **说明** 资源组是在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，详情请参见RAM资源分组与授权。

6. 设置**购买时长**（仅针对**包年包月**集群）和**集群数量**后，单击右侧的**立即购买**。
7. 在**确认订单**页面确认订单信息，阅读并选中服务协议，单击**立即开通**即可。

   开通成功后，需要10~15分钟创建集群，之后您就可以在**集群列表**中看到新创建的集群。

   说明
   - 当集群中的节点状态为**创建中**时，整个集群可能仍未创建完成，此时集群不可用。只有当集群状态为**运行中**时，集群才可以正常使用。
   - 请确认已选中正确的地域，否则无法看到您创建的集群。
   - 当您的数据量较大时，推荐您购买PolarDB存储包，相比按小时付费，预付费购买存储包有折扣，购买的容量越大，折扣力度就越大，详情请参见搭配存储包。
8. 在**支付**页面，确认未支付订单信息和支付方式，单击**订购**。
9. 集群创建成功后，登录PolarDB控制台，确认目标PolarDB集群的**复制延迟**小于60秒即可进行步骤二：迁移切换操作。

> **说明**
>
> - 集群创建后开始从RDS实例同步数据，您需要在7天内进行完成迁移操作，超过7天将自动关闭迁移功能。
> - 您可以在此步骤选择取消迁移，相关影响请参见迁移常见问题。

步骤二：迁移切换

1. 进入PolarDB控制台。
2. 找到目标集群，单击集群的ID。
3. 在**基本信息**页面单击**迁移切换**。

> **说明**
>
> - 切换过程一般小于5分钟。
> - 本操作将交换源RDS实例和目标PolarDB集群的读写状态（即将源RDS实例修改为只读，将PolarDB集群修改为可读可写），同时会更换复制方向（即将PolarDB集群的新增数据同步到RDS实例）。

4. 在**开始切换**对话框中，选择**带地址切换（应用程序不用改连接配置）**或**不带地址切换（应用程序需要改为新的PolarDB连接配置）**。
   - 若您选择**带地址切换（应用程序不用改连接配置）**，操作步骤如下：
     a. 选中**带地址切换（应用程序不用改连接配置）**，系统会自动交换RDS和PolarDB上的连接地址，您无需在应用程序端修改任何配置即可自动连接到PolarDB。

     > **注意** 使用**带地址切换（应用程序不用改连接配置）**前，请务必参见切换注意事项。

     b. 单击**确定**。
   - 若您选择**不带地址切换（应用程序需要改为新的PolarDB连接配置）**，操作步骤如下：
     a. 选中**不带地址切换（应用程序需要改为新的PolarDB连接配置）**，在迁移切换完成后，您需要尽快修改应用程序端的数据库连接地址。
     b. 单击**确定**。

c. 刷新页面，当**PolarDB读写状态**显示为**读写**后，尽快修改应用里的数据库连接地址。

> **说明** 迁移切换完成后，如果您发现数据存在异常等问题，可以进行回滚操作，快速恢复至迁移前的状态也可以选择迁移回滚（可选）。

步骤三：完成迁移

从RDS迁移后，您需要在7天内进行**完成迁移**操作。

> **警告** 由于本操作将中断PolarDB集群和RDS实例间的数据同步，不再提供迁移回滚功能，建议您使用一段时间PolarDB集群，确认正常后再执行本操作。

1. 登录PolarDB控制台。
2. 找到目标集群，单击集群的ID。
3. 在**基本信息**页面，单击**完成迁移**，在弹出的对话框中单击**确定**。

> **说明**
>
> - 单击**确定**后，系统将在约2分钟内中断同步关系，期间迁移状态将显示为**关闭同步**，请耐心等待迁移完成。
> - 您可以在**完成迁移**对话框内选择是否关闭PolarDB集群的Binlog。关闭Binlog会带来少量的写入性能提升，但关闭Binlog后PolarDB集群会自动重启使新配置生效。
> - 如果不再需要源RDS实例，可以释放实例。若实例为包年包月实例，可申请RDS转单优惠退款。详情请参见释放实例和包年包月RDS迁移至PolarDB后申请转单优惠退款。

迁移回滚（可选）

在完成迁移前，如果您发现数据存在异常等问题，可以进行回滚操作，快速恢复至迁移前的状态（RDS实例为可读可写，PolarDB集群为只读，同时会将RDS实例的数据同步到PolarDB集群）。

1. 登录PolarDB控制台。
2. 找到目标集群，单击集群ID。
3. 在**基本信息**页面单击**迁移回滚**。

4. 在**开始回切**对话框中，选择**带地址回切（应用程序不用改连接配置）**或**不带地址回切（应用程序需要改为原RDS连接配置）**。

- 若您选择**带地址回切（应用程序不用改连接配置）**，操作步骤如下：
  a. 选中**带地址回切（应用程序不用改连接配置）**，系统会自动交换RDS和PolarDB上的连接地址，您无需在应用程序端修改任何配置即可自动回切到RDS。
  b. 单击**确定**。此时RDS实例为可读可写，PolarDB集群为只读，同时会将RDS实例的数据同步到PolarDB集群。
- 若您选择**不带地址回切（应用程序需要改为原RDS连接配置）**，操作步骤如下：
  a. 选中**不带地址回切（应用程序需要改为原RDS连接配置）**，在迁移切换完成后，您需要尽快修改应用程序端的数据库连接地址。
  b. 单击**确定**。此时RDS实例为可读可写，PolarDB集群为只读，同时会将RDS实例的数据同步到PolarDB集群。
  c. 刷新页面，当**源RDS读写状态**显示为**读写**后，请尽快修改应用里的数据库连接地址为RDS连接地址。

迁移常见问题

- Q：一键升级RDS MySQL至PolarDB MySQL引擎、一键克隆RDS MySQL至PolarDB MySQL引擎和从RDS MySQL迁移至PolarDB MySQL三者有什么区别？

  A：3者间的区别如下表：

对比项	一键升级RDS MySQL至PolarDB MySQL引擎	一键克隆RDS MySQL至PolarDB MySQL引擎	从RDS MySQL迁移至PolarDB MySQL
是否需要DTS工具	×	×	√
是否支持迁移或同步增量数据	√	×	√
是否影响源RDS操作	×	×	×
源和目标的MySQL版本能否不同	×	×	√

> **说明** 表中√表示支持或需要，×表示不支持或不需要。

- Q：升级后的PolarDB MySQL引擎节点规格需要和源RDS MySQL的实例规格保持一致吗？

  A：您可以按需选择PolarDB MySQL引擎的规格，建议不低于源RDS实例规格。所有集群版的PolarDB节点均为独享型，性能稳定可靠。详情请参见计费项概览。

- Q：升级前是否需要先购买PolarDB MySQL引擎集群？

  A：您无需提前购买PolarDB MySQL引擎集群，步骤一：从RDS迁移即包含购买和创建一个与源RDS实例数据相同的PolarDB集群的操作。

- Q：从RDS迁移会影响源RDS实例吗？

  A：不会影响源RDS实例的正常运行。

- Q：平滑迁移对源RDS实例性能有影响吗？

  A：迁移不会影响源RDS实例上的使用操作，但数据迁移涉及查询操作，会消耗源RDS实例一部分的查询性能。

- Q：平滑迁移对业务有影响吗？

A：平滑迁移能够保证迁移过程不丢失数据，停机（即暂停业务，不产生增量数据，而非停用数据库）时间小于10分钟，如果有需要还可以进行回滚。

- Q：取消迁移会有什么影响？

  A：取消迁移后，源RDS实例可以修改参数；PolarDB集群恢复可读可写，且数据不会释放。手动取消时可以选择是否关闭PolarDB集群的Binlog，自动取消时不会关闭。

- Q：升级完成后，将业务切换到PolarDB，应用程序端的连接地址是否需要修改？

  A：您可以在迁移切换时选择**带地址切换（应用程序不用改连接配置）**，系统会自动交换RDS和PolarDB上的连接地址，您无需在应用程序端修改任何配置即可自动连接到PolarDB。

- Q：迁移切换时选择了**带地址切换（应用程序不用改连接配置）**，迁移完成后-为什么PolarDB集群仍然使用新的连接地址？

  A：仅当源RDS和目标PolarDB集群同时存在的连接地址才支持相互切换，默认情况下仅私网主地址支持带地址切换。如需切换其他连接地址，您需在切换前创建好对应的连接地址，否则不会切换。关于如何为PolarDB集群和RDS实例创建连接地址，请参见申请集群地址和主地址和设置连接地址。

- Q：源RDS实例中还包含只读实例，若选择**带地址切换（应用程序不用改连接配置）**，只读实例的连接地址能否一并切换？

  A：不能。一键升级至PolarDB不会迁移源RDS只读实例的连接地址，仅支持迁移源RDS主实例的连接地址和读写分离地址。您需要在应用程序端手动修改源RDS只读实例的连接地址为PolarDB的连接地址。

- Q：业务成功切换后，为什么连接不上PolarDB数据库或连接成功但只支持读操作无法执行写入操作？

  A：切换域名后，可能会存在DNS解析缓存问题，在缓存过期时间内可能会出现连接不上数据库或数据库只支持读操作无法执行写入操作等情况，建议您刷新一下服务器的DNS缓存。

- Q：一键升级到PolarDB前，能否先进行兼容性测试并简单评估迁移工作量？

  A：您可以先通过一键克隆RDS MySQL至PolarDB MySQL引擎功能克隆一份数据到PolarDB进行兼容性测试和评估迁移工作量，测试没有问题后再参照本文操作一键升级至PolarDB。

- Q：迁移切换后，为什么在PolarDB控制台上看不见**完成迁移**按钮？

  A：若您已经执行过**完成迁移**操作，该按钮将会消失，避免您重复执行相同操作。

- Q：一键升级至PolarDB后，还需要在目标PolarDB集群中创建与源RDS实例相同的账号和密码吗？

  A：不需要。升级后PolarDB集群将包含源RDS实例的账号密码、数据库、IP白名单和必要的参数等信息。

- Q：源RDS实例已开启了TDE或SSL，如何再迁移至PolarDB集群？

  A：已开启了TDE或SSL的RDS实例不支持一键升级至PolarDB集群，您可以选择使用DTS将源RDS迁移至PolarDB。更多详情，请参见从RDS MySQL迁移至PolarDB MySQL。

- Q：一键升级是否支持跨版本升级？如将RDS MySQL 5.6升级至PolarDB MySQL引擎 8.0版本？

  A：暂不支持。若需要将RDS MySQL 5.6跨版本升级至PolarDB MySQL引擎 8.0版本，您可以通过DTS进行迁移。更多详情，请参见从RDS MySQL迁移至PolarDB MySQL。

- Q：若在一键升级至PolarDB MySQL引擎前，源RDS实例已开启了DTS数据同步任务，升级时是否会影响该任务？

  A：不会。通过一键升级进行迁移时，会先从RDS复制一份全量数据至一个新的PolarDB集群，然后将增量数据保持同步至该PolarDB集群。源RDS上DTS数据同步任务的数据源仍然是源RDS，数据迁移至PolarDB并不会影响源RDS上的运行和操作。

  但迁移完成后，如果您将业务切换到新的PolarDB，且源RDS停止使用了，DTS的数据源是不会自动改到新PolarDB集群的，此时，您需要重新创建DTS同步任务，将数据源改为PolarDB集群。

相关API

API	描述
CreateDBCluster	创建PolarDB集群。 说明 一键升级时，参数CreationOption取值需要为MigrationFromRDS。
DescribeDBClusterMigration	查询PolarDB集群的迁移状态。
ModifyDBClusterMigration	修改迁移任务，进行任务的切换或回滚。
CloseDBClusterMigration	取消或完成迁移。

3.3.2. 包年包月RDS迁移至PolarDB后申请转单优惠退款

当您的业务从RDS迁移到PolarDB后，若确定业务已在PolarDB上稳定运行且不再需要RDS时，您可以通过工单为RDS实例申请转单优惠退款，避免浪费闲置的RDS资源。在符合前提条件的情况下，以购买RDS实例时实际支付的价格为基数，根据RDS实例的剩余时长按比例为您办理退款。

前提条件

- RDS实例的付费类型需为**包年包月（预付费）**，且数据库引擎需为MySQL。
- 目标PolarDB集群的创建方式需为**从RDS迁移**，且该PolarDB集群的包年包月（含按量转包年包月）订单金额需高于源RDS的退款金额。具体迁移过程，可参见一键升级RDS MySQL至PolarDB MySQL引擎。

退款规则

以购买源RDS实例时实际支付的价格为基数，根据源RDS实例的剩余时长按比例退款。退款流向可参见退订后资金流向。

假设源RDS实例的包年价格原价是20000元，购买时长为一年，购买时打折后实际支付价格（订单金额）为10000元。使用半年后，若您完成业务从RDS迁移至PolarDB，并申请了RDS转单优惠退款，则您可以获得5000元的退款金额。

退款申请

请提交工单联系售后客服申请RDS转单优惠退款。

3.3.3. 一键克隆RDS MySQL至PolarDB MySQL引擎

PolarDB支持从RDS MySQL一键克隆数据至新的PolarDB MySQL引擎集群。一键克隆功能将会新建一个与源RDS实例的数据相同的PolarDB集群，PolarDB集群包含源RDS实例的账号、数据库、IP白名单和必要的参数。

前提条件

- 源RDS实例版本需为RDS MySQL 5.6或5.7高可用版，且存储类型为本地SSD盘。
- ○ 针对RDS MySQL 5.6，内核小版本需为20190815或以上版本。

- 针对RDS MySQL 5.7，内核小版本需为20200331或以上版本。

> **说明** 您可以执行 `show variables like '%rds_release_date%';` 命令查看源RDS实例的内核小版本。如果源RDS实例的内核小版本低于上述指定版本，您可以将内核小版本升级到最新版。关于如何升级内核小版本，请参见升级内核小版本。

- 源RDS实例未开启TDE和SSL。
- 源RDS实例的表存储引擎为InnoDB。
- 如果RDS处于高安全模式（数据库代理模式），需要创建有高权限账号（请参见创建账号），或者切换到高性能模式（参见【重要】RDS网络链路升级说明），才能进行一键克隆。

背景信息

PolarDB是阿里云自研的下一代关系型云数据库，主要优势如下：

- 存储容量高：最高可达100 TB。
- 性能高：最高可以提升至MySQL的6倍。
- Serverless存储：存储容量无需提前购买，自动扩缩容，按使用量计费。
- 临时升配：临时升级规格，轻松应对短期的业务高峰。

详情请参见产品优势。

注意事项

源RDS实例的增量数据不会同步到PolarDB集群。

> **说明** 如果需要在新建PolarDB集群的同时，使源RDS实例的增量数据实时同步到PolarDB集群，即实现平滑迁移（不停机迁移），请参见一键升级RDS MySQL至PolarDB MySQL引擎。

功能亮点

- 免费
- 克隆过程数据0丢失

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击**创建新集群**。
4. 选择**商品类型**为**包年包月**或**按量付费**。

- **包年包月**：在创建集群时支付计算节点的费用，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。
- **按量付费**：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。

> **说明** PolarDB MySQL引擎已推出计算包。推荐购买按量付费的集群，因为按量付费的集群支持配合计算包使用，比包年包月付费方式更划算、更灵活。详情请参见购买方式1：按量付费+计算包（推荐）。

5. 设置如下参数。

参数	说明
地域	选择源RDS MySQL实例所在地域。 **说明** 新建的PolarDB集群也在此地域。
创建方式	选择**从RDS克隆**。
源RDS引擎	源RDS实例的引擎类型，固定为**MySQL**，不可变更。
源RDS版本	源RDS实例的版本，您可以选择**5.6**或**5.7**。
源RDS实例	选择源RDS实例，不包括只读实例。
主可用区	可用区是地域中的一个独立物理区域，不同可用区之间没有实质性区别。 您可以选择将PolarDB集群与ECS创建在同一可用区或不同的可用区。 您只需要选择主可用区，系统会自动选择备可用区。
网络类型	PolarDB集群的网络类型，不可变更。
VPC网络 **VPC交换机**	PolarDB集群所属的VPC和虚拟交换机。请确保PolarDB集群与需要连接的ECS创建于同一个VPC，否则它们无法通过内网互通，无法发挥最佳性能。
兼容性	PolarDB集群的数据库引擎版本，默认与源RDS引擎版本保持一致，不可变更。
系列	固定为**集群版（2-16个节点）【推荐】**，无需选择。
子系列	PolarDB MySQL引擎集群版支持**通用规格**和**独享规格**两种子系列，其中： ○ 关于两种类型的详细对比，请参见如何选择通用规格和独享规格。
节点规格	按需选择，建议不低于源RDS实例规格。关于PolarDB节点规格，详情请参见计算节点规格。
节点个数	无需选择。系统将自动创建一个与主节点规格相同的只读节点。

参数	说明
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，您可以选择**不区分大小写（默认）**或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，详情请参见释放集群。
集群名称	○ 集群名称长度为2~128个字符，以大小写字母或中文开头，可包含数字、英文句号（.）、下划线（_）或短划线（-）。 ○ 如果留空，系统将为您自动生成一个集群名称。创建集群后还可以修改集群名称。
资源组	从已创建资源组中选择一个目标资源组。 **说明** 资源组是在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，详情请参见RAM资源分组与授权。

6. 设置**购买时长**（仅针对**包年包月**集群）和**集群数量**后，单击右侧的**立即购买**。
7. 在**确认订单**页面确认订单信息，阅读并选中服务协议，单击**立即开通**即可。

   开通成功后，需要10~15分钟创建集群，之后您就可以在**集群列表**中看到新创建的集群。

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

8. 在**支付**页面，确认未支付订单信息和支付方式，单击**订购**。
9. 登录PolarDB控制台，查看新建的PolarDB集群的状态。

常见问题

Q：从RDS克隆会影响源RDS实例吗？

A：不会影响源RDS实例的正常运行。

相关API

API	描述
CreateDBCluster	创建PolarDB集群。 **说明** 一键克隆时，参数CreationOption取值需要为CloneFromRDS。

后续步骤

请尽快将应用的数据库连接地址修改为PolarDB的地址，详情请参见申请集群地址和主地址。

3.3.4. RDS MySQL迁移至PolarDB MySQL

通过数据传输服务DTS（Data Transmission Service），可以帮助您将MySQL迁移至PolarDB MySQL引擎集群。

支持的源数据库

MySQL迁移至PolarDB MySQL引擎集群，支持源数据库MySQL为以下类型。本文以RDS MySQL实例为例介绍配置流程，其他类型的源数据库配置流程与本案例类似。

- RDS MySQL实例。
- 以下类型的自建数据库：
  - 有公网IP的自建数据库。
  - ECS上的自建数据库。
  - 通过专线、VPN网关或智能网关接入的自建数据库。
  - 通过数据库网关接入的自建数据库。

前提条件

- 已创建源RDS MySQL实例，创建方式，请参见创建RDS MySQL实例。

- 已创建目标PolarDB MySQL引擎集群，详情请参见购买按量付费集群和购买包年包月集群。
- 目标PolarDB MySQL引擎集群的存储空间须大于源RDS MySQL实例占用的存储空间。

注意事项

类型	说明
源库限制	•
其他限制	•
特殊情况	• • 当目标库为PolarDB MySQL时 DTS会自动在PolarDB MySQL中创建数据库，如果待迁移的数据库名称不符合PolarDB MySQL的定义规范，您需要在配置迁移任务之前在PolarDB MySQL中创建数据库。相关操作，请参见管理数据库。

费用说明

迁移类型	链路配置费用	公网流量费用
结构迁移和全量数据迁移	不收费。	通过公网将数据迁移出阿里云时将收费，详情请参见产品定价。
增量数据迁移	收费，详情请参见产品定价。	

迁移类型

- 库表结构迁移

  DTS将源库中迁移对象的结构定义迁移到目标库。

  > **说明**
  > - 目前DTS支持结构迁移的对象为表、视图、触发器、存储过程和存储函数。
  > - 在结构迁移时，DTS会将视图、存储过程和函数中的DEFINER转换为INVOKER。
  > - 由于DTS不迁移USER信息，因此在调用目标库的视图、存储您也可以过程和函数时需要对调用者授予读写权限。

- 全量迁移

  DTS将源库中迁移对象的存量数据，全部迁移到目标库中。

- 增量迁移

  DTS在全量迁移的基础上，将源库的增量更新数据迁移到目标库中。通过增量数据迁移可以实现在自建应用不停服的情况下，平滑地完成数据迁移。

支持增量迁移的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

数据库账号的权限要求

数据库	结构迁移	全量迁移	增量迁移
RDS MySQL实例	SELECT权限	SELECT权限	REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和SELECT权限
PolarDB MySQL引擎集群	读写权限		

数据库账号创建及授权方法：

- RDS MySQL实例请参见创建账号和修改账号权限。
- PolarDB MySQL引擎集群请参见创建数据库账号。

操作步骤

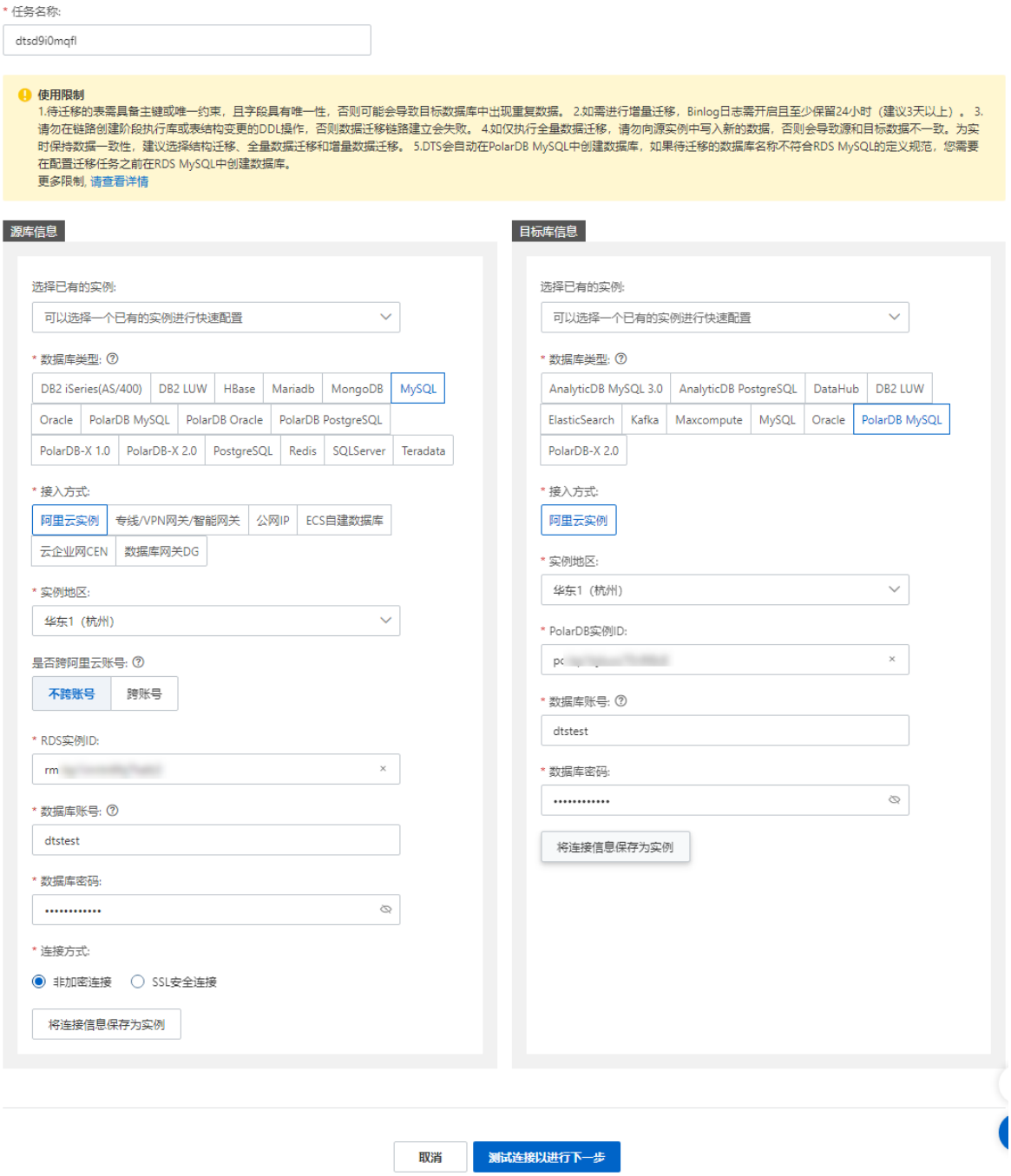
1. 登录新版DTS迁移任务的列表页面。

   > **说明** 您也可以登录DMS数据管理服务。在顶部菜单栏中，选择**集成与开发（DTS）** > **数据迁移**。

2. 在页面左上角，选择迁移实例所属地域。

3. 单击**创建任务**，配置源库及目标库信息。

   > **警告** 选择源和目标实例后，建议您仔细阅读页面上方显示的**使用限制**，以成功创建并执行迁移任务。



类别	配置	说明
无	**任务名称**	DTS会自动生成一个任务名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源库信息	**数据库类型**	选择**MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择源RDS MySQL实例所属地域。

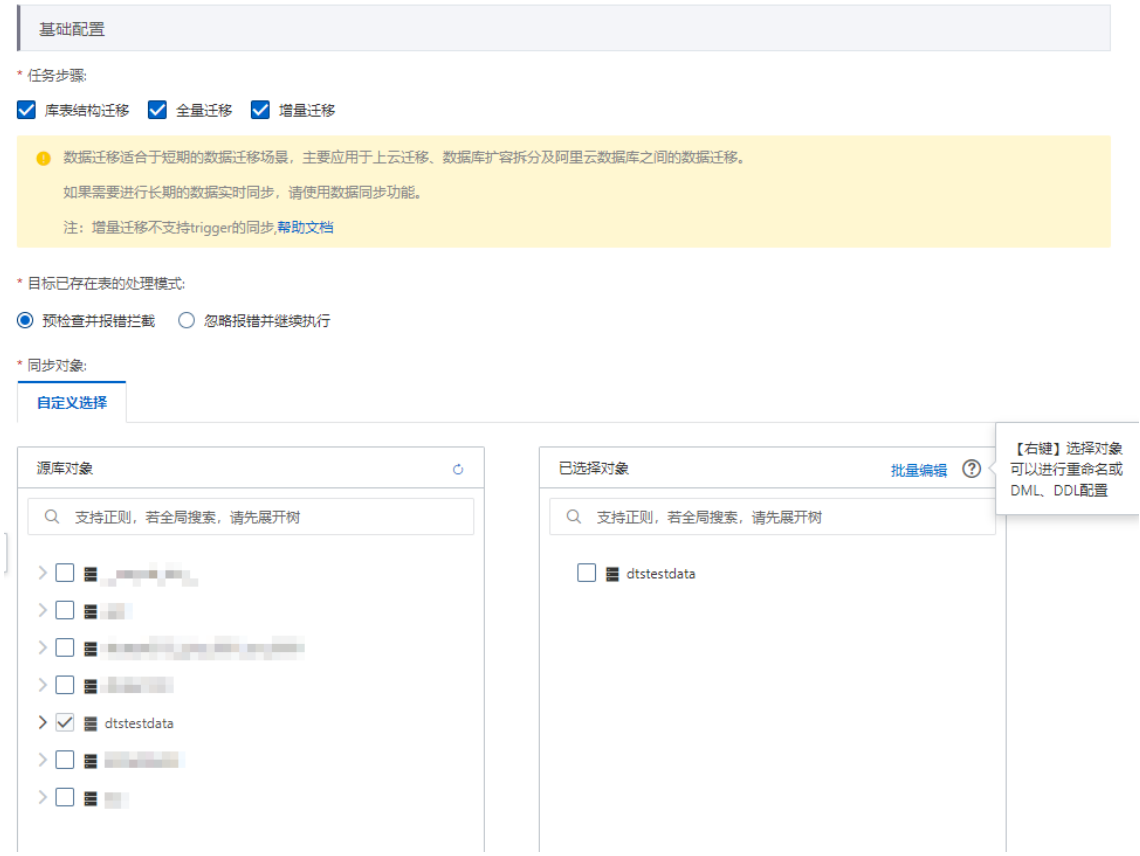
类别	配置	说明
	是否跨阿里云账号	本案例为同一阿里云账号间迁移，选择**不跨账号**。
	RDS实例ID	选择源RDS MySQL实例ID。
	数据库账号	填入源RDS MySQL实例的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号对应的密码。
	连接方式	根据需求选择**非加密连接**或**SSL安全连接**。如果设置为**SSL安全连接**，您需要提前开启RDS MySQL实例的SSL加密功能，详情请参见设置SSL加密。
目标库信息	**数据库类型**	选择**PolarDB MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择目标PolarDB MySQL引擎集群所属地域。
	PolarDB MySQL实例ID	选择目标PolarDB MySQL引擎集群ID。
	数据库账号	填入目标PolarDB MySQL引擎集群的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号对应的密码。

4. 配置完成后，单击页面右下角的**测试连接以进行下一步**。

> **警告**
>
> - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段；如果源或目标数据库是IDC自建数据库或其他云数据库，则需要您手动添加对应地区DTS服务的IP地址，以允许来自DTS服务器的访问。
> - 上述场景中，DTS自动添加或您手动添加DTS服务的公网IP地址段可能会存在安全风险，一旦使用本产品代表您已理解和确认其中可能存在的安全风险，并且需要您做好基本的安全防护，包括但不限于加强账号密码强度防范、限制各网段开放的端口号、内部各API使用鉴权方式通信、定期检查并限制不需要的网段，或者使用通过内网（专线/VPN网关/智能网关）的方式接入。
> - DTS任务完成或释放后，建议您手动检测并删除DTS相关的服务器IP地址段。

5. 配置任务对象及高级配置。

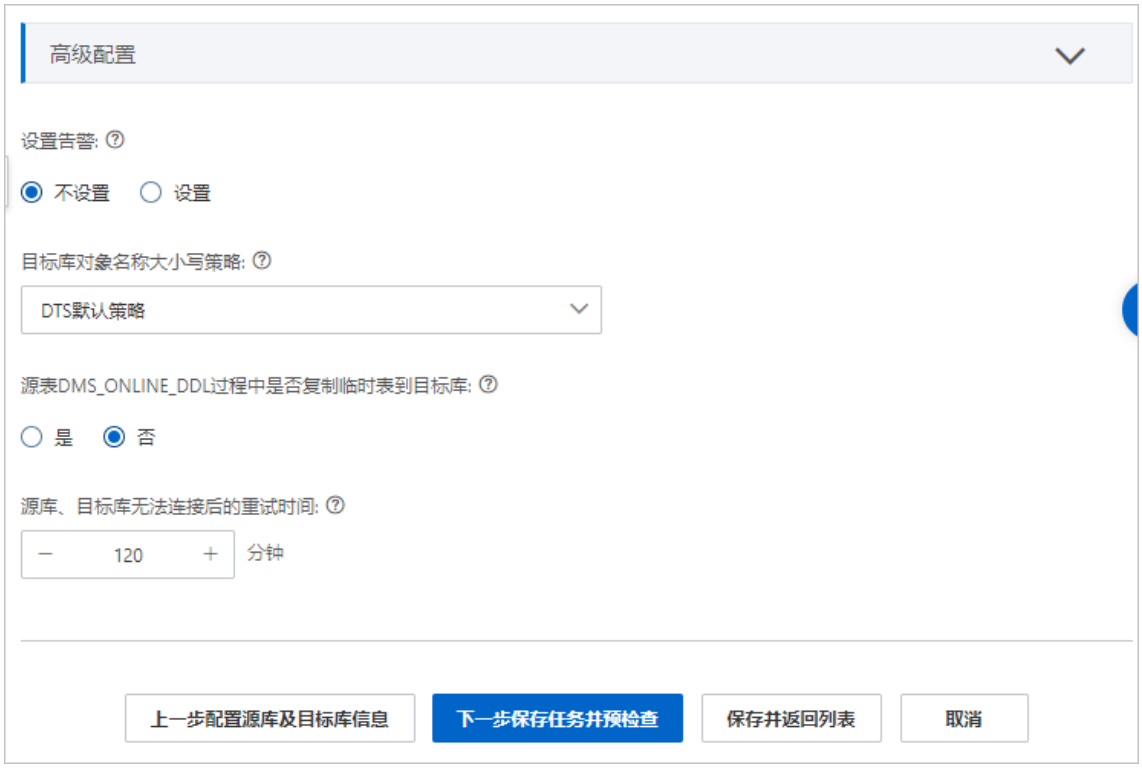
- **基础配置**



配置	说明
任务步骤	■ 如果只需要进行全量迁移，请同时勾选**库表结构迁移**和**全量迁移**。 ■ 如果需要进行不停机迁移，请同时勾选**库表结构迁移**、**全量迁移**和**增量迁移**。 **说明** 如果未选择**增量迁移**，为保障数据一致性，数据迁移期间请勿在源实例中写入新的数据。

配置	说明
目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据迁移任务不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以更改该表在目标库中的名称，请参见库表列名映射。 ■ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，在目标库遇到与源库主键的值相同的记录，则会保留目标库中的该条记录，即源库中的该条记录不会迁移至目标库中。 ■ 表结构不一致的情况下，可能导致只能迁移部分列的数据或迁移失败。
同步对象	在**同步对象**框中单击待迁移的对象，然后单击 › 将其移动到**已选择对象**框。 **说明**
映射名称更改	■ 如需更改单个迁移对象在目标实例中的名称，请右击**已选择对象**中的迁移对象，设置方式，请参见库表列名单个映射。 ■ 如需批量更改迁移对象在目标实例中的名称，请单击**已选择对象**方框右上方的 批量编辑 设置方式，请参见库表列名批量映射。 **说明** 如果使用了对象名映射功能，可能会导致依赖这个对象的其他对象迁移失败。
过滤待迁移数据	支持设置条件过滤数据，详情请参见通过SQL条件过滤任务数据。
增量迁移的SQL操作	选择增量迁移DDL或DML操作，请右击**已选择对象**中的迁移对象，在弹跳框中选择所需增量迁移的DML和DDL操作。支持的操作，请参见支持增量迁移的SQL操作。

- **高级配置**



配置	说明
设置告警	是否设置告警，当迁移失败或延迟超过阈值后，将通知告警联系人。 ■ **不设置**：不设置告警。 ■ **设置**：设置告警，您还需要设置告警阈值和告警联系人。
目标库对象名称大小写策略	您可以配置目标实例中迁移对象的库名、表名和列名的英文大小写策略。默认情况下选择**DTS默认策略**，您也可以选择与源库、目标库默认策略保持一致。更多信息，请参见目标库对象名称大小写策略。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否迁移Online DDL变更产生的临时表数据。 ■ **是**：迁移Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致迁移任务延迟。 ■ **否**：不迁移Online DDL变更产生的临时表数据，只迁移源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。

配置	说明
源、目标库无法连接重试时间	默认重试120分钟，您也可以在取值范围（10~1440分钟）内自定义重试时间，建议设置30分钟以上。如果DTS在设置的时间内重新连接上源、目标库，迁移任务将自动恢复。否则，迁移任务将失败。 **说明** ■ 针对同源或者同目标的多个DTS实例，如DTS实例A和DTS实例B，设置网络重试时间时A设置30分钟，B设置60分钟，则重试时间以低的30分钟为准。 ■ 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

6. 上述配置完成后，单击页面右下角的**下一步保存任务并预检查**。

   说明
   - 在迁移任务正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动迁移任务。
   - 如果预检查失败，单击具体检查项后的，查看失败详情。
     - 您可以根据提示修复后重新进行预检查。
     - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

7. **预检查通过率**显示为**100%**时，单击**下一步购买**。
8. 在**购买**页面，选择数据迁移实例的链路规格，详细说明请参见下表。

类别	参数	说明
信息配置	**链路规格**	DTS为您提供了不同性能的迁移规格，迁移链路规格的不同会影响迁移速率，您可以根据业务场景进行选择，详情请参见数据迁移链路规格说明。

9. 配置完成后，阅读并勾选**《数据传输（按量付费）服务条款》**。
10. 单击**购买并启动**，迁移任务正式开始，您可在数据迁移界面查看具体进度。

3.4. PolarDB间的数据迁移

3.4.1. PolarDB MySQL间迁移

通过数据传输服务DTS（Data Transmission Service），可以帮助您实现PolarDB MySQL引擎集群间的数据迁移。

> **说明** 目前PolarDB MySQL引擎集群暂不支持升级MySQL版本至8.0版本，您可以创建一个新的PolarDB MySQL引擎集群（8.0版本），使用本方法迁移原集群的数据至新集群。跨版本迁移时，建议创建一个按量付费的PolarDB MySQL引擎集群来测试兼容性，测试完成后可释放该集群。

前提条件

- 已创建源和目标PolarDB MySQL引擎集群，详情请参见购买按量付费集群和购买包年包月集群。
- 目标PolarDB MySQL引擎集群的存储空间须大于源PolarDB MySQL引擎集群占用的存储空间。

注意事项

类型	说明

源库限制	• 带宽要求：源库所属的服务器需具备足够出口带宽，否则将影响数据迁移速率。
	• 待迁移的表需具备主键或唯一约束，且字段具有唯一性，否则可能会导致目标数据库中出现重复数据。
	• 如迁移对象为表级别，且需进行编辑（如表列名映射），则单次迁移任务仅支持迁移至多1000张表。当超出数量限制，任务提交后会显示请求报错，此时建议您拆分待迁移的表，分批配置多个任务，或者配置整库的迁移任务。
	• 如需进行增量迁移，Binlog日志：
	  ◦ 需开启，并且loose_polar_log_bin为on。否则预检查阶段提示报错，且无法成功启动数据迁移任务。
	  ◦ 如为增量迁移任务，DTS要求源数据库的本地Binlog日志保存24小时以上，如为全量迁移和增量迁移任务，DTS要求源数据库的本地Binlog日志至少保留7天以上（您可在全量迁移完成后将Binlog保存时间设置为24小时以上），否则DTS可能因无法获取Binlog而导致任务失败，极端情况下甚至可能会导致数据不一致或丢失。由于您所设置的Binlog日志保存时间低于DTS要求的时间进而导致的问题，不在DTS的SLA保障范围内。
	• 源库的操作限制：
	  ◦ 在库表结构迁移和全量迁移阶段，请勿执行库或表结构变更的DDL操作，否则数据迁移任务会失败。
	  ◦ 如仅执行全量数据迁移，请勿向源实例中写入新的数据，否则会导致源和目标数据不一致。为实时保持数据一致性，建议选择结构迁移、全量数据迁移和增量数据迁移。

类型	说明
其他限制	• 建议源和目标PolarDB MySQL引擎的MySQL版本保持一致，以保障兼容性。 • 不支持迁移源PolarDB MySQL引擎只读节点。 • 执行数据迁移前需评估源库和目标库的性能，同时建议业务低峰期执行数据迁移。否则全量数据迁移时DTS占用源和目标库一定读写资源，可能会导致数据库的负载上升。 • 由于全量数据迁移会并发执行INSERT操作，导致目标数据库的表产生碎片，因此全量迁移完成后目标数据库的表存储空间会比源实例的表存储空间大。 • 请确认DTS对数据类型为FLOAT或DOUBLE的列的迁移精度是否符合业务预期。DTS会通过 `ROUND(COLUMN,PRECISION)` 来读取这两类列的值。如果没有明确定义其精度，DTS对FLOAT的迁移精度为38位，对DOUBLE的迁移精度为308位。 • DTS会尝试恢复七天之内迁移失败任务。因此业务切换至目标实例前，请务必结束或释放该任务，或者将DTS访问目标实例账号的写权限用 `revoke` 命令回收掉。避免该任务被自动恢复后，源端数据覆盖目标实例的数据。

费用说明

迁移类型	链路配置费用	公网流量费用
结构迁移和全量数据迁移	不收费。	通过公网将数据迁移出阿里云时将收费，详情请参见产品定价。
增量数据迁移	收费，详情请参见产品定价。	

迁移类型说明

- 库表结构迁移

  DTS将源库中迁移对象的结构定义迁移到目标库。

  > 说明
  > - 目前DTS支持结构迁移的对象为表、视图、触发器、存储过程和存储函数。
  > - 在结构迁移时，DTS会将视图、存储过程和函数中的DEFINER转换为INVOKER。
  > - 由于DTS不迁移USER信息，因此在调用目标库的视图、存储您也可以过程和函数时需要对调用者授予读写权限。

- 全量迁移

  DTS将源库中迁移对象的存量数据，全部迁移到目标库中。

- 增量迁移

  DTS在全量迁移的基础上，将源库的增量更新数据迁移到目标库中。通过增量数据迁移可以实现在自建应用不停服的情况下，平滑地完成数据迁移。

支持增量迁移的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE

操作类型	SQL操作语句
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

数据库账号的权限要求

数据库	权限要求
源PolarDB MySQL引擎集群	待迁移对象的读权限
目标PolarDB MySQL引擎集群	迁移对象的读写权限

PolarDB MySQL引擎集群请参见创建数据库账号。

操作步骤

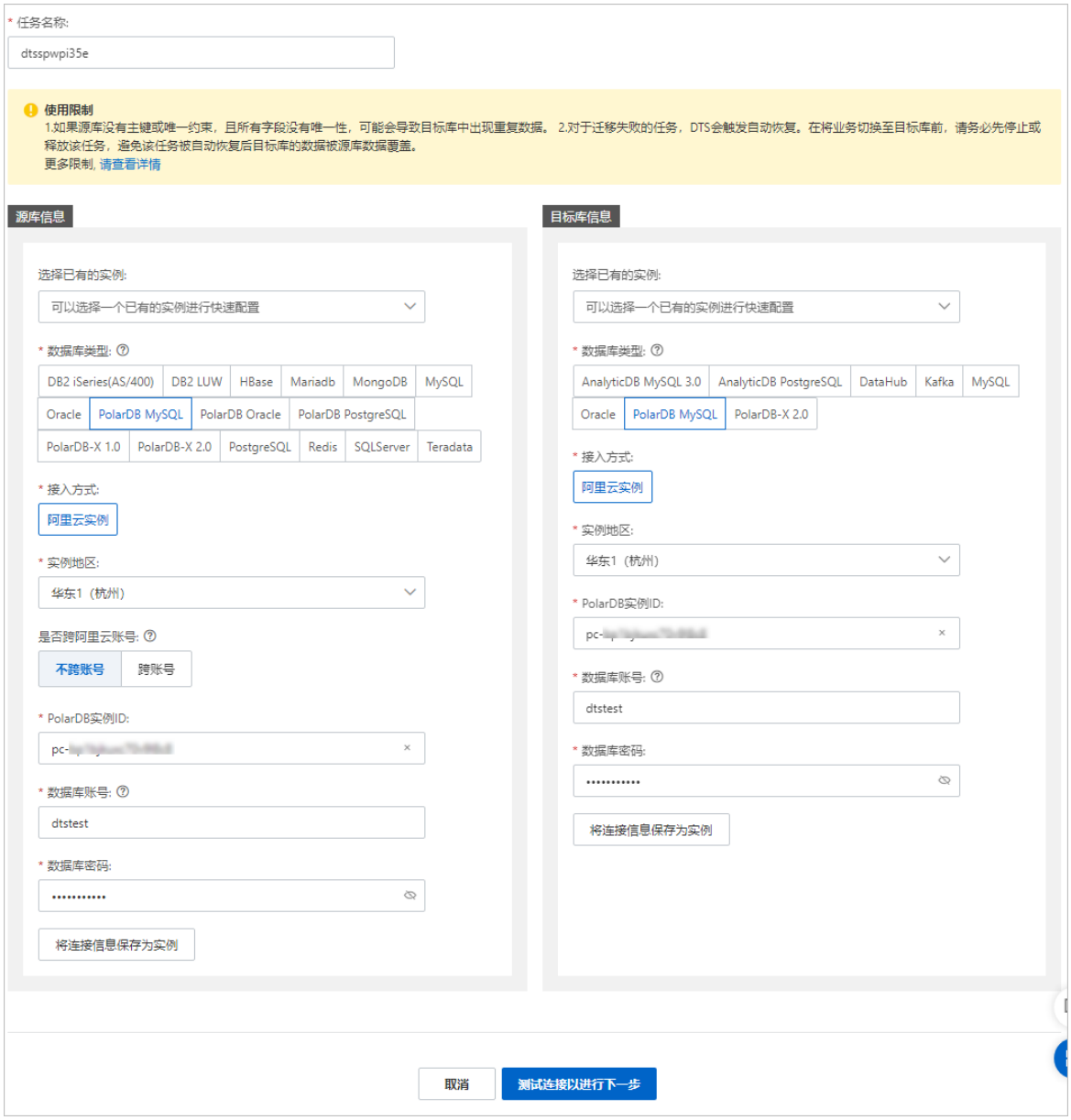
1. 登录新版DTS迁移任务的列表页面。

   > **说明** 您也可以登录DMS数据管理服务。在顶部菜单栏中，选择**集成与开发（DTS）** > **数据迁移**。

2. 在页面左上角，选择迁移实例所属地域。

3. 单击**创建任务**，配置源库及目标库信息。

   > **警告** 选择源和目标实例后，建议您仔细阅读页面上方显示的**使用限制**，以成功创建并执行迁移任务。



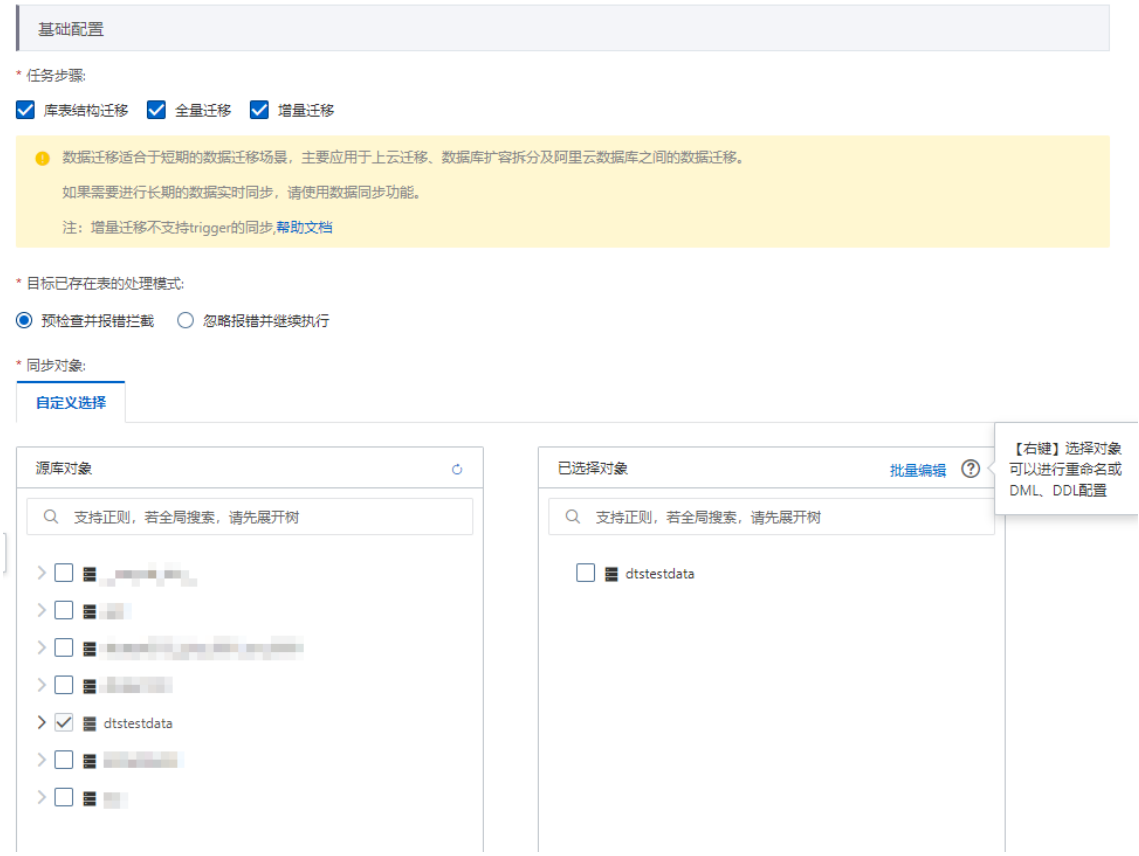
类别	配置	说明
无	**任务名称**	DTS会自动生成一个任务名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源库信息	**数据库类型**	选择**PolarDB MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择源PolarDB MySQL引擎集群所属地域。
	PolarDB MySQL实例ID	填入源PolarDB MySQL引擎集群ID。
	数据库账号	填入源PolarDB MySQL引擎集群的数据库账号，权限要求请参见数据库账号的权限要求。

类别	配置	说明
	数据库密码	填入该数据库账号对应的密码。
目标库信息	**数据库类型**	选择**PolarDB MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择目标PolarDB MySQL引擎集群所属地域。
	PolarDB MySQL实例ID	选择目标PolarDB MySQL引擎集群ID。
	数据库账号	填入目标PolarDB MySQL引擎集群的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号对应的密码。

4. 配置完成后，单击页面右下角的**测试连接以进行下一步**。

> **警告**
>
> - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段；如果源或目标数据库是IDC自建数据库或其他云数据库，则需要您手动添加对应地区DTS服务的IP地址，以允许来自DTS服务器的访问。
> - 上述场景中，DTS自动添加或您手动添加DTS服务的公网IP地址段可能会存在安全风险，一旦使用本产品代表您已理解和确认其中可能存在的安全风险，并且需要您做好基本的安全防护，包括但不限于加强账号密码强度防范、限制各网段开放的端口号、内部各API使用鉴权方式通信、定期检查并限制不需要的网段，或者使用通过内网（专线/VPN网关/智能网关）的方式接入。
> - DTS任务完成或释放后，建议您手动检测并删除DTS相关的服务器IP地址段。

5. 配置任务对象及高级配置。
   - **基础配置**



配置	说明
任务步骤	■ 如果只需要进行全量迁移，请同时勾选**库表结构迁移**和**全量迁移**。 ■ 如果需要进行不停机迁移，请同时勾选**库表结构迁移**、**全量迁移**和**增量迁移**。 **说明** 如果未选择**增量迁移**，为保障数据一致性，数据迁移期间请勿在源实例中写入新的数据。

配置	说明
目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据迁移任务不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以更改该表在目标库中的名称，请参见库表列名映射。 ■ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，在目标库遇到与源库主键的值相同的记录，则会保留目标库中的该条记录，即源库中的该条记录不会迁移至目标库中。 ■ 表结构不一致的情况下，可能导致只能迁移部分列的数据或迁移失败。
同步对象	在**同步对象**框中单击待迁移的对象，然后单击 › 将其移动到**已选择对象**框。 **说明**
映射名称更改	■ 如需更改单个迁移对象在目标实例中的名称，请右击**已选择对象**中的迁移对象，设置方式，请参见库表列名单个映射。 ■ 如需批量更改迁移对象在目标实例中的名称，请单击**已选择对象**方框右上方的 批量编辑 设置方式，请参见库表列名批量映射。 **说明** 如果使用了对象名映射功能，可能会导致依赖这个对象的其他对象迁移失败。
过滤待迁移数据	支持设置条件过滤数据，详情请参见通过SQL条件过滤任务数据。
增量迁移的SQL操作	选择增量迁移DDL或DML操作，请右击**已选择对象**中的迁移对象，在弹跳框中选择所需增量迁移的DML和DDL操作。支持的操作，请参见支持增量迁移的SQL操作。

- **高级配置**

配置	说明
设置告警	是否设置告警，当迁移失败或延迟超过阈值后，将通知告警联系人。 ■ **不设置**：不设置告警。 ■ **设置**：设置告警，您还需要设置告警阈值和告警联系人。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否迁移Online DDL变更产生的临时表数据。 ■ **是**：迁移Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致迁移任务延迟。 ■ **否**：不迁移Online DDL变更产生的临时表数据，只迁移源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。

配置	说明
源、目标库无法连接重试时间	默认重试120分钟，您也可以在取值范围（10~1440分钟）内自定义重试时间，建议设置30分钟以上。如果DTS在设置的时间内重新连接上源、目标库，迁移任务将自动恢复。否则，迁移任务将失败。 **说明** ■ 针对同源或者同目标的多个DTS实例，如DTS实例A和DTS实例B，设置网络重试时间时A设置30分钟，B设置60分钟，则重试时间以低的30分钟为准。 ■ 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

6. 上述配置完成后，单击页面右下角的**下一步保存任务并预检查**。

   说明
   - 在迁移任务正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动迁移任务。
   - 如果预检查失败，单击具体检查项后的，查看失败详情。
     - 您可以根据提示修复后重新进行预检查。
     - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

7. **预检查通过率**显示为**100%**时，单击**下一步购买**。
8. 在**购买**页面，选择数据迁移实例的链路规格，详细说明请参见下表。

类别	参数	说明
信息配置	**链路规格**	DTS为您提供了不同性能的迁移规格，迁移链路规格的不同会影响迁移速率，您可以根据业务场景进行选择，详情请参见数据迁移链路规格说明。

9. 配置完成后，阅读并勾选**《数据传输（按量付费）服务条款》**。
10. 单击**购买并启动**，迁移任务正式开始，您可在数据迁移界面查看具体进度。

3.5. 从其它数据库迁移至PolarDB

3.5.1. 自建MySQL迁移至PolarDB MySQL

通过数据传输服务DTS（Data Transmission Service），可以帮助您将自建MySQL数据库迁移至PolarDB MySQL引擎集群。

支持的源数据库

MySQL与PolarDB MySQL引擎集群间的迁移，支持源数据库MySQL为以下类型。本文以有公网IP的自建数据库为例介绍配置流程，其他类型的源数据库配置流程与本案例类似。

- RDS MySQL实例。
- 以下类型的自建数据库：
  - 有公网IP的自建数据库。
  - ECS上的自建数据库。
  - 通过专线、VPN网关或智能网关接入的自建数据库。
  - 通过数据库网关接入的自建数据库。

前提条件

- 自建MySQL数据库版本为5.1、5.5、5.6、5.7或8.0版本。
- 如果您的MySQL数据库部署在本地，那么您需要将DTS服务器的IP地址设置为该数据库远程连接的白名单，允许其访问您的数据库。详情请参见迁移、同步或订阅本地数据库时需添加的IP白名单。
- 已创建目标PolarDB MySQL引擎集群，详情请参见购买按量付费集群和购买包年包月集群。
- PolarDB MySQL引擎集群的存储空间须大于自建MySQL数据库占用的存储空间。

注意事项

类型	说明
源库限制	•
其他限制	•
特殊情况	• • 当目标库为PolarDB MySQL时 DTS会自动在PolarDB MySQL中创建数据库，如果待迁移的数据库名称不符合PolarDB MySQL的定义规范，您需要在配置迁移任务之前在PolarDB MySQL中创建数据库。相关操作，请参见管理数据库。

费用说明

迁移类型	链路配置费用	公网流量费用
结构迁移和全量数据迁移	不收费。	通过公网将数据迁移出阿里云时将收费，详情请参见产品定价。
增量数据迁移	收费，详情请参见产品定价。	

迁移类型说明

- 库表结构迁移

  DTS将源库中迁移对象的结构定义迁移到目标库。

> 说明
> - 目前DTS支持结构迁移的对象为表、视图、触发器、存储过程和存储函数。
> - 在结构迁移时，DTS会将视图、存储过程和函数中的DEFINER转换为INVOKER。
> - 由于DTS不迁移USER信息，因此在调用目标库的视图、存储您也可以过程和函数时需要对调用者授予读写权限。

- 全量迁移

  DTS将源库中迁移对象的存量数据，全部迁移到目标库中。

- 增量迁移

  DTS在全量迁移的基础上，将源库的增量更新数据迁移到目标库中。通过增量数据迁移可以实现在自建应用不停服的情况下，平滑地完成数据迁移。

支持增量迁移的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

数据库账号的权限要求

数据库	库表结构迁移	全量迁移	增量迁移
自建MySQL数据库	SELECT权限	SELECT权限	增量数据迁移：待迁移对象的SELECT权限 REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW 建库建表的权限，以允许DTS创建库dts，用于记录迁移期间的心跳数据
PolarDB MySQL引擎集群	读写权限		

数据库账号创建及授权方法：

- 自建MySQL数据库请参见为自建MySQL创建账号并设置binlog。
- PolarDB MySQL引擎集群请参见创建数据库账号。

操作步骤

1. 登录新版DTS迁移任务的列表页面。

> **说明** 您也可以登录DMS数据管理服务。在顶部菜单栏中，选择**集成与开发（DTS）** > **数据迁移**。

2. 在页面左上角，选择迁移实例所属地域。

3. 单击**创建任务**，配置源库及目标库信息。

> **警告** 选择源和目标实例后，建议您仔细阅读页面上方显示的**使用限制**，以成功创建并执行迁移任务。

类别	配置	说明
无	**任务名称**	DTS会自动生成一个任务名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
	数据库类型	选择MySQL。

类别	配置	说明
源库信息	接入方式	根据源库的部署位置进行选择，本文以**有公网IP的自建数据库**为例介绍配置流程。 **说明** 当自建数据库为其他实例类型时，您还需要执行相应的准备工作，详情请参见准备工作概览。
	实例地区	选择源自建MySQL数据库所属地域。
	主机名或IP地址	填入源自建MySQL数据库的访问地址，本案例中填入公网地址。
	端口	填入源自建MySQL数据库的服务端口（需开放至公网），默认为3306。
	数据库账号	填入源自建MySQL数据库的账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号对应的密码。
目标库信息		
	数据库类型	选择PolarDB MySQL。
	接入方式	选择**阿里云实例**。
	实例地区	选择目标PolarDB MySQL引擎集群所属地域。
	PolarDB MySQL实例ID	选择目标PolarDB MySQL引擎集群ID。
	数据库账号	填入目标PolarDB MySQL引擎集群的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号对应的密码。

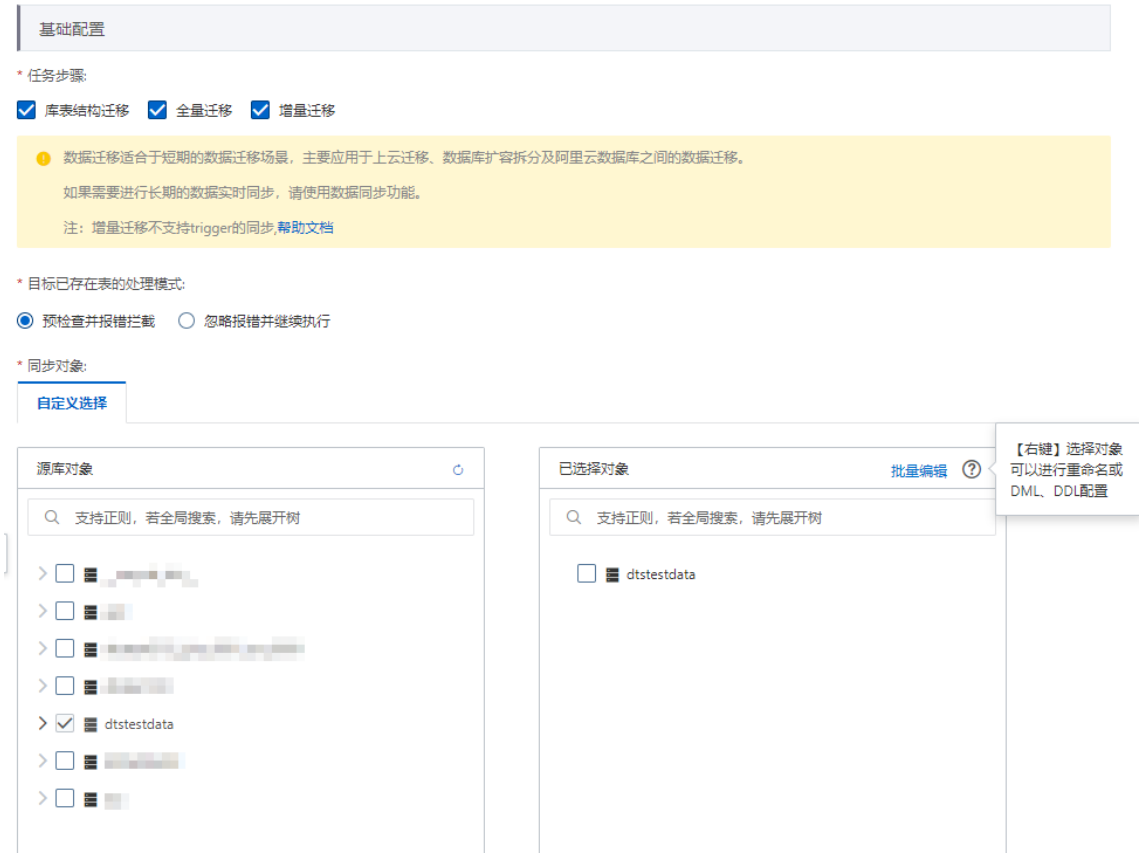
4. 配置完成后，单击页面右下角的**测试连接以进行下一步**。

> 警告
>
> - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段；如果源或目标数据库是IDC自建数据库或其他云数据库，则需要您手动添加对应地区DTS服务的IP地址，以允许来自DTS服务器的访问。
> - 上述场景中，DTS自动添加或您手动添加DTS服务的公网IP地址段可能会存在安全风险，一旦使用本产品代表您已理解和确认其中可能存在的安全风险，并且需要您做好基本的安全防护，包括但不限于加强账号密码强度防范、限制各网段开放的端口号、内部各API使用鉴权方式通信、定期检查并限制不需要的网段，或者使用通过内网（专线/VPN网关/智能网关）的方式接入。
> - DTS任务完成或释放后，建议您手动检测并删除DTS相关的服务器IP地址段。

5. 如果您的自建数据库具备白名单安全设置，您需要复制弹跳框中的DTS服务器IP地址，并加入自建数据库的白名单安全设置中。然后单击**测试连接以进行下一步**。

> 警告

6. 配置任务对象及高级配置。
   - **基础配置**

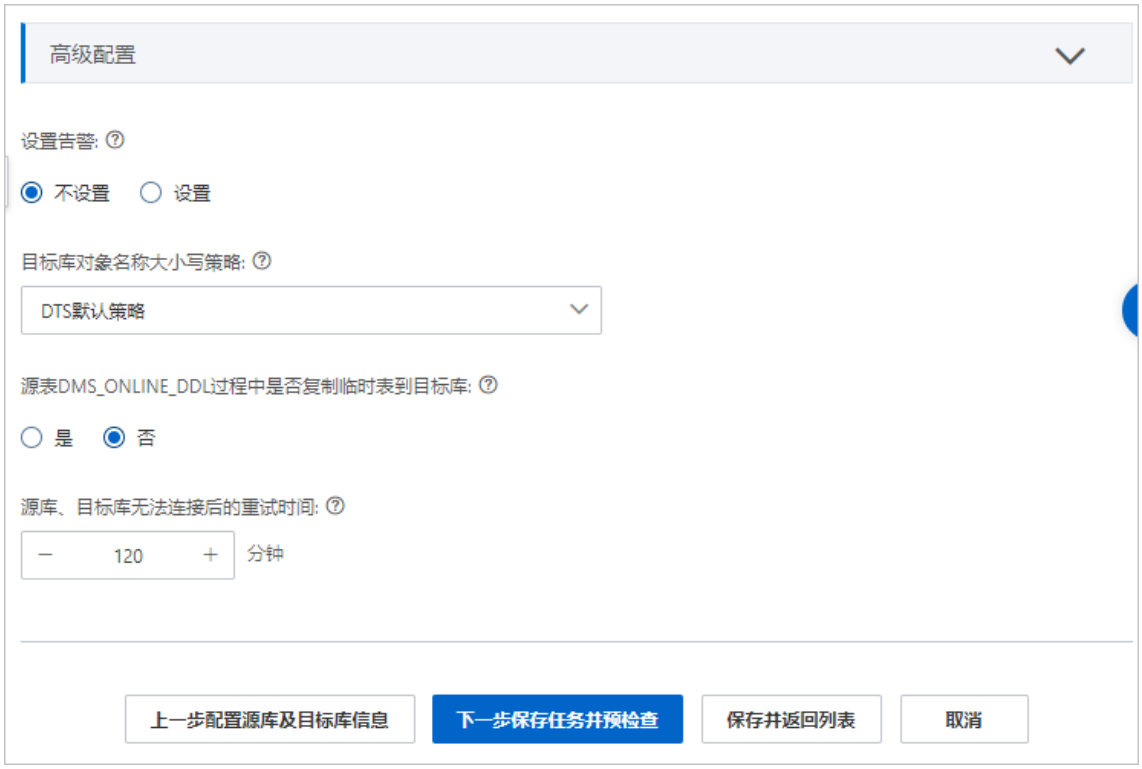


配置	说明

配置	说明
任务步骤	■ 如果只需要进行全量迁移，请同时勾选**库表结构迁移**和**全量迁移**。 ■ 如果需要进行不停机迁移，请同时勾选**库表结构迁移**、**全量迁移**和**增量迁移**。 **说明** 如果未选择**增量迁移**，为保障数据一致性，数据迁移期间请勿在源实例中写入新的数据。
目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据迁移任务不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以更改该表在目标库中的名称，请参见库表列名映射。 ■ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，在目标库遇到与源库主键的值相同的记录，则会保留目标库中的该条记录，即源库中的该条记录不会迁移至目标库中。 ■ 表结构不一致的情况下，可能导致只能迁移部分列的数据或迁移失败。
同步对象	在**同步对象**框中单击待迁移的对象，然后单击 › 将其移动到**已选择对象**框。 **说明**
映射名称更改	■ 如需更改单个迁移对象在目标实例中的名称，请右击**已选择对象**中的迁移对象，设置方式，请参见库表列名单个映射。 ■ 如需批量更改迁移对象在目标实例中的名称，请单击**已选择对象**方框右上方的 批量编辑 设置方式，请参见库表列名批量映射。 **说明** 如果使用了对象名映射功能，可能会导致依赖这个对象的其他对象迁移失败。
过滤待迁移数据	支持设置条件过滤数据，详情请参见通过SQL条件过滤任务数据。

配置	说明
增量迁移的SQL操作	选择增量迁移DDL或DML操作，请右击**已选择对象**中的迁移对象，在弹跳框中选择所需增量迁移的DML和DDL操作。支持的操作，请参见支持增量迁移的SQL操作。

- **高级配置**



配置	说明
设置告警	是否设置告警，当迁移失败或延迟超过阈值后，将通知告警联系人。 ■ **不设置**：不设置告警。 ■ **设置**：设置告警，您还需要设置告警阈值和告警联系人。
目标库对象名称大小写策略	您可以配置目标实例中迁移对象的库名、表名和列名的英文大小写策略。默认情况下选择**DTS默认策略**，您也可以选择与源库、目标库默认策略保持一致。更多信息，请参见目标库对象名称大小写策略。

配置	说明
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否迁移Online DDL变更产生的临时表数据。 ■ **是**：迁移Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致迁移任务延迟。 ■ **否**：不迁移Online DDL变更产生的临时表数据，只迁移源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源、目标库无法连接重试时间	默认重试120分钟，您也可以在取值范围（10~1440分钟）内自定义重试时间，建议设置30分钟以上。如果DTS在设置的时间内重新连接上源、目标库，迁移任务将自动恢复。否则，迁移任务将失败。 **说明** ■ 针对同源或者同目标的多个DTS实例，如DTS实例A和DTS实例B，设置网络重试时间时A设置30分钟，B设置60分钟，则重试时间以低的30分钟为准。 ■ 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

7. 上述配置完成后，单击页面右下角的**下一步保存任务并预检查**。

   说明
   - 在迁移任务正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动迁移任务。
   - 如果预检查失败，单击具体检查项后的，查看失败详情。
     - 您可以根据提示修复后重新进行预检查。
     - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

8. **预检查通过率**显示为**100%**时，单击**下一步购买**。
9. 在**购买**页面，选择数据迁移实例的链路规格，详细说明请参见下表。

类别	参数	说明
信息配置	**链路规格**	DTS为您提供了不同性能的迁移规格，迁移链路规格的不同会影响迁移速率，您可以根据业务场景进行选择，详情请参见数据迁移链路规格说明。

10. 配置完成后，阅读并勾选**《数据传输（按量付费）服务条款》**。
11. 单击**购买并启动**，迁移任务正式开始，您可在数据迁移界面查看具体进度。

3.5.2. 从自建MySQL迁移至PolarDB MySQL引擎（mysqldump工具）

本文介绍如何使用mysqldump工具将自建MySQL数据库迁移至PolarDB MySQL引擎。

前提条件

目标PolarDB MySQL引擎集群需已完成如下操作：

- 创建数据库
- 创建数据库账号
- 设置白名单
- 申请公网连接地址

迁移方式对比

您可以通过mysqldump或DTS工具将自建MySQL数据库迁移至PolarDB MySQL引擎，下表对比了两种迁移方式的差异供您参考。

对比项	mysqldump	DTS
自建MySQL数据库版本	无限制	自建MySQL数据库版本为5.1、5.5、5.6、5.7或8.0版本。
是否支持结构迁移和全量数据迁移	支持	支持
是否支持增量数据迁移	不支持	支持
是否支持不停机迁移	不支持	支持

> **说明** 使用DTS工具进行数据迁移的操作步骤，请参见从自建MySQL迁移至PolarDB MySQL。

注意事项

迁移后的表名不区分大小写，统一变为小写。

操作步骤

> **说明** 本文以自建MySQL 8.0版本为例，在Linux系统上演示相关操作步骤。

1. 使用mysqldump导出自建数据库的数据、存储过程、触发器和函数。

   > **说明** 导出期间请勿进行数据更新，耐心等待导出完成。

i. 在Linux命令行下导出自建数据库的数据，命令如下：

> **说明** 针对自建数据库的连接地址：
> - 若自建MySQL数据库部署在ECS实例上，请填入 `127.0.0.1` 。
> - 若自建MySQL数据库部署在本地，请填入该数据库的公网连接地址。

```
mysqldump -h <自建数据库的连接地址>  -u root -p --opt --default-character-set=utf8 --hex-blob <自建数据库名> --skip-triggers --skip-lock-tables > /tmp/<自建数据库名>.sql
```

示例

```
mysqldump -h 127.0.0.1 -u root -p --opt --default-character-set=utf8 --hex-blob testdb --skip-triggers --skip-lock-tables > /tmp/testdb.sql
```

ii. （可选）在Linux命令行下导出存储过程、触发器和函数，命令如下：

> **说明** 若数据库中没有使用存储过程、触发器和函数，可跳过该步骤。

```
mysqldump -h 127.0.0.1 -u root -p --opt --default-character-set=utf8 --hex-blob <自建数据库名> -R | sed -e 's/DEFINER[ ]*=[ ]*[^*]*\*/\*/' > /tmp/<自建数据库名>Trigger.sql
```

示例

```
mysqldump -h 127.0.0.1 -u root -p --opt --default-character-set=utf8 --hex-blob testdb -R | sed -e 's/DEFINER[ ]*=[ ]*[^*]*\*/\*/' > /tmp/testdbTrigger.sql
```

2. 将导出的文件导入到目标PolarDB集群中，命令如下：

```
mysql -h <PolarDB集群连接地址> -P <PolarDB集群端口> -u <PolarDB集群账号> -p <PolarDB数据库名称> < /tmp/<自建数据库名>.sql
mysql -h <PolarDB集群连接地址> -P <PolarDB集群端口> -u <PolarDB集群账号> -p <PolarDB数据库名称> < /tmp/<自建数据库名>Trigger.sql
```

示例

```
mysql -h polardbtest.mysql.polardb.rds.aliyuncs.com -P 3306 -u testuser -p testdb  < /tmp/testdb.sql
mysql -h polardbtest.mysql.polardb.rds.aliyuncs.com -P 3306 -u testuser -p testdb  < /tmp/testdbTrigger.sql
```

> **说明**
> - PolarDB数据库名称需要是PolarDB集群上已创建的数据库。创建数据库操作，请参见创建数据库。
> - PolarDB集群账号需要是高权限账号或具有读写权限的账号。

3. 导入成功后，您可以登录PolarDB集群数据库中查看数据是否正常。具体操作，请参见连接数据库集群。

常见问题

Q：`Access denied; you need (at least one of) the SUPER privilege(s) for this operation` 报错怎么解决？

A：SQL脚本里面包括SUPER权限的语句，将相关语句删除再执行。

3.5.3. 从Amazon Aurora MySQL迁移至PolarDB MySQL

本文介绍如何使用数据传输服务DTS（Data Transmission Service），将Amazon Aurora MySQL迁移至阿里云PolarDB MySQL。DTS支持结构迁移、全量数据迁移以及增量数据迁移，同时使用这三种迁移类型可以实现在自建应用不停服的情况下，平滑地完成数据库迁移。

前提条件

- 为保障DTS可以通过公网连接至Amazon Aurora MySQL，Amazon Aurora MySQL的网络与安全配置中须将公开可用性功能设置为**是**。
- 已创建阿里云PolarDB MySQL集群，详情请参见PolarDB MySQL集群。
- 阿里云PolarDB MySQL的存储空间须大于Amazon Aurora MySQL已使用的存储空间。

注意事项

- DTS在执行全量数据迁移时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据迁移前评估源库和目标库的性能，同时建议您在业务低峰期执行数据迁移（例如源库和目标库的CPU负载在30%以下）。
- 如果源数据库没有主键或唯一约束，且所有字段没有唯一性，可能会导致目标数据库中出现重复数据。
- 对于数据类型为FLOAT或DOUBLE的列，DTS会通过 `ROUND(COLUMN,PRECISION)` 来读取该列的值。如果没有明确定义其精度，DTS对FLOAT的迁移精度为38位，对DOUBLE的迁移精度为308位，请确认迁移精度是否符合业务预期。
- 如果待迁移数据库名称不符合阿里云PolarDB MySQL的定义规范，您需要在配置迁移任务之前在阿里云PolarDB MySQL中创建数据库。

  > **说明** 关于阿里云PolarDB MySQL的定义规范和创建数据库的操作方法，请参见创建数据库。

- 对于迁移失败的任务，DTS会触发自动恢复。在您将业务切换至目标实例前，请务必先结束或释放该任务，避免该任务被自动恢复后，导致源端数据覆盖目标实例的数据。

费用说明

迁移类型	链路配置费用	公网流量费用
结构迁移和全量数据迁移	不收费。	通过公网将数据迁移出阿里云时将收费，详情请参见产品定价。
增量数据迁移	收费，详情请参见产品定价。	

迁移类型说明

- 结构迁移

DTS将待迁移对象的结构定义迁移到阿里云PolarDB MySQL，目前DTS支持结构迁移的对象为表、视图、触发器、存储过程、存储函数，不支持event的结构迁移。

> **说明**
> - 在结构迁移时，DTS会将视图、存储过程和函数中的DEFINER转换为INVOKER。
> - 由于DTS不迁移user信息，因此在调用目标库的视图、存储过程和函数时需要对调用者授予读写权限。

- 全量数据迁移

  DTS会将Amazon Aurora MySQL中待迁移对象的存量数据，全部迁移到阿里云PolarDB MySQL中。

  > **说明** 由于全量数据迁移会并发INSERT导致目标实例的表存在碎片，全量迁移完成后目标实例的表空间会比源实例大。

- 增量数据迁移

  在全量迁移的基础上，DTS会读取Amazon Aurora MySQL的binlog信息，将Amazon Aurora MySQL的增量更新数据同步到阿里云PolarDB MySQL中。通过增量数据迁移可以实现在应用不停服的情况下，平滑地完成MySQL数据库的迁移。

数据库账号的权限要求

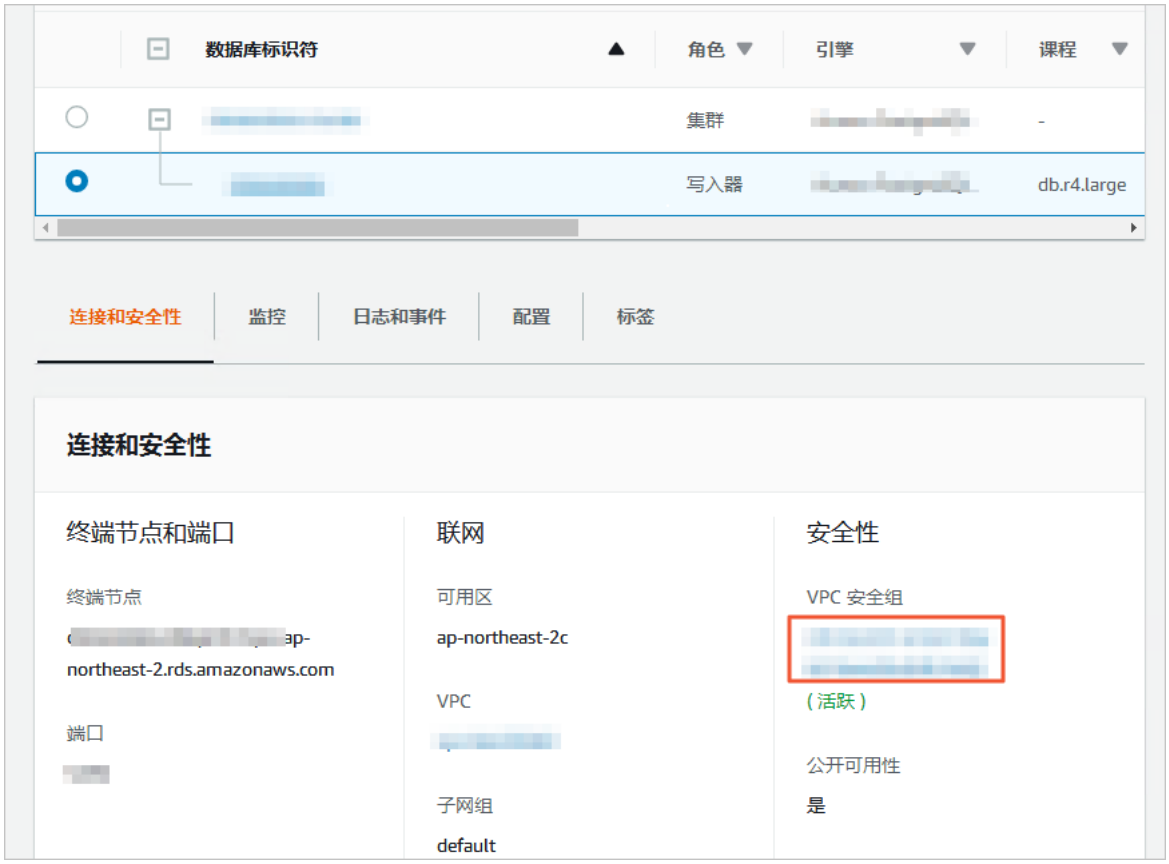
数据库	结构迁移	全量迁移	增量迁移
Amazon Aurora MySQL	迁移对象的SELECT权限	迁移对象的SELECT权限	REPLICATION SLAVE、REPLICATION CLIENT、SHOW VIEW和对迁移对象执行SELECT操作的权限
阿里云PolarDB MySQL	迁移对象的读写权限	迁移对象的读写权限	迁移对象的读写权限

数据库账号创建及授权方法：

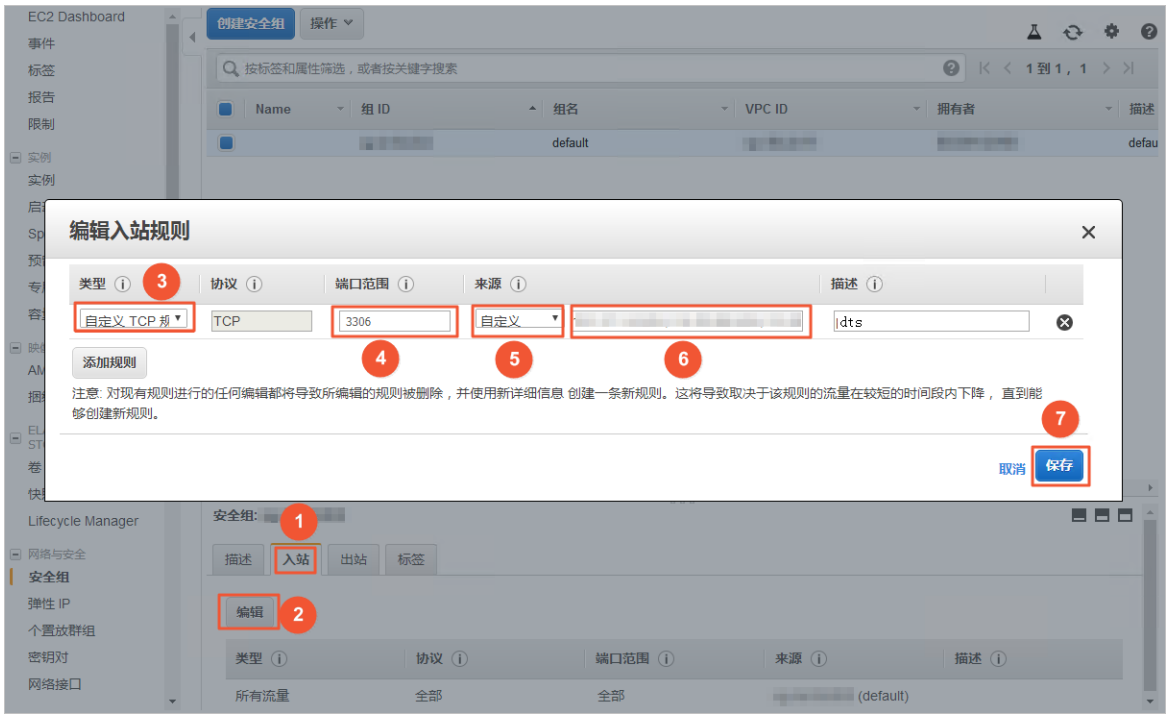
- Amazon Aurora MySQL请参见为自建MySQL创建账号并设置binlog中创建账号的部分。
- 阿里云PolarDB MySQL请参见创建账号。

迁移前准备工作

1. 登录Amazon Aurora控制台。
2. 进入Amazon Aurora MySQL的**基本信息**页面。
3. 选择角色为**写入器**的节点。
4. 在**连接和安全性**区域框，单击对应的VPC安全组名称。



5. 在**安全组设置**页面，将对应区域的DTS服务器地址添加至入站规则中，IP地址段详情请参见迁移、同步或订阅本地数据库时需添加的IP白名单。



> **说明**
>
> - 您只需添加目标数据库所在区域对应的DTS IP地址段。例如，源数据库地区为新加坡，目标数据库地区为杭州，您只需要添加杭州地区的DTS IP地址段。
> - 在加入IP地址段时，您可以一次性添加所需的IP地址，无需逐条添加入站规则。

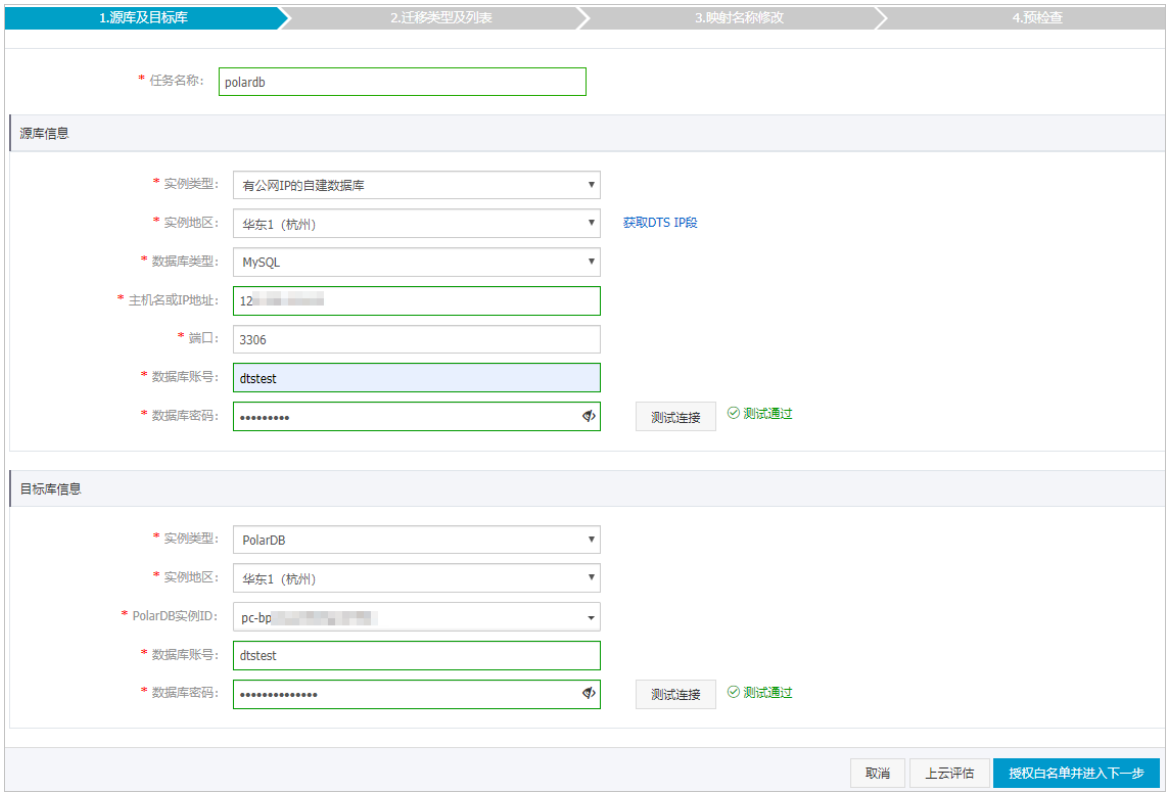
6. 登录Amazon Aurora MySQL数据库，设置binlog日志保存时间。如果不需要增量数据迁移，可跳过本步骤。

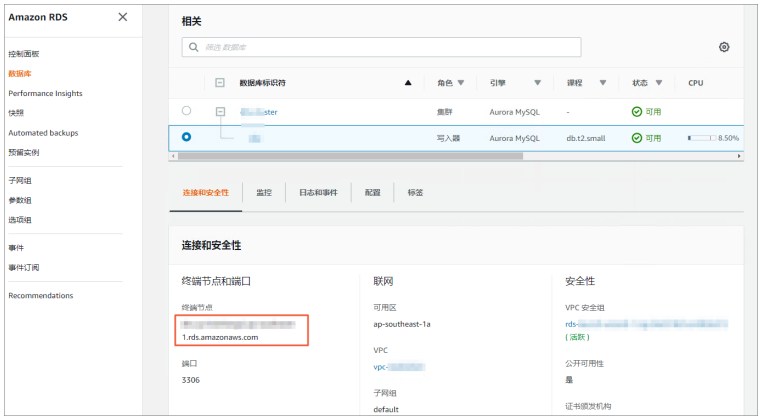
```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **说明**
>
> - 上述命令将binlog日志的保存设置为24小时，最大可设置为168个小时，即7天。
> - Amazon Aurora MySQL的binlog日志需处于开启状态，且binlog_format需设置为row；当MySQL为5.6及以上版本时，binlog_row_image需设置为full。

操作步骤

1. 登录数据传输控制台。
2. 在左侧导航栏，单击**数据迁移**。
3. 在**迁移任务列表**页面顶部，选择迁移的目标集群所属地域。
4. 单击页面右上角的**创建迁移任务**。
5. 配置迁移任务的源库及目标库信息。



类别	配置	说明
无	任务名称	DTS会自动生成一个任务名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源库信息	实例类型	选择**有公网IP的自建数据库**。
	实例地区	当实例类型选择为**有公网IP的自建数据库**时，**实例地区**无需设置。
	数据库类型	选择**MySQL**。
	主机名或IP地址	填入Amazon Aurora MySQL的访问地址 。 **说明** 您可以在Amazon Aurora MySQL的**基本信息**页面，获取数据库的访问地址。 
	端口	填入Amazon Aurora MySQL的服务端口，默认为**3306**。
	数据库账号	填入Amazon Aurora MySQL的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号的密码。 **说明** 源库信息填写完毕后，您可以单击**数据库密码**后的**测试连接**来验证填入的信息是否正确。如果填写正确则提示**测试通过**；如果提示**测试失败**，单击**测试失败**后的**诊断**，根据提示调整填写的源库信息。
	实例类型	选择**PolarDB**。
	实例地区	选择阿里云PolarDB MySQL实例所属地域。

类别	配置	说明
目标库信息	PolarDB实例ID	选择阿里云PolarDB MySQL实例ID。
	数据库账号	填入阿里云PolarDB MySQL的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号的密码。 **说明** 目标库信息填写完毕后，您可以单击**数据库密码**后的**测试连接**来验证填入的信息是否正确。如果填写正确则提示**测试通过**；如果提示**测试失败**，单击**测试失败**后的**诊断**，根据提示调整填写的目标库信息。

6. 配置完成后，单击页面右下角的**授权白名单并进入下一步**。

   说明 此步骤会将DTS服务器的IP地址自动添加到阿里云PolarDB MySQL的白名单中，用于保障DTS服务器能够正常连接阿里云PolarDB MySQL。

7. 选择迁移对象及迁移类型。

配置	说明
迁移类型	◦ 如果只需要进行全量迁移，则同时勾选**结构迁移**和**全量数据迁移**。 ◦ 如果需要进行不停机迁移，则同时勾选**结构迁移**、**全量数据迁移**和**增量数据迁移**。 **说明** 如果未选择**增量数据迁移**，为保障数据一致性，数据迁移期间请勿在Amazon Aurora MySQL中写入新的数据。

配置	说明
迁移对象	在**迁移对象**框中单击待迁移的对象，然后单击 › 将其移动至**已选择对象**框。 **说明** ○ 迁移对象选择的粒度为库、表、列。 ○ 默认情况下，迁移对象在目标库中的名称与源库保持一致。如果您需要改变迁移对象在目标库中的名称，需要使用对象名映射功能，详情请参见库表列映射。 ○ 如果使用了对象名映射功能，可能会导致依赖这个对象的其他对象迁移失败。
映射名称更改	如需更改迁移对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。
源、目标库无法连接重试时间	默认重试12小时，您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，迁移任务将自动恢复。否则，迁移任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否迁移Online DDL变更产生的临时表数据。 ○ **是**：迁移Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致迁移任务延迟。 ○ **否**：不迁移Online DDL变更产生的临时表数据，只迁移源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。

8. 上述配置完成后，单击页面右下角的**预检查并启动**。

说明

- 在迁移任务正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动迁移任务。
- 如果预检查失败，单击具体检查项后的，查看失败详情。
  - 您可以根据提示修复后重新进行预检查。
  - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

9. 预检查通过后，单击**下一步**。
10. 在弹出的**购买配置确认**对话框，选择**链路规格**并选中**数据传输（按量付费）服务条款**。

11. 单击**购买并启动**，迁移任务正式开始。
    - 结构迁移+全量数据迁移

      请勿手动结束迁移任务，否则可能会导致数据不完整。您只需等待迁移任务完成即可，迁移任务会自动结束。
    - 结构迁移+全量数据迁移+增量数据迁移

      迁移任务不会自动结束，您需要手动结束迁移任务。

      > **注意** 请选择合适的时间手动结束迁移任务，例如业务低峰期或准备将业务切换至目标集群时。

      a. 观察迁移任务的进度变更为**增量迁移**，并显示为**无延迟**状态时，将源库停写几分钟，此时**增量迁移**的状态可能会显示延迟的时间。

      b. 等待迁移任务的**增量迁移**再次进入**无延迟**状态后，手动结束迁移任务。

12. 将业务切换至阿里云PolarDB MySQL。

3.6. 从PolarDB迁移至其它数据库

3.6.1. PolarDB MySQL迁移至RDS MySQL

通过数据传输服务DTS（Data Transmission Service），您可以将PolarDB MySQL迁移至MySQL（包括自建MySQL、RDS MySQL实例）。

支持的目标数据库

PolarDB MySQL迁移至MySQL，支持目标数据库为以下类型。本文以RDS MySQL实例为例介绍配置流程，其他类型的目标数据库的配置流程与本案例类似。

- RDS MySQL实例。
- 以下类型的自建数据库：
  - 有公网IP的自建数据库。
  - ECS上的自建数据库。
  - 通过专线、VPN网关或智能网关接入的自建数据库。
  - 通过数据库网关接入的自建数据库。

前提条件

- 已创建源PolarDB MySQL引擎集群，详情请参见购买按量付费集群和购买包年包月集群。
- 目标RDS MySQL实例的存储空间须大于源集群占用的存储空间。

注意事项

类型	说明
源库限制	•
注意事项	• • • • •
特殊情况	当目标库为RDS MySQL时，DTS会自动在RDS MySQL中创建数据库，如果待迁移的数据库名称不符合RDS MySQL的定义规范，您需要在配置迁移任务之前在RDS MySQL中创建数据库。相关操作，请参见创建数据库。

费用说明

迁移类型	链路配置费用	公网流量费用
结构迁移和全量数据迁移	不收费。	通过公网将数据迁移出阿里云时将收费，详情请参见产品定价。
增量数据迁移	收费，详情请参见产品定价。	

迁移类型说明

- 库表结构迁移

  DTS将源库中迁移对象的结构定义迁移到目标库。

  > **说明**
  > - 目前DTS支持结构迁移的对象为表、视图、触发器、存储过程和存储函数。
  > - 在结构迁移时，DTS会将视图、存储过程和函数中的DEFINER转换为INVOKER。
  > - 由于DTS不迁移USER信息，因此在调用目标库的视图、存储您也可以过程和函数时需要对调用者授予读写权限。

- 全量迁移

  DTS将源库中迁移对象的存量数据，全部迁移到目标库中。

- 增量迁移

  DTS在全量迁移的基础上，将源库的增量更新数据迁移到目标库中。通过增量数据迁移可以实现在自建应用不停服的情况下，平滑地完成数据迁移。

支持增量迁移的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE

操作类型	SQL操作语句
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

数据库账号的权限要求

数据库	权限要求
PolarDB MySQL引擎集群	待迁移对象的读权限
RDS MySQL实例	迁移对象的读写权限

数据库账号创建及授权方法：

- PolarDB MySQL引擎集群请参见创建数据库账号。
- RDS MySQL实例请参见创建账号和修改账号权限。

操作步骤

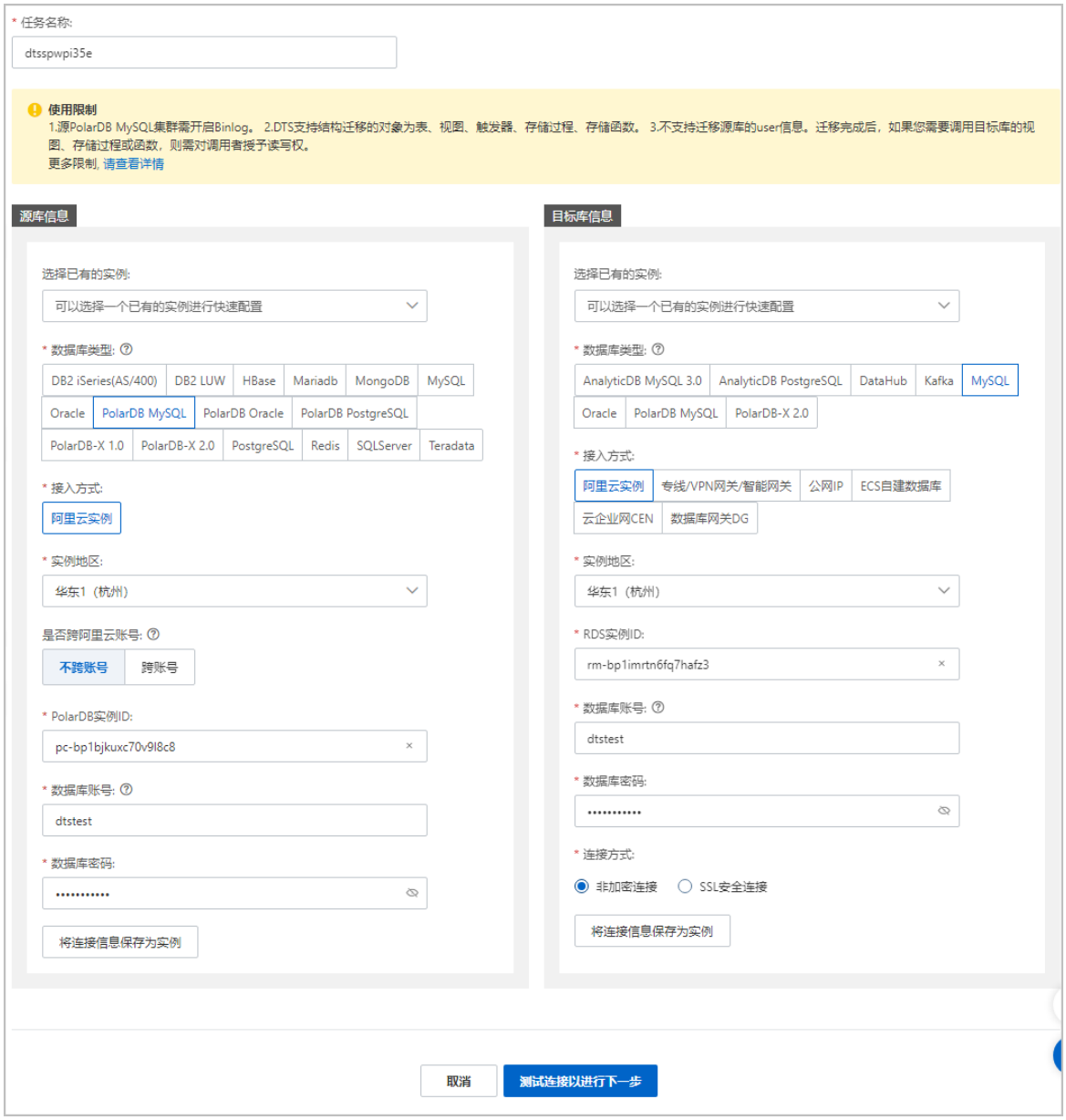
1. 登录新版DTS迁移任务的列表页面。

   > **说明** 您也可以登录DMS数据管理服务。在顶部菜单栏中，选择**集成与开发（DTS）** > **数据迁移**。

2. 在页面左上角，选择迁移实例所属地域。

3. 单击**创建任务**，配置源库及目标库信息。

   > **警告** 选择源和目标实例后，建议您仔细阅读页面上方显示的**使用限制**，以成功创建并执行迁移任务。



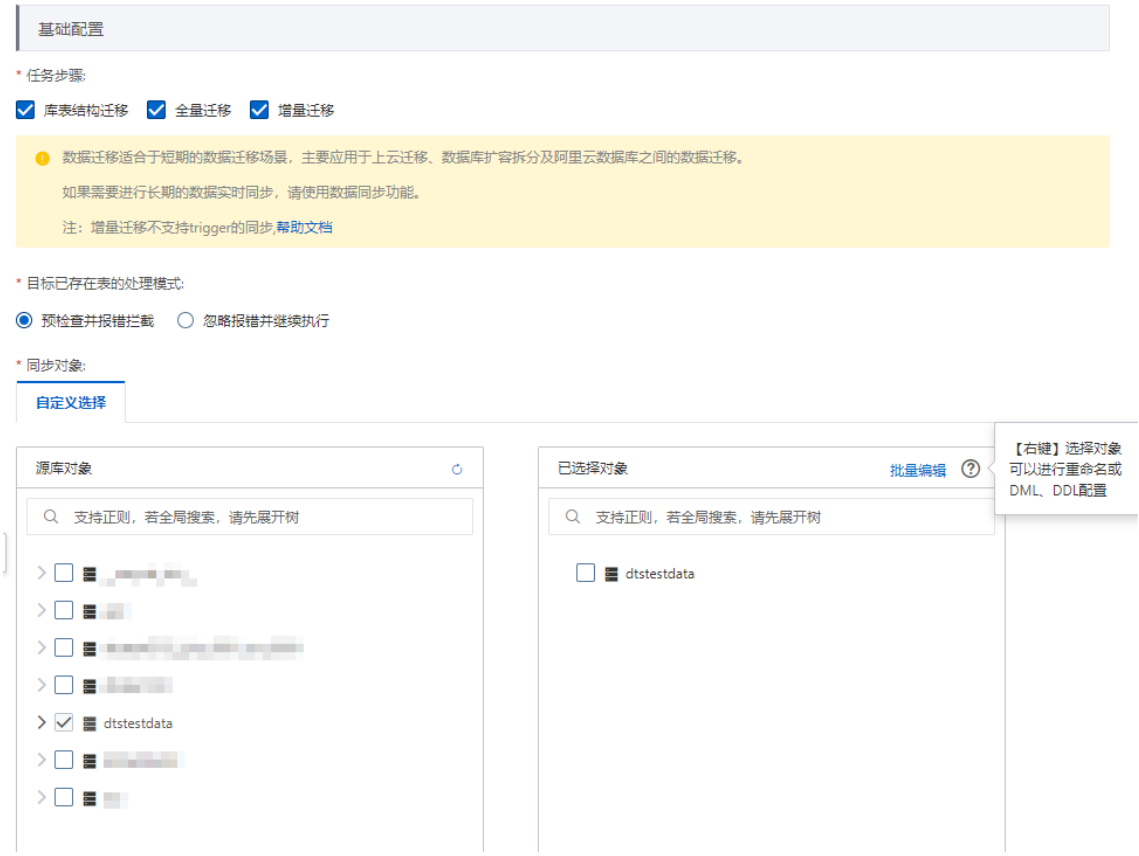
类别	配置	说明
无	**任务名称**	DTS会自动生成一个任务名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源库信息	**数据库类型**	选择**PolarDB MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择源PolarDB MySQL引擎集群所属地域。
	PolarDB MySQL实例ID	填入源PolarDB MySQL引擎集群ID。
	数据库账号	填入源PolarDB MySQL引擎集群的数据库账号，权限要求请参见数据库账号的权限要求。

类别	配置	说明
	数据库密码	填入该数据库账号对应的密码。
目标库信息	**数据库类型**	选择**MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择目标RDS MySQL实例所属地域。
	RDS实例ID	选择目标RDS MySQL实例ID。
	数据库账号	填入目标RDS MySQL实例的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入该数据库账号对应的密码。
	连接方式	根据需求选择**非加密连接**或**SSL安全连接**。如果设置为**SSL安全连接**，您需要提前开启RDS MySQL实例的SSL加密功能，详情请参见设置SSL加密。

4. 配置完成后，单击页面右下角的**测试连接以进行下一步**。

> **警告**
>
> - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段；如果源或目标数据库是IDC自建数据库或其他云数据库，则需要您手动添加对应地区DTS服务的IP地址，以允许来自DTS服务器的访问。
> - 上述场景中，DTS自动添加或您手动添加DTS服务的公网IP地址段可能会存在安全风险，一旦使用本产品代表您已理解和确认其中可能存在的安全风险，并且需要您做好基本的安全防护，包括但不限于加强账号密码强度防范、限制各网段开放的端口号、内部各API使用鉴权方式通信、定期检查并限制不需要的网段，或者使用通过内网（专线/VPN网关/智能网关）的方式接入。
> - DTS任务完成或释放后，建议您手动检测并删除DTS相关的服务器IP地址段。

5. 配置任务对象及高级配置。
   - **基础配置**



配置	说明
任务步骤	■ 如果只需要进行全量迁移，请同时勾选**库表结构迁移**和**全量迁移**。 ■ 如果需要进行不停机迁移，请同时勾选**库表结构迁移**、**全量迁移**和**增量迁移**。 **说明** 如果未选择**增量迁移**，为保障数据一致性，数据迁移期间请勿在源实例中写入新的数据。

配置	说明
目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据迁移任务不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以更改该表在目标库中的名称，请参见库表列名映射。 ■ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，在目标库遇到与源库主键的值相同的记录，则会保留目标库中的该条记录，即源库中的该条记录不会迁移至目标库中。 ■ 表结构不一致的情况下，可能导致只能迁移部分列的数据或迁移失败。
同步对象	在**同步对象**框中单击待迁移的对象，然后单击 › 将其移动到**已选择对象**框。 **说明**
映射名称更改	■ 如需更改单个迁移对象在目标实例中的名称，请右击**已选择对象**中的迁移对象，设置方式，请参见库表列名单个映射。 ■ 如需批量更改迁移对象在目标实例中的名称，请单击**已选择对象**方框右上方的 批量编辑 设置方式，请参见库表列名批量映射。 **说明** 如果使用了对象名映射功能，可能会导致依赖这个对象的其他对象迁移失败。
过滤待迁移数据	支持设置条件过滤数据，详情请参见通过SQL条件过滤任务数据。
增量迁移的SQL操作	选择增量迁移DDL或DML操作，请右击**已选择对象**中的迁移对象，在弹跳框中选择所需增量迁移的DML和DDL操作。支持的操作，请参见支持增量迁移的SQL操作。

- **高级配置**

配置	说明
设置告警	是否设置告警，当迁移失败或延迟超过阈值后，将通知告警联系人。 ■ **不设置**：不设置告警。 ■ **设置**：设置告警，您还需要设置告警阈值和告警联系人。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否迁移Online DDL变更产生的临时表数据。 ■ **是**：迁移Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致迁移任务延迟。 ■ **否**：不迁移Online DDL变更产生的临时表数据，只迁移源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。

配置	说明
源、目标库无法连接重试时间	默认重试120分钟，您也可以在取值范围（10~1440分钟）内自定义重试时间，建议设置30分钟以上。如果DTS在设置的时间内重新连接上源、目标库，迁移任务将自动恢复。否则，迁移任务将失败。 **说明** ■ 针对同源或者同目标的多个DTS实例，如DTS实例A和DTS实例B，设置网络重试时间时A设置30分钟，B设置60分钟，则重试时间以低的30分钟为准。 ■ 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

6. 上述配置完成后，单击页面右下角的**下一步保存任务并预检查**。

   说明
   - 在迁移任务正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动迁移任务。
   - 如果预检查失败，单击具体检查项后的，查看失败详情。
     - 您可以根据提示修复后重新进行预检查。
     - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

7. **预检查通过率**显示为**100%**时，单击**下一步购买**。
8. 在**购买**页面，选择数据迁移实例的链路规格，详细说明请参见下表。

类别	参数	说明
信息配置	**链路规格**	DTS为您提供了不同性能的迁移规格，迁移链路规格的不同会影响迁移速率，您可以根据业务场景进行选择，详情请参见数据迁移链路规格说明。

9. 配置完成后，阅读并勾选**《数据传输（按量付费）服务条款》**。
10. 单击**购买并启动**，迁移任务正式开始，您可在数据迁移界面查看具体进度。

3.7. PolarDB间的数据同步

3.7.1. PolarDB MySQL集群间的单向同步

PolarDB是阿里巴巴自主研发的下一代关系型分布式云原生数据库，可完全兼容MySQL，具备简单易用、高性能、高可靠、高可用等优势。通过数据传输服务DTS（Data Transmission Service），您可以实现PolarDB MySQL集群间的单向数据同步。

前提条件

- 已购买源和目标PolarDB MySQL集群，详情请参见创建PolarDB MySQL集群。支持的数据库版本，请参见同步方案概览。
- 源PolarDB MySQL集群已开启Binlog，详情请参见如何开启Binlog。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据同步前评估源库和目标库的性能，同时建议您在业务低峰期执行数据同步（例如源库和目标库的CPU负载在30%以下）。
-
- 全量初始化过程中，并发INSERT会导致目标集群的表碎片，全量初始化完成后，目标集群的表空间比源库的表空间大。
- 如果数据同步的源库没有主键或唯一约束，且记录的全字段没有唯一性，可能会出现重复数据。
- 建议源和目标PolarDB MySQL集群的数据库版本保持一致，或者从低版本同步到高版本以保障兼容性。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据同步前评估源库和目标库的性能，同时建议您在业务低峰期执行数据同步（例如源库和目标库的CPU负载在30%以下）。
-
- 全量初始化过程中，并发INSERT会导致目标集群的表碎片，全量初始化完成后，目标集群的表空间比源库的表空间大。
- 如果数据同步的源库没有主键或唯一约束，且记录的全字段没有唯一性，可能会出现重复数据。

支持同步的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE、REPLACE
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

支持的同步架构

- 一对一单向同步
- 一对多单向同步
- 级联单向同步
- 多对一单向同步

关于各类同步架构的介绍及注意事项，请参见数据同步拓扑介绍。

功能限制

- 不兼容触发器

  当同步对象为整个库，且库中的触发器（TRIGGER）会更新库内某个表时，可能导致源和目标库的数据不一致。相关解决方案请参见源库存在触发器时如何配置同步作业。

- RENAME TABLE限制

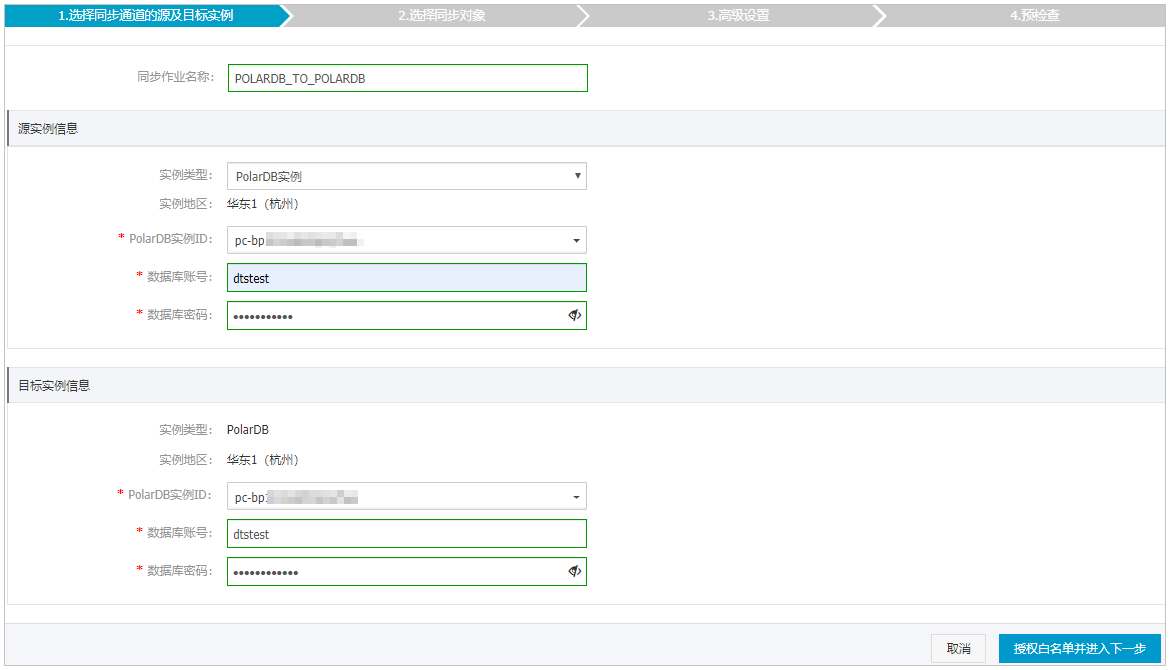
  RENAME TABLE操作可能导致同步数据不一致。例如同步对象只包含某个表，如果同步过程中源实例对该表执行了重命名操作，那么该表的数据将不会同步到目标库。为避免该问题，您可以在数据同步配置时将该表所属的整个数据库作为同步对象。

操作步骤

1. 购买数据同步作业，详情请参见购买流程。

   > **说明** 购买时，选择源实例和目标实例均为PolarDB，并选择同步拓扑为**单向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。
5. 定位至已购买的数据同步实例，单击**配置同步链路**。
6. 配置同步通道的源实例及目标实例信息。



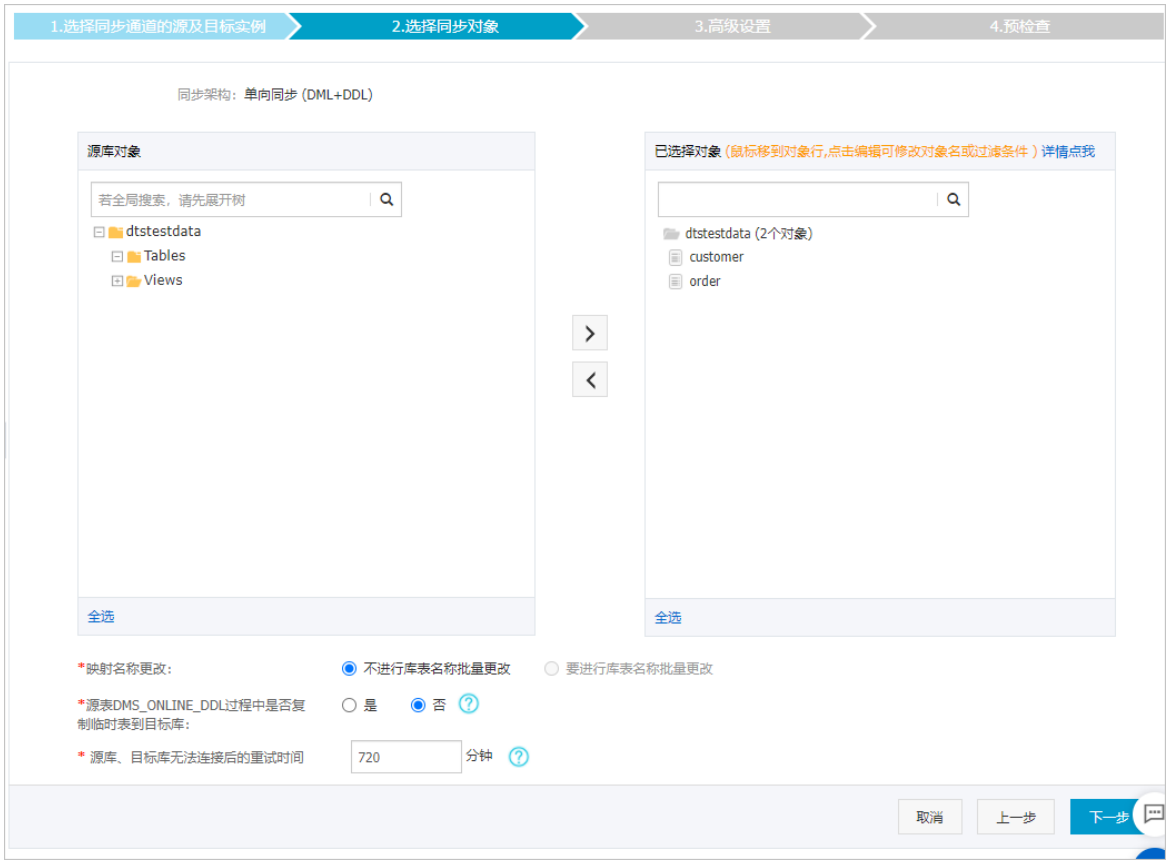
类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
	实例类型	固定为**PolarDB实例**，不可变更。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。

类别	配置	说明
源实例信息	PolarDB实例ID	选择源PolarDB集群ID。
	数据库账号	填入连接PolarDB集群的数据库账号。 **说明** 该账号需具备REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和所有同步对象的SELECT权限。
	数据库密码	填入数据库账号对应的密码。
目标实例信息	实例类型	固定为**PolarDB**，不可变更。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	PolarDB实例ID	选择目标PolarDB集群ID。
	数据库账号	填入连接PolarDB集群的数据库账号。 **说明** 用于数据同步的数据库账号需具备目标同步对象的ALL权限。
	数据库密码	填入数据库账号对应的密码。

7. 单击页面右下角的**授权白名单并进入下一步**。

   说明 此步骤会将DTS服务器的IP地址自动添加到源和目标PolarDB集群的白名单中，用于保障DTS服务器能够正常连接源和目标集群。

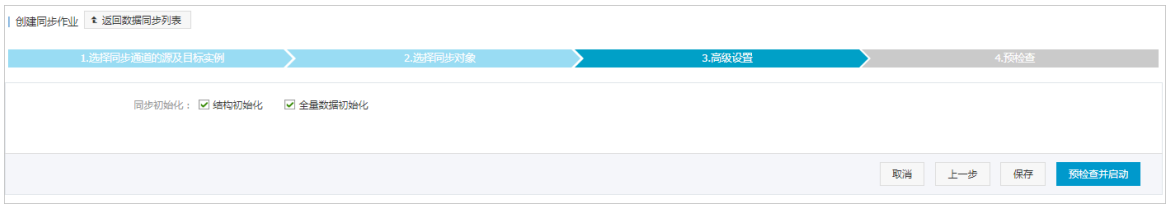
8. 配置目标已存在表的处理模式和同步对象。



配置项目	配置说明
目标已存在表的处理模式	○ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据同步作业不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以设置同步对象在目标实例中的名称来避免表名冲突。 ○ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，如果在目标库遇到与源库主键的值相同的记录，在初始化阶段会保留目标库中的该条记录；在增量同步阶段则会覆盖目标库的该条记录。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。

配置项目	配置说明
选择同步对象	在**源库对象**框中单击待同步的对象，然后单击 › 图标将其移动至**已选择对象**框。 同步对象的选择粒度为库、表。 **说明** ○ 如果选择整个库作为同步对象，那么该库中所有对象的结构变更操作都会同步至目标库。 ○ 默认情况下，同步对象的名称保持不变。如果您需要改变同步对象在目标集群中的名称，请使用对象名映射功能，详情请参见设置同步对象在目标实例中的名称。
映射名称更改	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ○ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ○ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源、目标库无法连接重试时间	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

9. 上述配置完成后，单击页面右下角的**下一步**。
10. 配置同步初始化的高级配置信息。



> **说明** 同步初始化类型细分为：结构初始化，全量数据初始化。选中**结构初始化**和**全量数据初始化**后，DTS会在增量数据同步之前，将源数据库中待同步对象的结构和存量数据，同步到目标数据库。

11. 上述配置完成后，单击页面右下角的**预检查并启动**。

> **说明**
> - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
> - 如果预检查失败，单击具体检查项后的，查看失败详情。
>   - 您可以根据提示修复后重新进行预检查。
>   - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

12. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，同步作业将正式开始。
13. 等待同步作业的链路初始化完成，直至处于**同步中**状态。

    您可以在**数据同步**页面，查看数据同步作业的状态。

3.7.2. PolarDB MySQL集群间的双向同步

数据传输服务DTS（Data Transmission Service）支持两个PolarDB MySQL集群间的双向数据同步，适用于异地多活、异地容灾等多种应用场景。本文介绍双向数据同步的配置步骤。

前提条件

- 已购买源和目标PolarDB MySQL集群（简称PolarDB集群），详情请参见创建PolarDB MySQL集群。
- 源和目标PolarDB集群已开启Binlog，详情请参见如何开启Binlog。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据同步前评估源库和目标库的性能，同时建议您在业务低峰期执行数据同步（例如源库和目标库的CPU负载在30%以下）。
- 全量初始化过程中，并发INSERT会导致目标集群的表碎片，全量初始化完成后，目标集群的表空间比源集群的表空间大。
- 如果数据同步的源集群没有主键或唯一约束，且记录的全字段没有唯一性，可能会出现重复数据。
- 如双向同步任务的源实例或目标实例位于海外地域，则仅支持同地域的双向同步，不支持跨地域的双向同步。例如，支持日本地域间的双向同步，不支持日本地域与法兰克福地域间的双向同步。

功能限制

- 目前DTS仅支持一对一双向同步，暂不支持多个PolarDB集群间的双向同步。
-
- 不兼容触发器

  当同步对象为整个库，且库中的触发器（TRIGGER）会更新库内某个表时，可能导致源和目标库的数据不一致。相关解决方案请参见源库存在触发器时如何配置同步作业。

- RENAME TABLE限制

  RENAME TABLE操作可能导致同步数据不一致。例如同步对象只包含某个表，如果同步过程中源实例对该表执行了重命名操作，那么该表的数据将不会同步到目标库。为避免该问题，您可以在数据同步配置时将该表所属的整个数据库作为同步对象。

- DDL语法同步方向限制

  为保障双向同步链路的稳定性，DDL更新只能在其中一个同步方向进行同步。即一旦某个同步方向配置了DDL同步，则在反方向上不支持DDL同步，只进行DML同步。

支持同步的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE、REPLACE
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

支持的冲突检测

为保障同步数据的一致性，您需要确保同一个主键、业务主键或唯一键的记录只在一个PolarDB集群上执行更新。如果发生了误操作，在两个PolarDB集群上均执行更新，那么将出现同步冲突。

DTS通过冲突检测和修复最大程度地维护双向同步实例的稳定性。目前DTS支持进行检测的冲突类型包括：

- INSERT导致的唯一性冲突

  同步INSERT语句时违背了唯一性约束，例如双向同步的两个节点同时或者在极为接近的时间新增了某个主键值相同的记录，那么同步到对端时，会因为已经存在相同主键值的记录，导致INSERT操作的同步失败。

- UPDATE更新的记录不完全匹配
  - UPDATE要更新的记录在同步目标集群中不存在时，DTS会自动转化为INSERT，此时可能会出现唯一键的唯一性冲突。
  - UPDATE要更新的记录出现主键或唯一键冲突。
- DELETE对应的记录不存在

  DELETE要删除的记录在同步的目标集群中不存在。出现这种冲突时，不论配置何种冲突修复策略，DTS都会自动忽略DELETE操作。

> 注意
>
> - 由于数据同步两端的系统时间可能存在差异、同步存在延时等多种因素，DTS无法完全保证冲突检测机制能够完全防止数据的冲突。在使用双向同步时，您需要在业务层面配合进行相应的改造，保证同一个主键、业务主键或唯一键的记录只在双向同步的某个节点进行更新。
> - 对于上述数据同步的冲突，DTS提供了修复策略，您可以在配置双向同步时选择。

操作步骤

1. 购买数据同步作业，详情请参见购买流程。

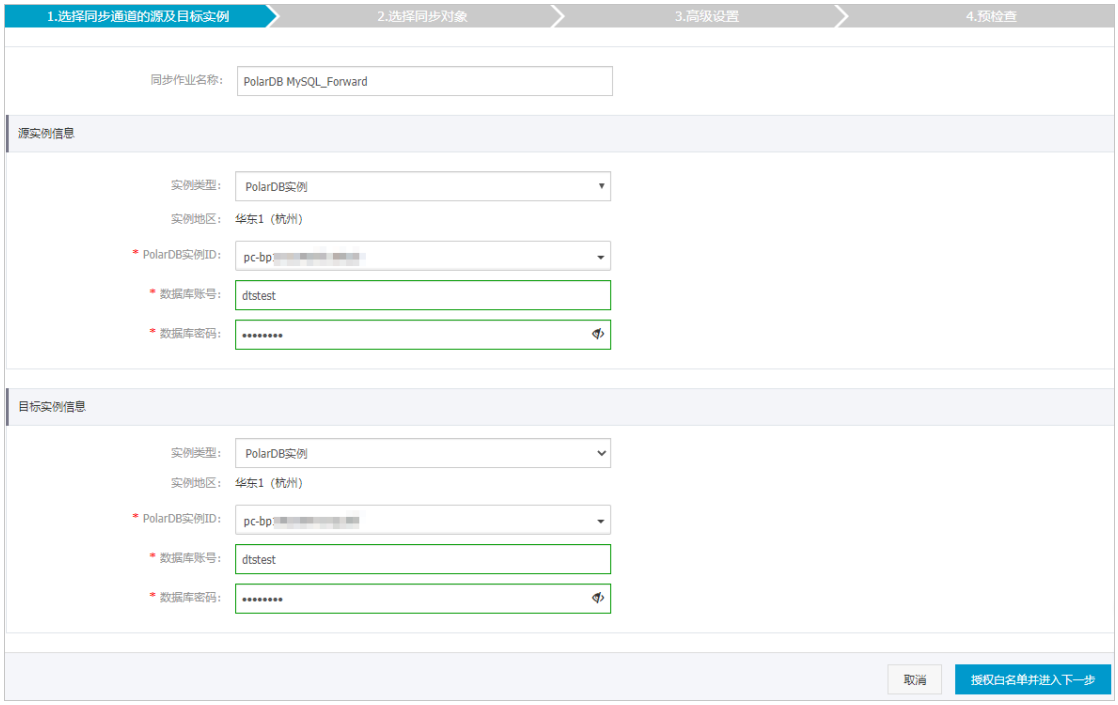
   > 注意　购买时，选择源实例和目标实例均为PolarDB，并选择同步拓扑为**双向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。
5. 配置正向同步作业。
   i. 定位至已购买的数据同步实例，单击该实例下第一个同步作业**操作**列的**配置同步链路**。

   > 注意　一个双向数据同步实例会包含两个同步作业，需要分别进行配置。在配置反向同步作业时，定位至第二个同步作业，单击**操作**列的**配置同步链路**。

   ii. 配置同步作业的源集群及目标集群信息。



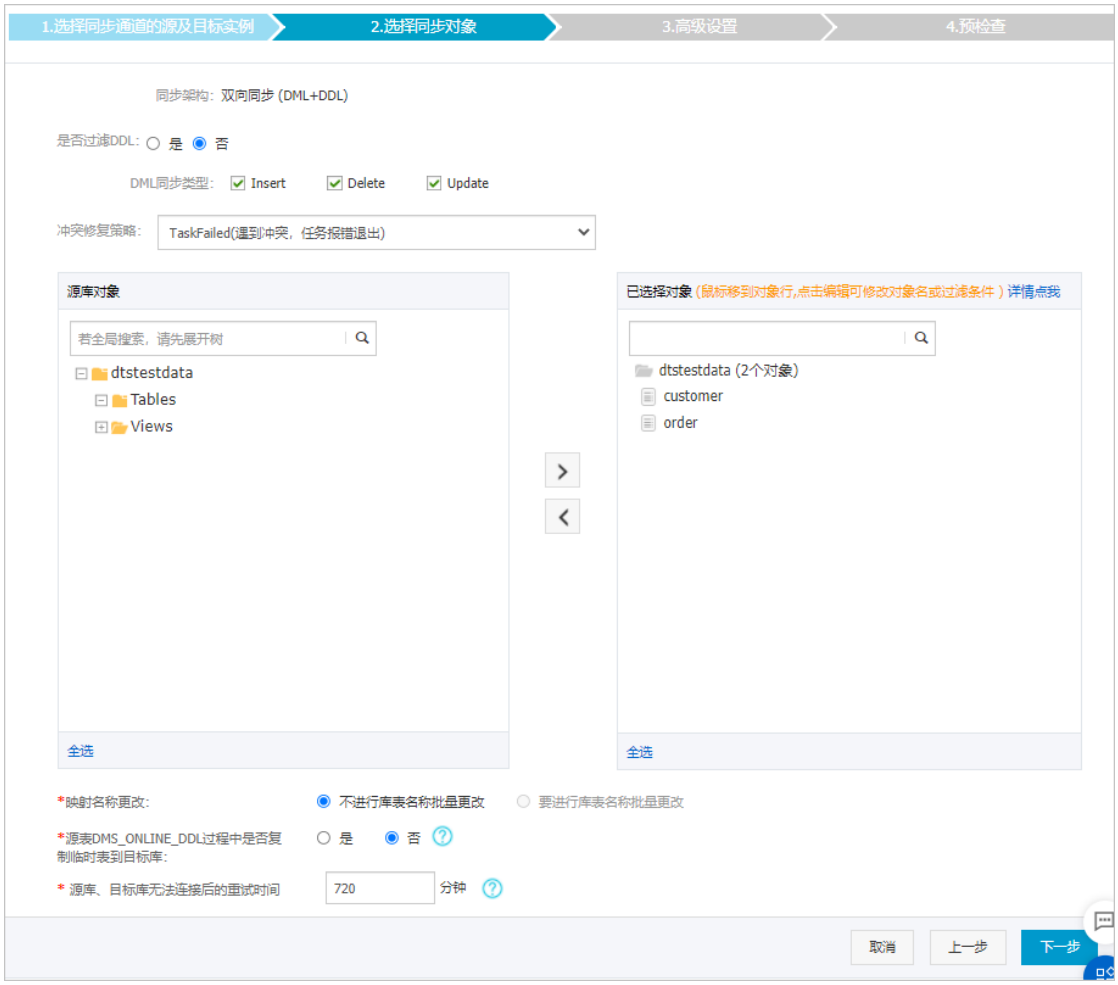
类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	固定为**PolarDB实例**，不可变更。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	PolarDB实例ID	选择源PolarDB集群ID。 注意　稍后配置反向同步作业时，此处需选择为正向同步作业里的目标PolarDB集群ID。
	数据库账号	填入连接PolarDB集群的数据库账号，该账号需具备待同步对象读写权限。
	数据库密码	填入数据库账号的密码。
目标实例信息	实例类型	固定为**PolarDB实例**，不可变更。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	PolarDB实例ID	选择目标PolarDB集群ID。 注意　稍后配置反向同步作业时，此处需选择为正向同步作业里的源PolarDB集群ID。

类别	配置	说明
	数据库账号	填入连接PolarDB集群的数据库账号，该账号需具备目标同步对象读写权限。
	数据库密码	填入数据库账号的密码。

iii. 单击页面右下角的**授权白名单并进入下一步**。

> **说明**
>
> - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段。
> - DTS任务完成或释放后，建议您手动删除添加的DTS服务器IP地址段。

iv. 配置同步策略及对象信息。



类别	配置	说明

类别	配置	说明
同步策略配置	是否过滤DDL	■ 选择为**是**：不同步DDL操作。 ■ 选择为**否**：同步DDL操作。 **注意** DDL语法同步方向限制 。为保障双向同步链路的稳定性，只支持正向同步DDL，不支持反向同步DDL。
	DML同步类型	选择需要同步的DML类型，默认为**Insert**、**Update**、**Delete**，您可以根据业务需求调整。
	冲突修复策略	选择同步冲突的修复策略，默认为**TaskFailed**，您可以根据业务情况选择合适的冲突修复策略。 ■ **TaskFailed**（遇到冲突，任务报错退出） 默认的冲突修复策略。当数据同步遇到上述冲突类型时，同步任务直接报错并退出，同步任务进入失败状态，需要用户介入修复任务。 ■ **Ignore**（遇到冲突，直接使用目标实例中的冲突记录） 当数据同步遇到上述的冲突类型时，直接跳过当前同步语句，继续往下执行，选择使用目标库中的冲突记录。 ■ **Overwrite**（遇到冲突，直接覆盖目标实例中的冲突记录） 当数据同步遇到上述的冲突类型时，直接覆盖目标库中的冲突记录。
	目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标库中是否有同名的表。如果目标库中没有同名的表，则通过该检查项目；如果目标库中有同名的表，则在预检查阶段提示错误，数据同步作业不会被启动。 **注意** 如果目标库中同名的表不方便删除或重命名，您可以设置同步对象在目标实例中的名称来避免表名冲突。 ■ **忽略报错并继续执行**：跳过目标库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，如果在目标库遇到与源库主键的值相同的记录，在初始化阶段会保留目标库中的该条记录；在增量同步阶段则会覆盖目标库的该条记录。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。

类别	配置	说明
选择同步对象	无	在**源库对象**框中单击待同步的对象（选择粒度为库或表），然后单击 > 将其移动到**已选择对象**框。 **注意** ■ 如果选择整个库作为同步对象，该库中所有对象的结构变更操作都会同步至目标库。 ■ 默认情况下，同步对象的名称保持不变。如果您需要改变同步对象在目标库中的名称，需要使用对象名映射功能，详情请参见设置同步对象在目标实例中的名称。
映射名称更改	无	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	无	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ■ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ■ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源、目标库无法连接重试时间	无	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

v. 上述配置完成后，单击页面右下角的**下一步**。

vi. 配置同步初始化的高级配置信息。

> **注意**
>
> - 此步骤会将源集群中待同步对象的结构及数据同步至目标集群，作为后续增量同步数据的基线数据。同步初始化类型细分为：**结构初始化**、**全量数据初始化**。默认情况下，需要同时选择**结构初始化**和**全量数据初始化**。
> - 在配置反向同步作业时，如果同步对象已经初始化至目标集群，则直接同步增量数据。

vii. 上述配置完成后，单击页面右下角的**预检查并启动**。

> **说明**
>
> - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
> - 如果预检查失败，单击具体检查项后的，查看失败详情。
>
> - 您可以根据提示修复后重新进行预检查。
> - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

viii. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，正向同步作业将正式开始。

6. 等待正向同步作业完成初始化，直至处于**同步中**状态。

   您可以在**数据同步**页面，查看数据同步作业的状态。

7. 配置反向同步作业。

   i. 定位至第二个同步作业，单击**配置同步链路**。

   ii. 重复步骤5中的ii到viii配置步骤，完成反向同步作业的配置。

执行结果

等待一段时间后，两个同步作业的链路状态均会处于**同步中**。

3.8. PolarDB与其它数据库的数据同步

3.8.1. RDS MySQL同步至PolarDB MySQL集群

通过数据传输服务DTS（Data Transmission Service），可以实现MySQL同步至PolarDB MySQL引擎集群。

支持的源数据库

MySQL同步至PolarDB MySQL引擎集群，支持源数据库MySQL为以下类型。本文以RDS MySQL实例为例介绍配置流程，其他类型的源数据库配置流程与本案例类似。

- RDS MySQL实例。
- ECS上的自建数据库。
- 通过专线、VPN网关或智能网关接入的自建数据库。
- 通过数据库网关接入的自建数据库。
- 通过云企业网CEN接入的自建数据库。

前提条件

- 已创建源RDS MySQL实例，创建方式，请参见创建RDS MySQL实例。
- 已创建目标PolarDB MySQL引擎集群，详情请参见购买按量付费集群和购买包年包月集群。
- PolarDB MySQL引擎集群的存储空间须大于RDS MySQL实例占用的存储空间。

注意事项

类型	说明
源库限制	•
其他限制	•
特殊情况	

支持的同步架构

- 一对一单向同步
- 一对多单向同步
- 级联单向同步

- 多对一单向同步

关于各类同步架构的介绍及注意事项，请参见数据同步拓扑介绍。

支持同步的SQL

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

操作步骤

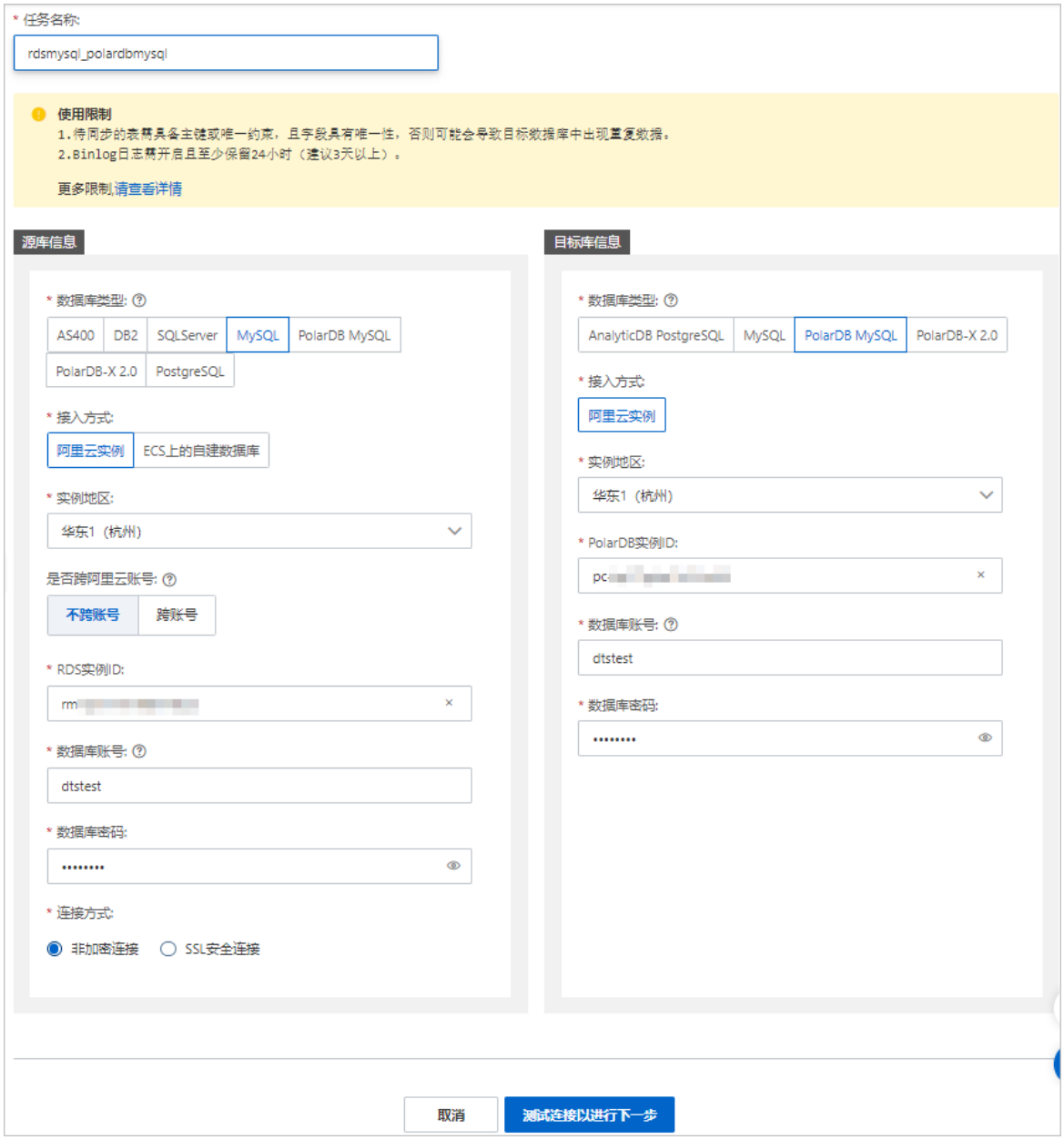
1. 登录新版DTS同步任务的列表页面。

   说明　您也可以登录DMS数据管理服务。在顶部菜单栏中，选择**集成与开发（DTS）** > **数据集成** > **数据同步**。

2. 在页面左上角，选择同步实例所属地域。

3. 单击**创建任务**，配置源库及目标库信息。

   警告　选择源和目标实例后，建议您仔细阅读页面上方显示的**使用限制**，以成功创建并执行同步任务。



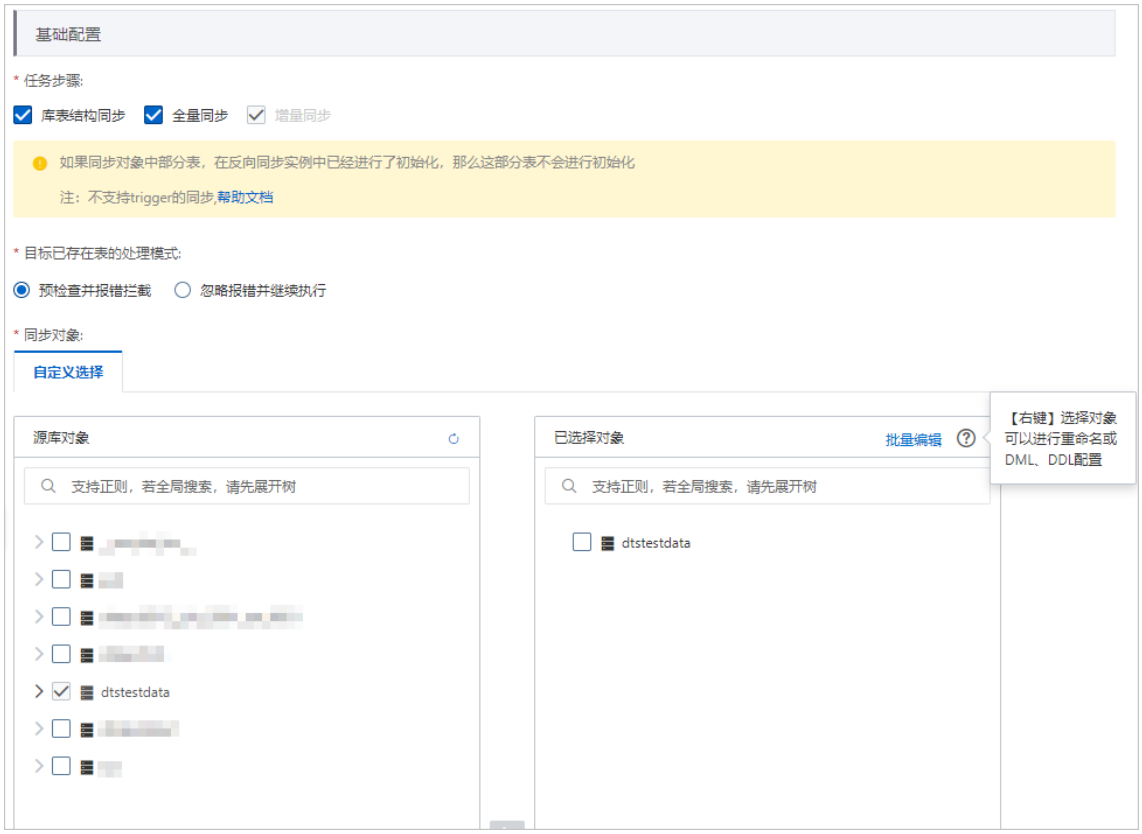
类别	配置	说明
无	**任务名称**	DTS会自动生成一个任务名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
	数据库类型	选择**MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择源RDS MySQL实例所属地域。
	是否跨阿里云账号	本场景为同一阿里云账号间迁移，选择**不跨账号**。

类别	配置	说明
源库信息	RDS实例ID	选择源RDS MySQL实例ID。 说明 源和目标RDS MySQL实例可以不同或相同，即您可以使用DTS实现两个RDS MySQL实例间的数据迁移或同一RDS MySQL实例内的数据迁移。
	数据库账号	填入源RDS MySQL实例的数据库账号，需具备REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和SELECT权限。
	数据库密码	填入该数据库账号对应的密码。
	连接方式	根据需求选择**非加密连接**或**SSL安全连接**。如果设置为**SSL安全连接**，您需要提前开启RDS MySQL实例的SSL加密功能，详情请参见设置SSL加密。
目标库信息	数据库类型	选择**PolarDB MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择目标PolarDB MySQL引擎集群所属地域。
	PolarDB MySQL实例ID	选择目标PolarDB MySQL引擎集群ID。
	数据库账号	填入目标PolarDB MySQL引擎集群的数据库账号，需具备读写权限。
	数据库密码	填入该数据库账号对应的密码。

4. 配置完成后，单击页面右下角的**测试连接以进行下一步**。

   说明
   - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段。
   - DTS任务完成或释放后，建议您手动删除添加的DTS服务器IP地址段。

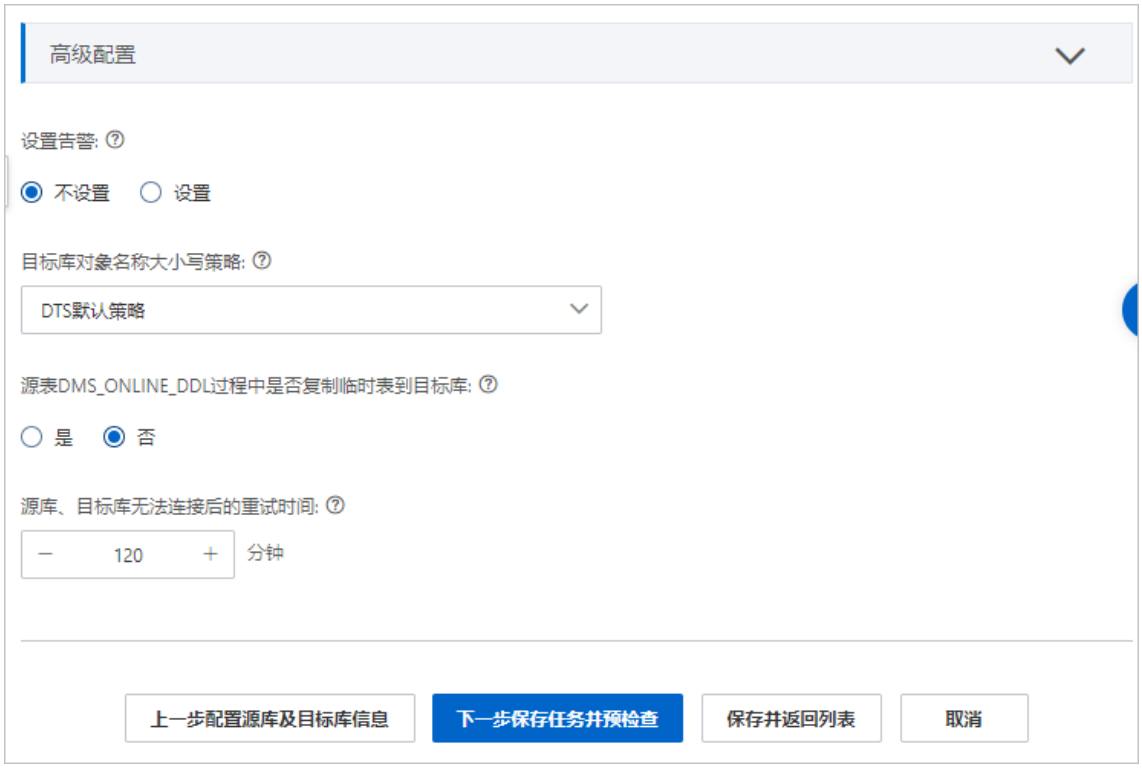
5. 配置任务对象及高级配置。
   - **基础配置**



配置	说明
任务步骤	固定选中**增量同步**。默认情况下，您还需要同时选中**库表结构同步**和**全量同步**。预检查完成后，DTS会将源实例中待同步对象的全量数据在目标集群中初始化，作为后续增量同步数据的基线数据。

配置	说明
目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据同步任务不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以更改该表在目标库中的名称，请参见库表列名映射。 ■ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，如在目标库遇到与源库主键的值相同的记录： ■ 全量期间，DTS会保留目标集群中的该条记录，即源库中的该条记录不会同步至目标数据库中。 ■ 增量期间，DTS不会保留目标集群中的该条记录，即源库中的该条记录会覆盖至目标数据库中。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。
同步对象	在**源库对象**框中单击待同步对象，然后单击 › 将其移动至**已选择对象**框。 **说明** 同步对象选择的粒度为库、表、列。若选择的同步对象为表或列，其他对象（如视图、触发器、存储过程）不会被同步至目标库。
映射名称更改	■ 如需更改单个同步对象在目标实例中的名称，请单击**已选择对象**中的同步对象，设置方式，请参见库表列名单个映射。 ■ 如需批量更改同步对象在目标实例中的名称，请单击**已选择对象**方框右上方的**批量编辑**，设置方式，请参见库表列名批量映射。
过滤待同步数据	支持设置WHERE条件过滤数据，请参见通过SQL条件过滤任务数据。
同步的SQL操作	请右击**已选择对象**中的同步对象，在弹跳框中选择所需同步的DML和DDL操作，支持的操作，请参见支持同步的SQL。

- **高级配置**



配置	说明
设置告警	是否设置告警，当同步失败或延迟超过阈值后，将通知告警联系人。 ■ **不设置**：不设置告警。 ■ **设置**：设置告警，您还需要设置告警阈值和告警联系人。
目标库对象名称大小写策略	您可以配置目标实例中迁移对象的库名、表名和列名的英文大小写策略。默认情况下选择**DTS默认策略**，您也可以选择与源库、目标库默认策略保持一致。更多信息，请参见目标库对象名称大小写策略。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ■ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ■ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。

配置	说明
源库、目标库无法连接后的重试时间	默认重试120分钟，您也可以在取值范围（10~1440分钟）内自定义重试时间，建议设置30分钟以上。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** ■ 针对同源或者同目标的多个DTS实例，如DTS实例A和DTS实例B，设置网络重试时间时A设置30分钟，B设置60分钟，则重试时间以低的30分钟为准。 ■ 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

6. 上述配置完成后，单击页面下方的**下一步保存任务并预检查**。

   说明
   - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
   - 如果预检查失败，单击具体检查项后的，查看失败详情。
     - 您可以根据提示修复后重新进行预检查。
     - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

7. **预检查通过率**显示为**100%**时，单击**下一步购买**。
8. 在**购买**页面，选择数据同步实例的计费方式、链路规格，详细说明请参见下表。

类别	参数	说明
信息配置	计费方式	○ 预付费（包年包月）：在新建实例时支付费用。适合长期需求，价格比按量付费更实惠，且购买时长越长，折扣越多。 ○ 后付费（按量付费）：按小时扣费。适合短期需求，用完可立即释放实例，节省费用。
	链路规格	DTS为您提供了不同性能的同步规格，同步链路规格的不同会影响同步速率，您可以根据业务场景进行选择，详情请参见数据同步链路规格说明。
	订购时长	在预付费模式下，选择包年包月实例的时长和数量，包月可选择1~9个月，包年可选择1~3年。 **说明** 该选项仅在付费类型为**预付费**时出现。

9. 配置完成后，阅读并勾选**《数据传输（按量付费）服务条款》**。
10. 单击**购买并启动**，同步任务正式开始，您可在数据同步界面查看具体任务进度。

3.8.2. PolarDB MySQL同步至RDS MySQL

通过数据传输服务DTS（Data Transmission Service），可以实现PolarDB MySQL引擎集群同步至MySQL（包括自建MySQL、RDS MySQL实例）。

支持的目标数据库

PolarDB MySQL引擎集群同步至MySQL，支持目标数据库为以下类型。本文以RDS MySQL实例为例介绍配置流程，其他类型的目标数据库的配置流程与本案例类似。

- RDS MySQL实例。
- ECS上的自建数据库。
- 通过专线、VPN网关或智能网关接入的自建数据库。
- 通过数据库网关接入的自建数据库。
- 通过云企业网CEN接入的自建数据库。

前提条件

- 已创建源PolarDB MySQL引擎集群，详情请参见购买按量付费集群和购买包年包月集群。
- 已创建目标RDS MySQL实例，创建方式，请参见创建RDS MySQL实例。
- 目标RDS MySQL实例的存储空间须大于源PolarDB MySQL引擎集群占用的存储空间。

注意事项

类型	说明
源库限制	•
其他限制	• • • • •

优惠活动

DTS优惠活动，最低0折

支持的同步架构

- 一对一单向同步
- 一对多单向同步
- 级联单向同步
- 多对一单向同步
- 一对一双向同步

关于各类同步架构的介绍及注意事项，请参见数据同步拓扑介绍。

支持同步的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

操作步骤

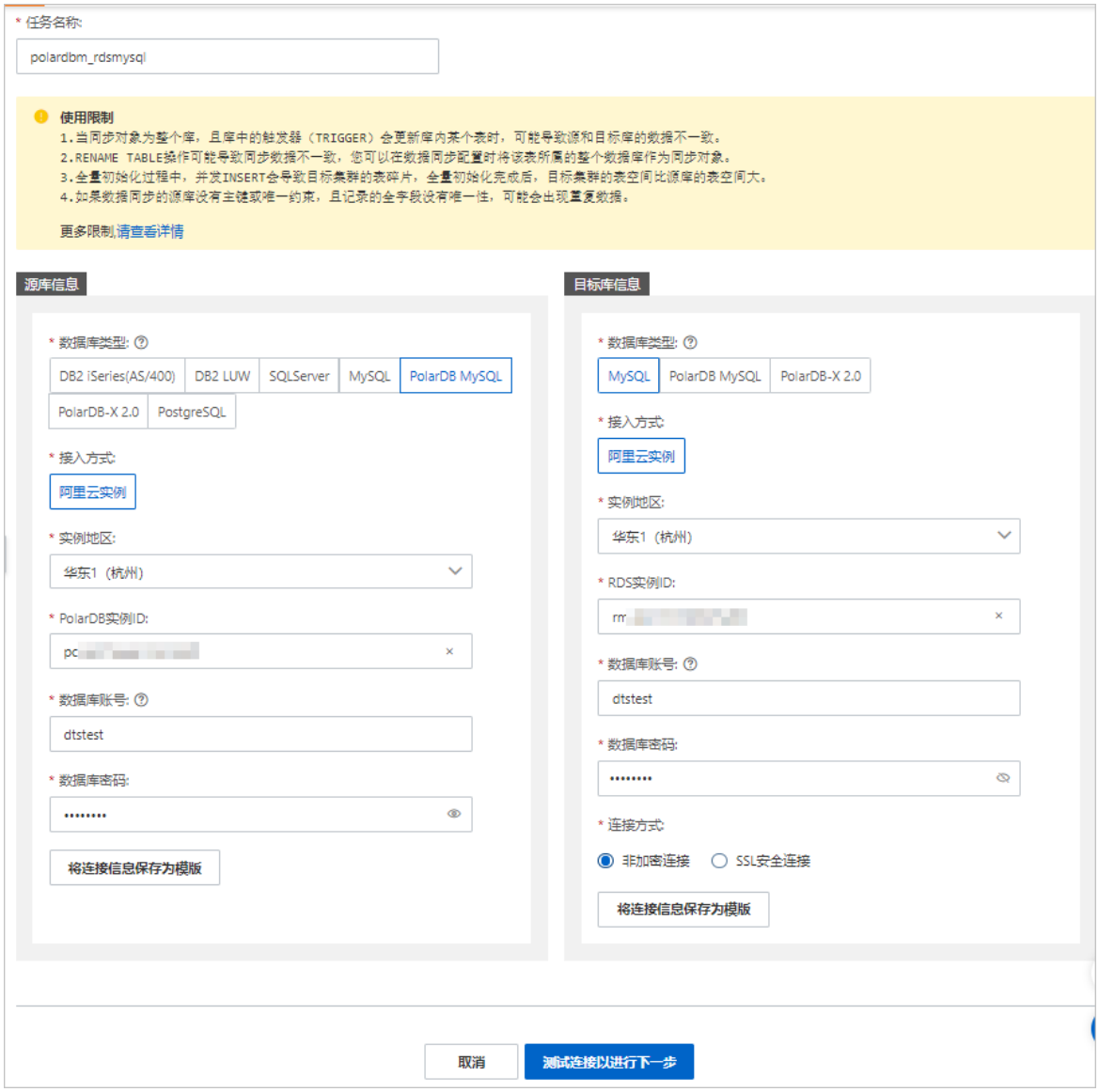
1. 登录新版DTS同步任务的列表页面。

   > 说明　您也可以登录DMS数据管理服务。在顶部菜单栏中，选择**集成与开发（DTS）** > **数据集成** > **数据同步**。

2. 在页面左上角，选择同步实例所属地域。

3. 单击**创建任务**，配置源库及目标库信息。

   > 警告　选择源和目标实例后，建议您仔细阅读页面上方显示的**使用限制**，以成功创建并执行同步任务。



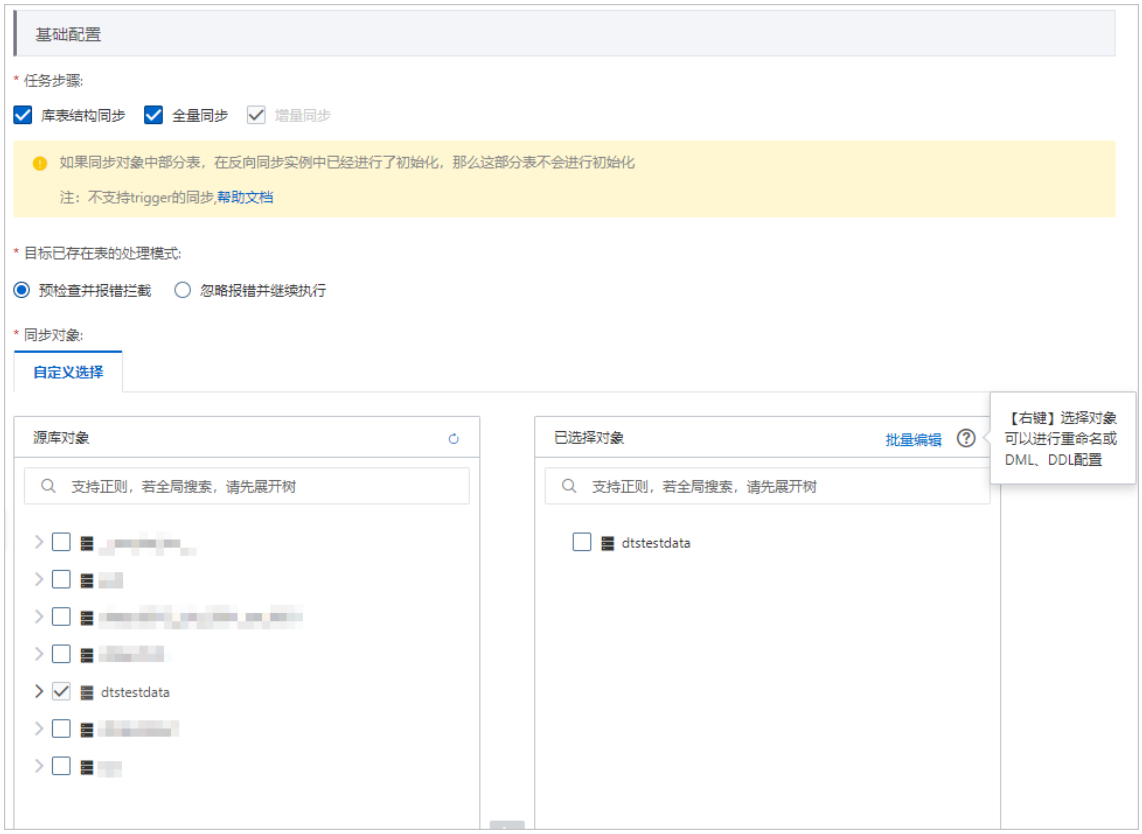
类别	配置	说明
无	**任务名称**	DTS会自动生成一个任务名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源库信息	**数据库类型**	选择**PolarDB MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择源PolarDB MySQL引擎集群所属地域。
	PolarDB MySQL实例ID	选择源PolarDB MySQL引擎集群ID。
	数据库账号	填入源PolarDB MySQL引擎集群的数据库账号，需具备待同步对象的读权限。

类别	配置	说明
	数据库密码	填入该数据库账号对应的密码。
目标库信息	**数据库类型**	选择**MySQL**。
	接入方式	选择**阿里云实例**。
	实例地区	选择目标RDS MySQL实例所属地域。
	RDS实例ID	选择目标RDS MySQL实例ID。
	数据库账号	填入目标RDS MySQL实例的数据库账号，需具备读写权限。
	数据库密码	填入该数据库账号对应的密码。
	连接方式	根据需求选择**非加密连接**或**SSL安全连接**。如果设置为**SSL安全连接**，您需要提前开启RDS MySQL实例的SSL加密功能，详情请参见设置SSL加密。

4. 配置完成后，单击页面右下角的**测试连接以进行下一步**。

> 说明
> - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段。
> - DTS任务完成或释放后，建议您手动删除添加的DTS服务器IP地址段。

5. 配置任务对象及高级配置。
   - **基础配置**



配置	说明
任务步骤	固定选中**增量同步**。默认情况下，您还需要同时选中**库表结构同步**和**全量同步**。预检查完成后，DTS会将源实例中待同步对象的全量数据在目标集群中初始化，作为后续增量同步数据的基线数据。

配置	说明
目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据同步任务不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以更改该表在目标库中的名称，请参见库表列名映射。 ■ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，如在目标库遇到与源库主键的值相同的记录： ■ 全量期间，DTS会保留目标集群中的该条记录，即源库中的该条记录不会同步至目标数据库中。 ■ 增量期间，DTS不会保留目标集群中的该条记录，即源库中的该条记录会覆盖至目标数据库中。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。
同步对象	在**源库对象**框中单击待同步对象，然后单击 › 将其移动至**已选择对象**框。 **说明** 同步对象选择的粒度为库、表、列。若选择的同步对象为表或列，其他对象（如视图、触发器、存储过程）不会被同步至目标库。
映射名称更改	■ 如需更改单个同步对象在目标实例中的名称，请单击**已选择对象**中的同步对象，设置方式，请参见库表列名单个映射。 ■ 如需批量更改同步对象在目标实例中的名称，请单击**已选择对象**方框右上方的**批量编辑**，设置方式，请参见库表列名批量映射。
过滤待同步数据	支持设置WHERE条件过滤数据，请参见通过SQL条件过滤任务数据。
同步的SQL操作	请右击**已选择对象**中的同步对象，在弹跳框中选择所需同步的DML和DDL操作，支持的操作，请参见支持同步的SQL操作。

- **高级配置**

配置	说明
设置告警	是否设置告警，当同步失败或延迟超过阈值后，将通知告警联系人。 ■ **不设置**：不设置告警。 ■ **设置**：设置告警，您还需要设置告警阈值和告警联系人。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ■ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ■ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源库、目标库无法连接后的重试时间	默认重试120分钟，您也可以在取值范围（10~1440分钟）内自定义重试时间，建议设置30分钟以上。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** ■ 针对同源或者同目标的多个DTS实例，如DTS实例A和DTS实例B，设置网络重试时间时A设置30分钟，B设置60分钟，则重试时间以低的30分钟为准。 ■ 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

6. 上述配置完成后，单击页面下方的**下一步保存任务并预检查**。

   > **说明**
   > - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
   > - 如果预检查失败，单击具体检查项后的，查看失败详情。
   >   - 您可以根据提示修复后重新进行预检查。
   >   - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

7. **预检查通过率**显示为**100%**时，单击**下一步购买**。
8. 在**购买**页面，选择数据同步实例的计费方式、链路规格，详细说明请参见下表。

类别	参数	说明
信息配置	计费方式	○ 预付费（包年包月）：在新建实例时支付费用。适合长期需求，价格比按量付费更实惠，且购买时长越长，折扣越多。 ○ 后付费（按量付费）：按小时扣费。适合短期需求，用完可立即释放实例，节省费用。
	链路规格	DTS为您提供了不同性能的同步规格，同步链路规格的不同会影响同步速率，您可以根据业务场景进行选择，详情请参见数据同步链路规格说明。
	订购时长	在预付费模式下，选择包年包月实例的时长和数量，包月可选择1~9个月，包年可选择1~3年。 **说明** 该选项仅在付费类型为**预付费**时出现。

9. 配置完成后，阅读并勾选**《数据传输（按量付费）服务条款》**。
10. 单击**购买并启动**，同步任务正式开始，您可在数据同步界面查看具体任务进度。

3.8.3. 从PolarDB MySQL同步至云原生数据仓库AnalyticDB MySQL

云原生数据仓库AnalyticDB MySQL是阿里巴巴自主研发的海量数据实时高并发在线分析（Realtime OLAP）云计算服务，可以对千亿级数据进行毫秒级的即时多维分析透视和业务探索。通过数据传输服务DTS（Data Transmission Service），您可以将PolarDB MySQL同步到云原生数据仓库AnalyticDB MySQL，帮助您快速构建企业内部BI、交互查询、实时报表等系统。

前提条件

- 已创建目标云原生数据仓库AnalyticDB MySQL集群，详情请参见创建云原生数据仓库AnalyticDB MySQL（2.0）或创建云原生数据仓库AnalyticDB MySQL（3.0）。
- 确保目标云原生数据仓库AnalyticDB MySQL具备充足的存储空间。

- PolarDB MySQL已开启Binlog，详情请参见如何开启Binlog。
- 如果同步的目标为云原生数据仓库AnalyticDB MySQL（2.0），源PolarDB MySQL中待同步的对象不能包含云原生数据仓库AnalyticDB MySQL（2.0）保留的库名和列名，否则将造成数据同步失败或DDL操作同步失败。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据同步前评估源库和目标库的性能，同时建议您在业务低峰期执行数据同步（例如源库和目标库的CPU负载在30%以下）。
- 请勿在数据同步时，对源库的同步对象使用gh-ost或pt-online-schema-change等类似工具执行在线DDL变更，否则会导致同步失败。

  > **说明** 如果同步的目标为云原生数据仓库AnalyticDB MySQL（3.0），您可以使用DMS提供的相关功能来执行在线DDL变更，详情请参见DDL无锁变更。

- 由于云原生数据仓库AnalyticDB MySQL（3.0）本身的使用限制，当云原生数据仓库AnalyticDB MySQL（3.0）集群中的节点磁盘空间使用量超过80%，该集群将被锁定。请提前根据待同步的对象预估所需空间，确保目标集群具备充足的存储空间。
- 暂不支持同步前缀索引，如果源库存在前缀索引可能导致数据同步失败。

术语/概念对应关系

PolarDB MySQL	云原生数据仓库AnalyticDB MySQL
数据库	• 云原生数据仓库AnalyticDB MySQL（2.0）：表组 • 云原生数据仓库AnalyticDB MySQL（3.0）：数据库
表	• 云原生数据仓库AnalyticDB MySQL（2.0）：表 • 云原生数据仓库AnalyticDB MySQL（3.0）：表

> **说明** 关于云原生数据仓库AnalyticDB MySQL中表组和表的相关介绍，请参见常见术语。

支持同步的SQL操作

目标数据库版本	支持的SQL操作
云原生数据仓库AnalyticDB MySQL2.0	• DDL操作：ADD COLUMN、DROP COLUMN、MODIFY COLUMN • DML操作：INSERT、UPDATE、DELETE

目标数据库版本	支持的SQL操作
云原生数据仓库AnalyticDB MySQL3.0	• DDL操作：CREATE TABLE、DROP TABLE、RENAME TABLE、TRUNCATE TABLE、ADD COLUMN、DROP COLUMN、MODIFY COLUMN • DML操作：INSERT、UPDATE、DELETE 说明 如果在数据同步的过程中变更了源表的字段类型，同步作业将报错并中断。您可以提交工单处理或手动修复，详情请参见修复因变更字段类型导致的同步失败。

数据库账号的权限要求

数据库	所需权限
PolarDB MySQL	待同步对象的读权限。
云原生数据仓库AnalyticDB MySQL（2.0）	无需填写数据库账号信息，DTS会自动创建账号并授权。
云原生数据仓库AnalyticDB MySQL（3.0）	读写权限。

数据库账号的创建和授权方法，请参见创建PolarDB MySQL数据库账号或创建云原生数据仓库AnalyticDB MySQL数据库账号。

数据类型映射关系

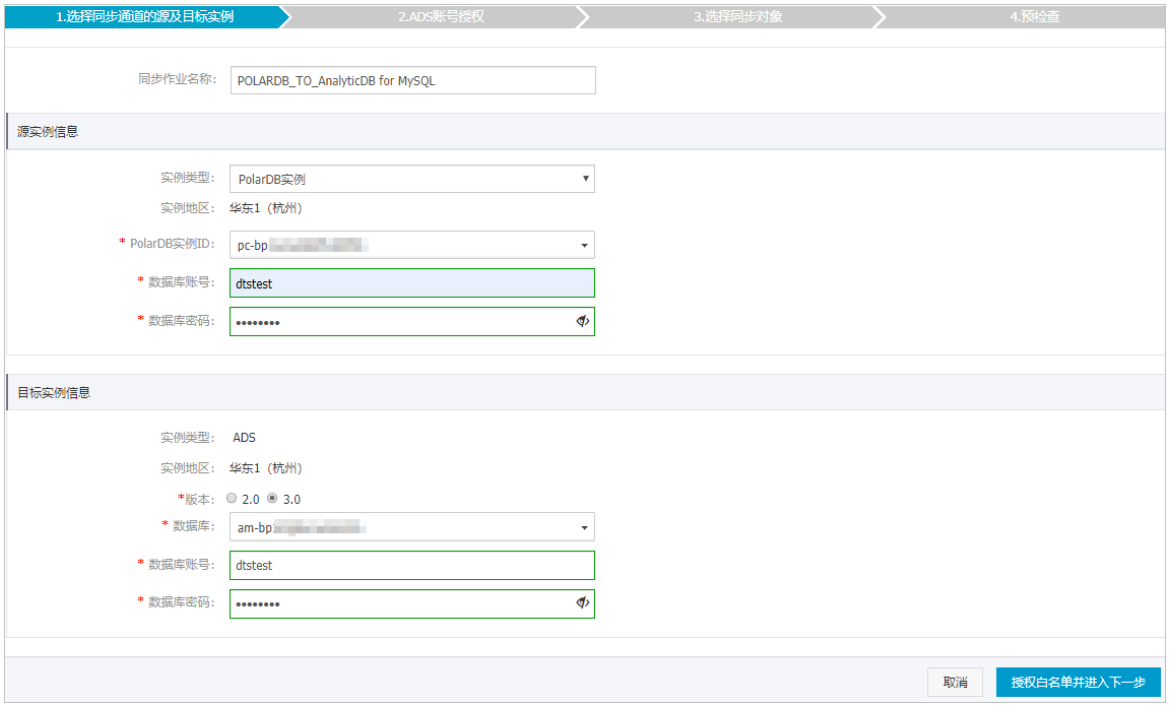
详情请参见结构初始化涉及的数据类型映射关系。

操作步骤

1. 购买数据同步实例。

   说明 购买时，选择源实例为PolarDB、目标实例为AnalyticDB MySQL，并选择同步拓扑为**单向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。
5. 定位至已购买的数据同步实例，单击**配置同步链路**。
6. 配置同步通道的源实例及目标实例信息。



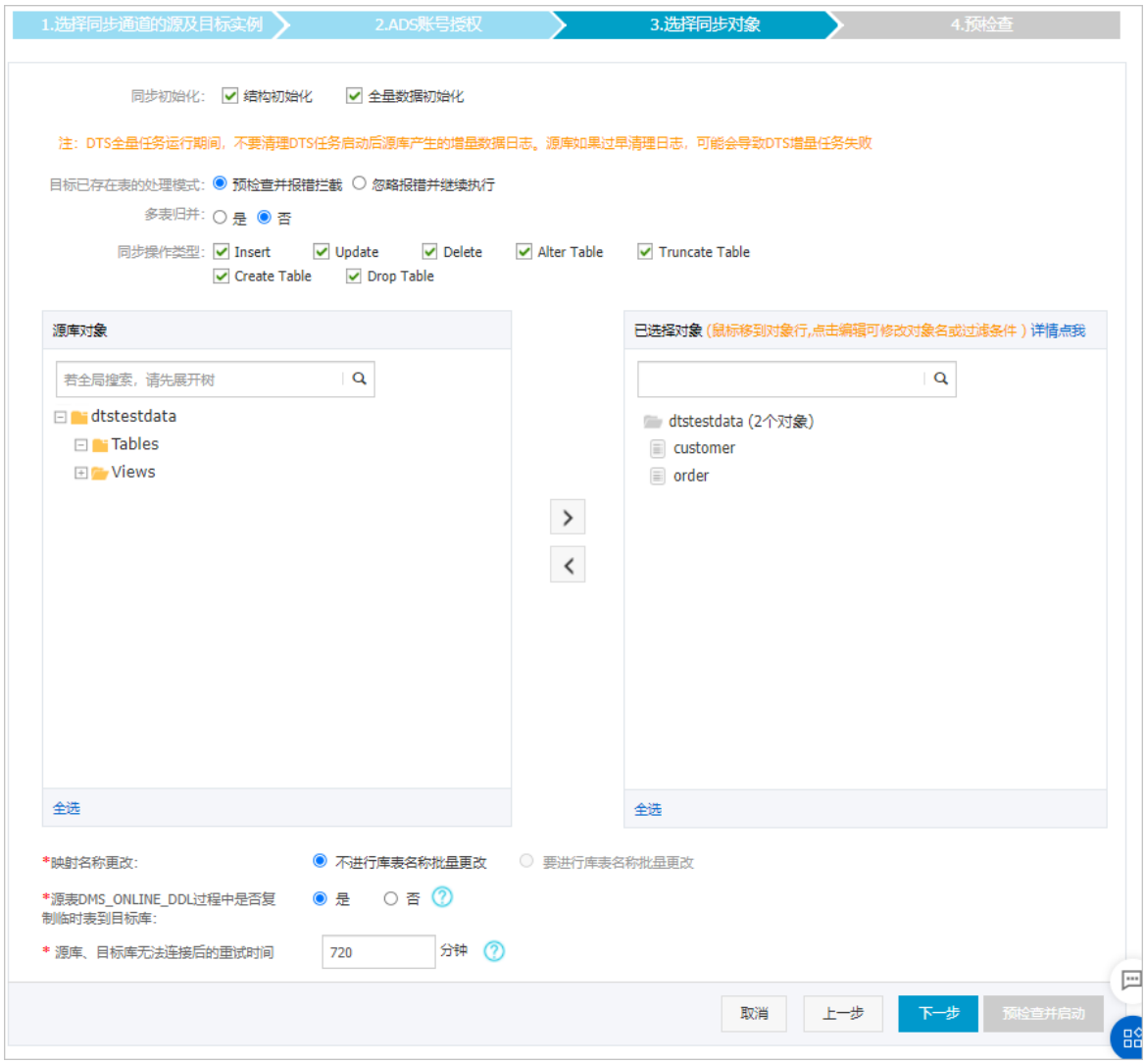
类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	固定为**PolarDB实例**，不可变更。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	PolarDB实例ID	选择源PolarDB集群ID。
	数据库账号	填入PolarDB集群的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入数据库账号对应的密码。
目标实例信息	实例类型	固定为**ADS**，不可变更。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	版本	根据目标云原生数据仓库AnalyticDB MySQL集群的版本，选择**2.0**或**3.0**。 **说明** ○ 选择为**2.0**后，DTS将自动创建数据库账号并进行授权，您无需配置**数据库账号**和**数据库密码**。 ○ 选择为**3.0**后，您还需要配置**数据库账号**和**数据库密码**。
	数据库	选择目标云原生数据仓库AnalyticDB MySQL的集群ID。

类别	配置	说明
	数据库账号	填入云原生数据仓库AnalyticDB MySQL的数据库账号，权限要求请参见数据库账号的权限要求。
	数据库密码	填入数据库账号对应的密码。

7. 单击页面右下角的**授权白名单并进入下一步**。

> **说明** 此步骤会将DTS服务器的IP地址自动添加到PolarDB MySQL和云原生数据仓库AnalyticDB MySQL的白名单中，用于保障DTS服务器能够正常连接源和目标集群。

8. 配置同步策略及对象信息。



配置	说明
同步初始化	默认情况下，您需要同时选中**结构初始化**和**全量数据初始化**。预检查完成后，DTS会将源实例中待同步对象的结构及数据在目标集群中初始化，作为后续增量同步数据的基线数据。

配置	说明
目标已存在表的处理模式	○ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据同步作业不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以更改该表在目标库中的名称，详情请参见设置同步对象在目标实例中的名称。 ○ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，在目标库遇到与源库主键的值相同的记录，则会保留目标集群中的该条记录，即源库中的该条记录不会同步至目标数据库中。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。
多表归并	○ 选择为**是**：DTS将在每个表中增加 `__dts_data_source` 列来存储数据来源，且不再支持DDL同步。 ○ 选择为**否**：默认选项，支持DDL同步。 **说明** 多表归并功能基于任务级别，即不支持基于表级别执行多表归并。如果需要让部分表执行多表归并，另一部分不执行多表归并，您可以创建两个数据同步作业。
同步操作类型	根据业务选中需要同步的操作类型，支持的同步操作详情请参见支持同步的SQL操作，默认情况下都处于选中状态。

配置	说明
选择同步对象	在**源库对象**框中单击待同步的对象，然后单击 › 图标将其移动至**已选择对象**框。 同步对象的选择粒度为库、表。 **说明** ○ 如果选择整个库作为同步对象，那么该库中所有对象的结构变更操作会同步至目标库。 ○ 如果选择某个表作为同步对象，那么只有这个表的ADD COLUMN操作会同步至目标库。 ○ 默认情况下，同步对象的名称保持不变。如果您需要同步对象在目标集群上名称不同，请使用对象名映射功能，详情请参见设置同步对象在目标实例中的名称。
映射名称更改	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ○ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ○ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源、目标库无法连接重试时间	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

9. 上述配置完成后，单击页面右下角的**下一步**。

10. 设置待同步的表在目标库中类型。

> **说明** 选择了**结构初始化**后，您需要定义待同步的表在云原生数据仓库AnalyticDB MySQL中的**类型**、主**键列**、**分区列**等信息，详情请参见ADB 2.0 SQL手册和ADB 3.0 SQL手册。

11. 上述配置完成后，单击页面右下角的**预检查并启动**。

> **说明**
> - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
> - 如果预检查失败，单击具体检查项后的，查看失败详情。
>   - 您可以根据提示修复后重新进行预检查。
>   - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

12. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，同步作业将正式开始。
13. 等待同步作业的链路初始化完成，直至处于**同步中**状态。

    您可以在**数据同步**页面，查看数据同步作业的状态。

3.8.4. 从PolarDB MySQL同步至云原生数据仓库AnalyticDB PostgreSQL

云原生数据仓库AnalyticDB PostgreSQL版（原分析型数据库PostgreSQL版）为您提供简单、快速、经济高效的PB级云端数据仓库解决方案。通过数据传输服务DTS（Data Transmission Service），您可以将PolarDB MySQL同步至云原生数据仓库AnalyticDB PostgreSQL，帮助您快速实现对海量数据的即席查询分析、ETL处理和可视化探索。

前提条件

- PolarDB MySQL集群已开启Binlog，详情请参见如何开启Binlog。
- PolarDB MySQL集群中待同步的数据表必须具备主键。
- 已创建目标云原生数据仓库AnalyticDB PostgreSQL实例，详情请参见创建云原生数据仓库AnalyticDB PostgreSQL实例。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数

据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据同步前评估源库和目标库的性能，同时建议您在业务低峰期执行数据同步（例如源库和目标库的CPU负载在30%以下）。

- 全量初始化过程中，并发INSERT会导致目标实例的表碎片，全量初始化完成后，目标实例的表空间比源集群的表空间大。

同步限制

- 同步对象仅支持数据表。
- 不支持BIT、VARBIT、GEOMETRY、ARRAY、UUID、TSQUERY、TSVECTOR、TXID_SNAPSHOT类型的数据同步。
- 暂不支持同步前缀索引，如果源库存在前缀索引可能导致数据同步失败。
- 在数据同步时，请勿对源库的同步对象使用gh-ost或pt-online-schema-change等类似工具执行在线DDL变更，否则会导致同步失败。

支持同步的SQL操作

- DML操作：INSERT、UPDATE、DELETE。
- DDL操作：ADD COLUMN。

> **说明** 不支持CREATE TABLE操作，如果您需要将新增的表作为同步对象，则需要执行新增同步对象操作。

支持的同步架构

- 1对1单向同步。
- 1对多单向同步。
- 多对1单向同步。

术语对应关系

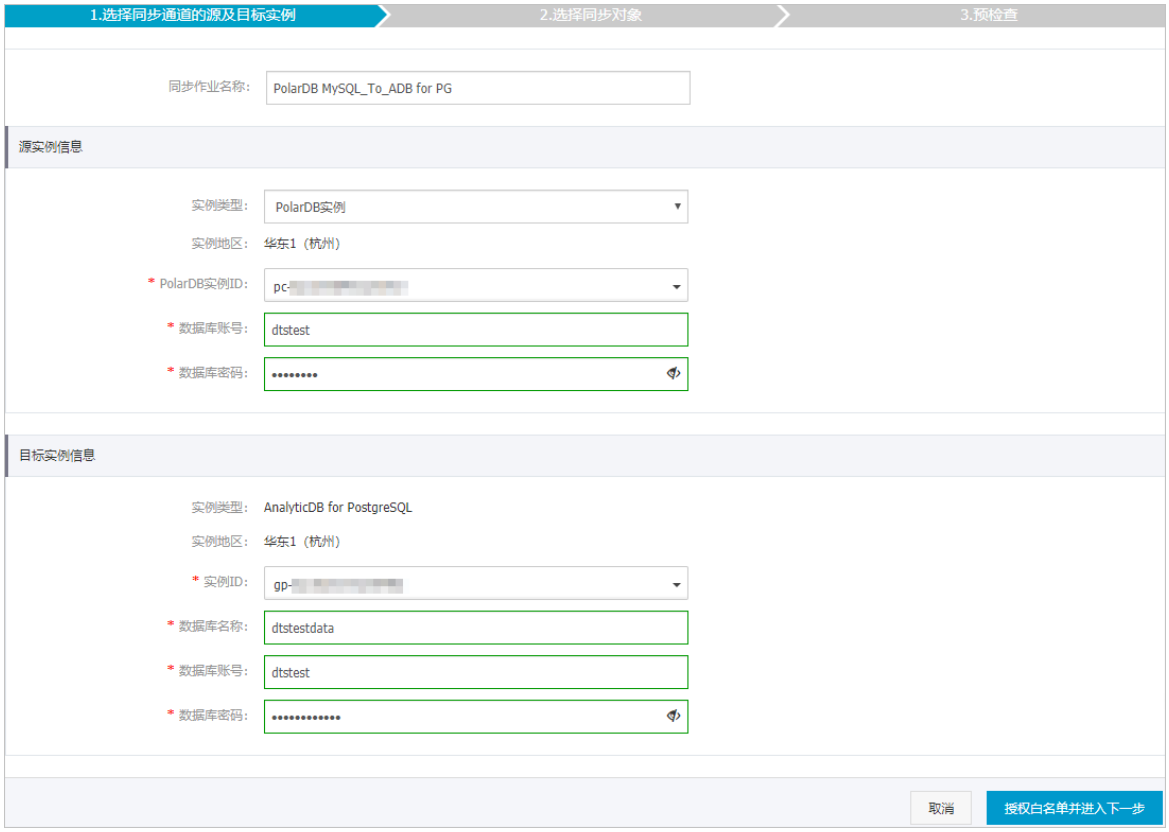
PolarDB MySQL	云原生数据仓库AnalyticDB PostgreSQL
Database	Schema
Table	Table

操作步骤

1. 购买数据同步作业，详情请参见购买流程。

   > **说明** 购买时，选择源实例为**PolarDB**、目标实例为**AnalyticDB PostgreSQL**，并选择同步拓扑为**单向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。
5. 定位至已购买的数据同步实例，单击**配置同步链路**。
6. 配置同步通道的源实例及目标实例信息。



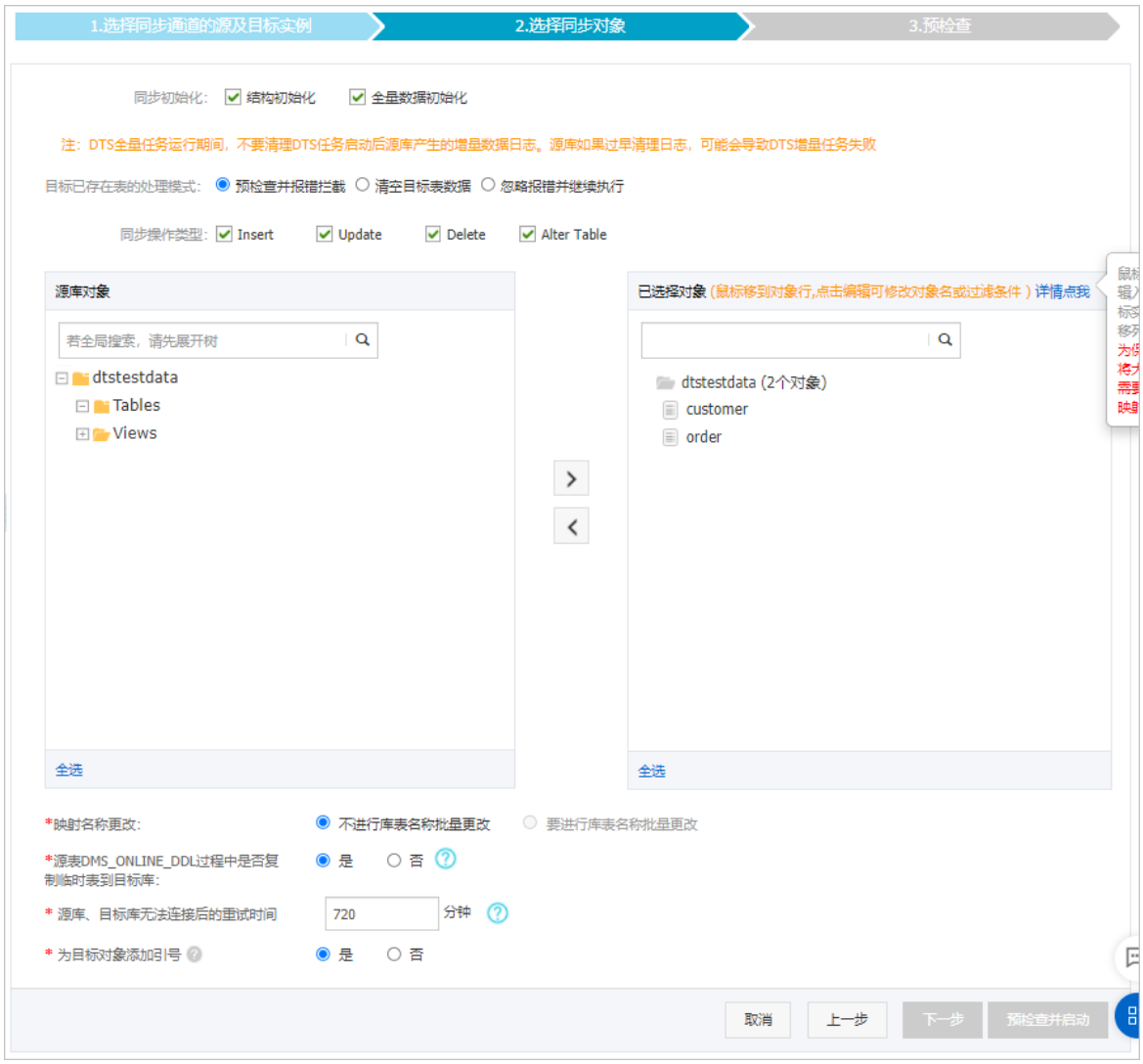
类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	固定为**PolarDB实例**。
	实例地区	购买数据同步实例时选择的源PolarDB集群的地域信息，不可变更。
	PolarDB实例ID	选择PolarDB集群ID。
	数据库账号	填入PolarDB集群的数据库账号。 **说明** 该账号需具备待同步对象的读权限。
	数据库密码	填入该数据库账号的密码。
	实例类型	固定为**AnalyticDB for PostgreSQL**，无需设置。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	实例ID	选择云原生数据仓库AnalyticDB PostgreSQL实例ID。
	数据库名称	填入云原生数据仓库AnalyticDB PostgreSQL实例中，待同步的目标表所属的数据库名称。

目标实例信息 类别	配置	说明
	数据库账号	填入云原生数据仓库AnalyticDB PostgreSQL的**初始账号**，详情请参见创建数据库账号。 **说明** 您也可以填入具备RDS_SUPERUSER权限的账号，创建方法请参见用户权限管理。
	数据库密码	填入数据库账号的密码。

7. 单击页面右下角的**授权白名单并进入下一步**。

   说明 此步骤会将DTS服务器的IP地址自动添加PolarDB MySQL和云原生数据仓库AnalyticDB PostgreSQL的白名单中，用于保障DTS服务器能够正常连接源集群和目标实例。

8. 配置同步策略及同步对象。



类别	配置	说明
同步策略配置	同步初始化	默认情况下，您需要同时选中**结构初始化**和**全量数据初始化**。预检查完成后，DTS会将源实例中待同步对象的结构及数据在目标实例中初始化，作为后续增量同步数据的基线数据。
	目标已存在表的处理模式	○ **清空目标表的数据** 在预检查阶段跳过**同名对象存在性检查**的检查项目。全量初始化之前将目标表的数据清空。适用于完成同步任务测试后的正式同步场景。 ○ **忽略报错并继续执行** 在预检查阶段跳过**同名对象存在性检查**的检查项目。全量初始化时直接追加数据。适用于多张表同步到一张表的汇总同步场景。
	同步操作类型	根据业务需求选择需要同步的操作类型： ○ Insert ○ Update ○ Delete ○ AlterTable
选择同步对象	无	在**源库对象**框中单击待同步的表，然后单击 › 图标将其移动至**已选择对象**框。 **说明** ○ 同步对象的选择粒度为表。 ○ 如果需要目标表中的列名称与源表不同，需要使用DTS的字段映射功能，详情请参见设置同步对象在目标实例中的名称。
映射名称更改	无	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。

类别	配置	说明
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	无	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ○ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ○ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源、目标库无法连接重试时间	无	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

9. 设置待同步的表在云原生数据仓库AnalyticDB PostgreSQL中的主键列和分布列信息。

说明 当您在上一步中选择了**结构初始化**才会出现该页面。关于主键列和分布列的详细说明，请参见表的约束定义和表分布键定义。

10. 上述配置完成后，单击页面右下角的**预检查并启动**。

> 说明
> - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
> - 如果预检查失败，单击具体检查项后的，查看失败详情。
>   - 您可以根据提示修复后重新进行预检查。
>   - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

11. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，同步作业将正式开始。
12. 等待同步作业的链路初始化完成，直至处于**同步中**状态。

    您可以在**数据同步**页面，查看数据同步作业的状态。

3.8.5. 从PolarDB MySQL同步至Datahub

阿里云流式数据服务DataHub是流式数据（Streaming Data）的处理平台，提供对流式数据的发布、订阅和分发功能，让您可以轻松构建基于流式数据的分析和应用。通过数据传输服务（Data Transmission Service，简称DTS），您可以将PolarDB MySQL同步至DataHub，帮助您快速实现使用流计算等大数据产品对数据实时分析。

前提条件

- DataHub实例的地域为华东1、华东2、华北2或华南1。
- DataHub实例中，已创建用作接收同步数据的项目（Project），详情请参见创建项目。
- PolarDB MySQL集群已开启Binlog，详情请参见如何开启Binlog。
- PolarDB MySQL中待同步的表需具备主键或唯一约束。

功能限制

- 不支持全量数据初始化，即DTS不会将源PolarDB集群中同步对象的存量数据同步至目标DataHub实例。
- 仅支持表级别的数据同步。
- 不支持新增列的数据同步，即源数据表新增了某个列，该列的数据不会同步至目标DataHub实例。
- 数据同步的过程中，请勿对源库中待同步的表执行DDL变更，否则会导致同步失败。

支持同步的SQL操作

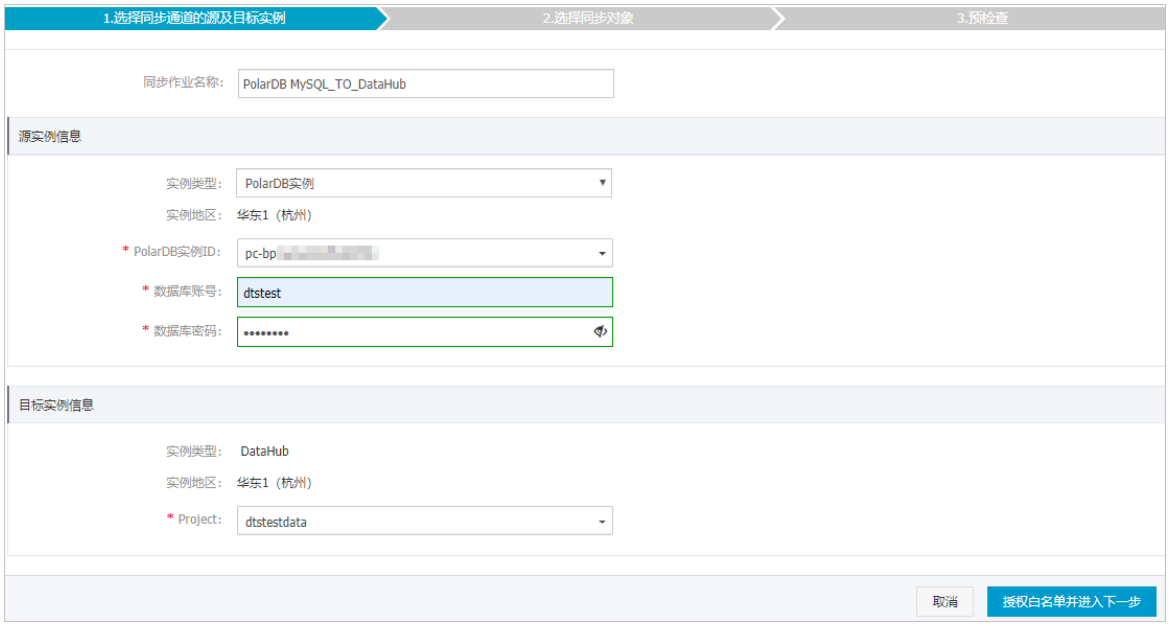
INSERT、UPDATE、DELETE。

操作步骤

1. 购买数据同步作业，详情请参见购买流程。

> **说明** 购买时，选择源实例为PolarDB、目标实例为DataHub，并选择同步拓扑为**单向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。
5. 定位至已购买的数据同步实例，单击**配置同步链路**。
6. 配置同步作业的源实例及目标实例信息。

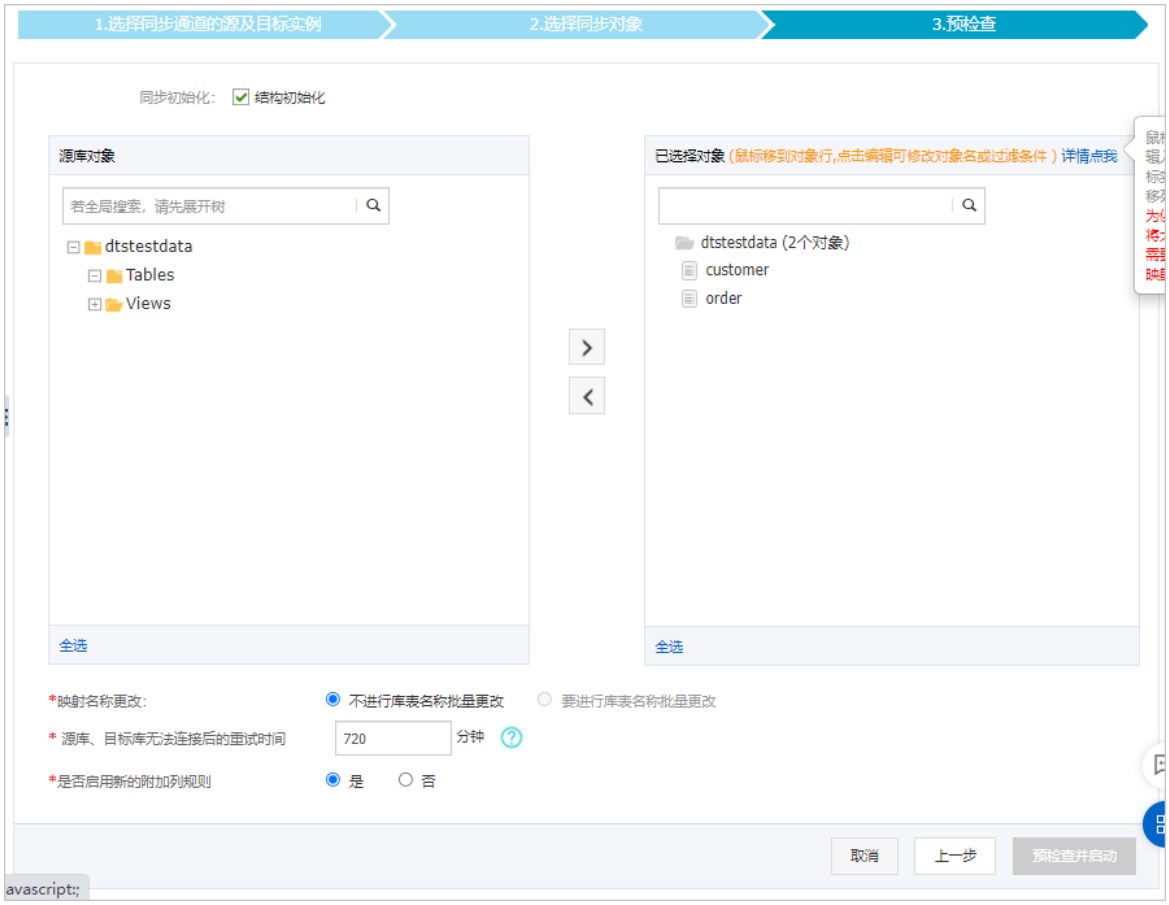


类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	固定为**PolarDB实例**，不可变更。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	PolarDB实例ID	选择源PolarDB集群ID。
	数据库账号	填入PolarDB集群的数据库账号。
	数据库密码	填入数据库账号对应的密码。
目标实例信息	实例类型	固定为**DataHub**，不可变更。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	Project	选择DataHub实例的**Project**。

7. 单击页面右下角的**授权白名单并进入下一步**。

> **说明** 此步骤会将DTS服务器的IP地址自动添加到PolarDB集群的白名单中，用于保障DTS服务器能够正常连接PolarDB集群。

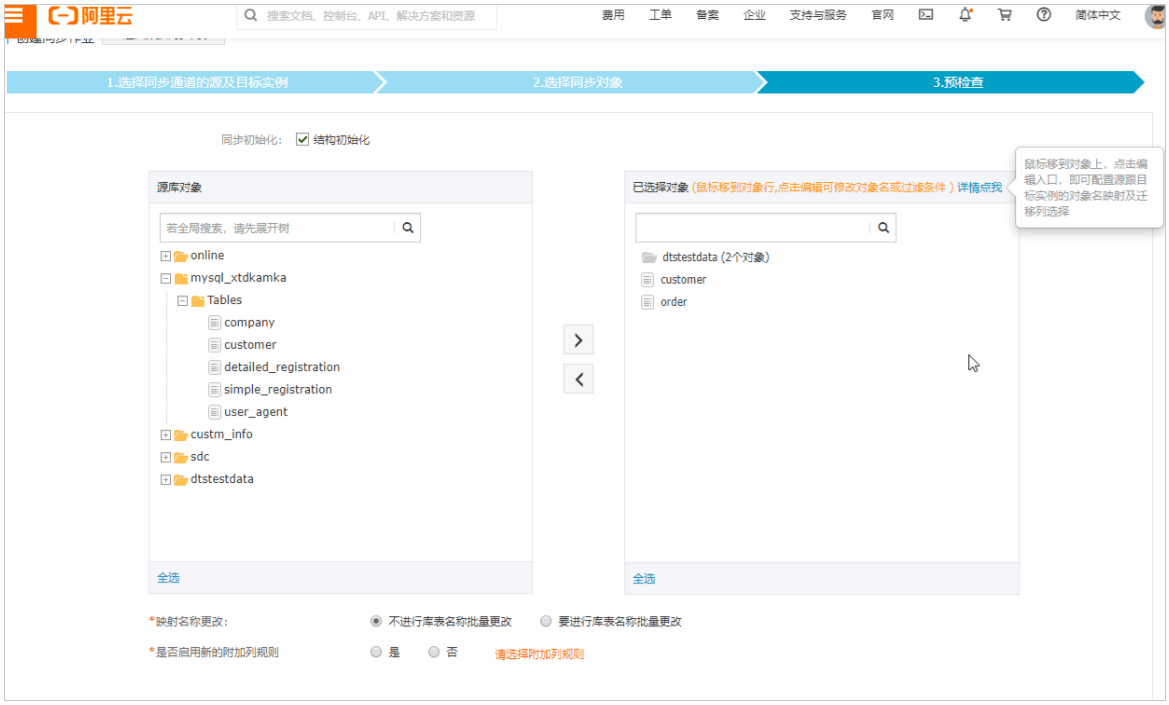
8. 配置同步策略和同步对象。



配置	说明
同步初始化	勾选**结构初始化**。 > **说明** 勾选**结构初始化**后，在数据同步作业的初始化阶段，DTS会将同步对象的结构信息（例如表结构）同步至目标DataHub实例。

配置	说明
选择同步对象	在**源库对象**框中单击待迁移的对象，然后单击 › 将其移动至**已选择对象**框。 **说明** • 同步对象的选择粒度为表。 • 默认情况下，同步对象的名称保持不变。如果您需要改变同步对象在目标实例中的名称，需要使用对象名映射功能，详情请参见设置同步对象在目标实例中的名称。
选择附加列规则	DTS在将数据同步到DataHub时，会在同步的目标Topic中添加一些附加列。如果附加列和目标Topic中已有的列出现名称冲突将会导致数据同步失败。您需要根据业务需求选择**是否启用新的附加列规则**为**是**或**否**。 **警告** 在选择附加列规则前，您需要评估附加列和目标Topic中已有的列是否会出现名称冲突。关于附加列的规则和定义说明，请参见附加列名称和定义说明。
映射名称更改	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 • **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 • **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源、目标库无法连接重试时间	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

9. （可选）将鼠标指针放置在**已选择对象**框中待同步的Topic名上，单击对象后出现的**编辑**，然后在弹出的对话框中设置Shardkey（即用于分区的key）。



10. 上述配置完成后，单击页面右下角的**预检查并启动**。

> **说明**
> - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
> - 如果预检查失败，单击具体检查项后的，查看失败详情。
>   - 您可以根据提示修复后重新进行预检查。
>   - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

11. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，同步作业将正式开始。
12. 等待同步作业的链路初始化完成，直至处于**同步中**状态。

    您可以在**数据同步**页面，查看数据同步作业的状态。

Topic结构定义说明

DTS在将数据变更同步至DataHub实例的Topic时，目标Topic中除了存储变更数据外，还会新增一些附加列用于存储元信息，示例如下。

> **说明** 本案例中的业务字段为 `id`、`name`、`address`，由于在配置数据同步时选用的是旧版附加列规则，DTS会为业务字段添加 `dts_` 的前缀。

dts_id	dts_name	dts_address	dts_record_id	dts_operation_flag	dts_instance_id	dts_db_name	dts_table_name	dts_utc_timestamp	dts_before_flag	dts_after_flag
10006	[redacted]	曲靖市	1574832130000000000	U		dtstestdata	customer	1574832130	Y	N
10006	[redacted]	杭州市	1574832130000000000	U		dtstestdata	customer	1574832130	N	Y
10009	[redacted]	马鞍山市	1574832919000000000	D		dtstestdata	customer	1574832919	Y	N
10112	[redacted]	北京市	1574832919000000000	I		dtstestdata	customer	1574832919	N	Y

结构定义说明：

旧版附加列名称	新版附加列名称	数据类型	说明
`dts_record_id`	`new_dts_sync_dts_record_id`	String	增量日志的记录ID，为该日志唯一标识。 **说明** • 正常情况下是全局唯一自增的，在容灾的情况下会有回退且无法保证自增和唯一。 • 如果增量日志的操作类型为UPDATE，那么增量更新会被拆分成两条记录（分别记录更新前和更新后的值），且 `dts_record_id` 的值相同。
`dts_operation_flag`	`new_dts_sync_dts_operation_flag`	String	操作类型，取值： • I：INSERT操作。 • D：DELETE操作。 • U：UPDATE操作。
`dts_instance_id`	`new_dts_sync_dts_instance_id`	String	数据库的server ID。暂不支持显示实际的值，目前固定为 `null` 。
`dts_db_name`	`new_dts_sync_dts_db_name`	String	数据库名称。
`dts_table_name`	`new_dts_sync_dts_table_name`	String	表名。
`dts_utc_timestamp`	`new_dts_sync_dts_utc_timestamp`	String	操作时间戳，即binlog的时间戳（UTC 时间）。
`dts_before_flag`	`new_dts_sync_dts_before_flag`	String	所有列的值是否更新前的值，取值：Y或N。
`dts_after_flag`	`new_dts_sync_dts_after_flag`	String	所有列的值是否更新后的值，取值：Y或N。

关于dts_before_flag和dts_after_flag的补充说明

对于不同的操作类型，增量日志中的 `dts_before_flag` 和 `dts_after_flag` 定义如下：

- INSERT

当操作类型为INSERT时，所有列的值为新插入的值，即为更新后的值，所以 `dts_before_flag` 取值为N， `dts_after_flag` 取值为Y，示例如下。

dts_id	dts_name	dts_address	dts_record_id	dts_operation_flag	dts_instance_id	dts_db_name	dts_table_name	dts_utc_timestamp	dts_before_flag	dts_after_flag
10112	[redacted]	北京市	1574832919000000000	I		dtstestdata	customer	1574832919	N	Y

- UPDATE

  当操作类型为UPDATE时，DTS会将UPDATE操作拆为两条增量日志。这两条增量日志的 `dts_record_id` 、 `dts_operation_flag` 及 `dts_utc_timestamp` 对应的值相同。

  第一条增量日志记录了更新前的值，所以 `dts_before_flag` 取值为Y， `dts_after_flag` 取值为N。第二条增量日志记录了更新后的值，所以 `dts_before_flag` 取值为N， `dts_after_flag` 取值为Y，示例如下。

dts_id	dts_name	dts_address	dts_record_id	dts_operation_flag	dts_instance_id	dts_db_name	dts_table_name	dts_utc_timestamp	dts_before_flag	dts_after_flag
10006	[redacted]	曲靖市	1574832130000000000	U		dtstestdata	customer	1574832130	Y	N
10006	[redacted]	杭州市	1574832130000000000	U		dtstestdata	customer	1574832130	N	Y

- DELETE

  当操作类型为DELETE时，增量日志中所有的列值为被删除的值，即列值不变，所以 `dts_before_flag` 取值为Y， `dts_after_flag` 取值为N，示例如下。

dts_id	dts_name	dts_address	dts_record_id	dts_operation_flag	dts_instance_id	dts_db_name	dts_table_name	dts_utc_timestamp	dts_before_flag	dts_after_flag
10009	[redacted]	马鞍山市	1574832919000000000	D		dtstestdata	customer	1574832919	Y	N

后续操作

配置完数据同步作业后，您可以对同步到DataHub实例中的数据执行计算分析。更多详情，请参见阿里云实时计算。

3.8.6. 从PolarDB MySQL同步到Kafka

Kafka是应用较为广泛的分布式、高吞吐量、高可扩展性消息队列服务，普遍用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等大数据领域，是大数据生态中不可或缺的产品之一。通过数据传输服务DTS（Data Transmission Service），您可以将PolarDB MySQL同步至自建Kafka集群，扩展消息处理能力。

前提条件

- Kafka集群的版本为0.10.1.0-2.7.0版本。
- PolarDB MySQL已开启Binlog，详情请参见如何开启Binlog。

注意事项

如果源数据库没有主键或唯一约束，且所有字段没有唯一性，可能会导致目标数据库中出现重复数据。

功能限制

- 仅支持表粒度的数据同步。
- 不支持自动调整同步对象。

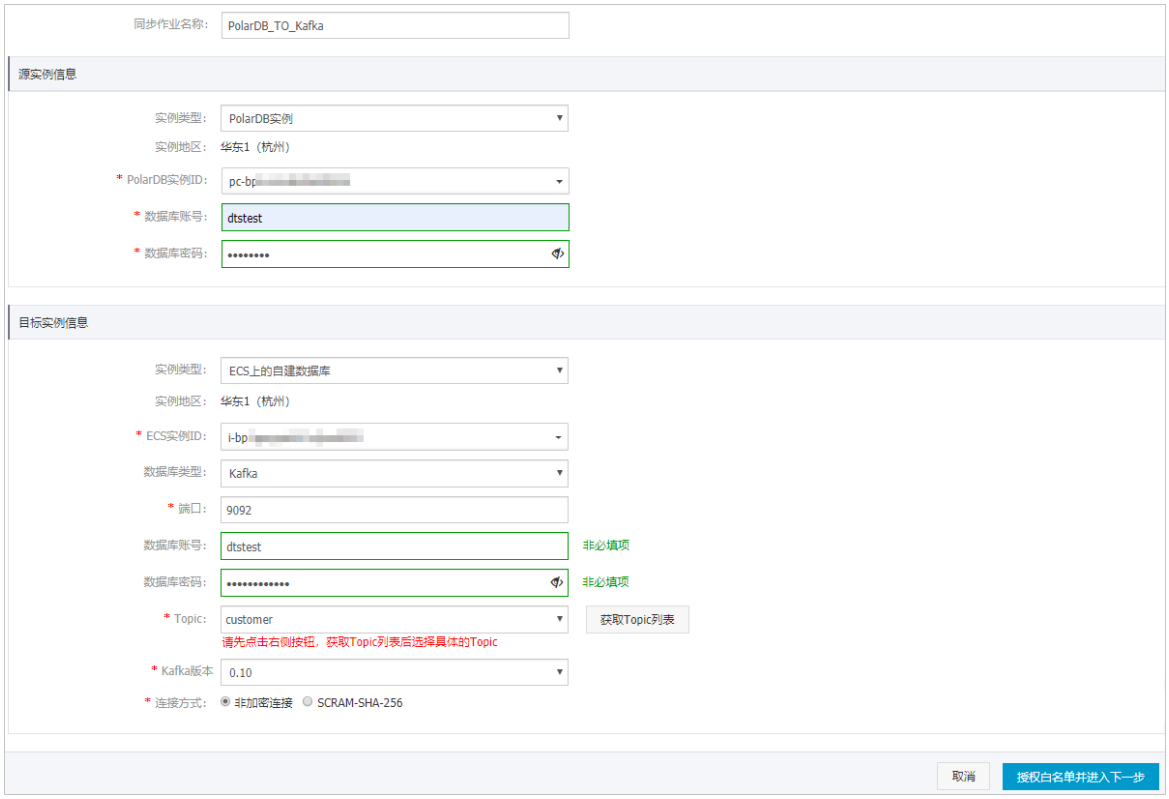
> **说明** 如果在同步的过程中，对源库中待同步的表执行了重命名操作，且重命名后的名称不在同步对象中，那么该表将不再被同步到目标Kafka集群中。如果该表还需要同步，那么您需要新增同步对象。

操作步骤

1. 购买数据同步作业，详情请参见购买流程。

   > 说明　购买时，选择源实例为PolarDB、目标实例为Kafka，并选择同步拓扑为**单向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。
5. 定位至已购买的数据同步实例，单击**配置同步链路**。
6. 配置源实例及目标实例信息。



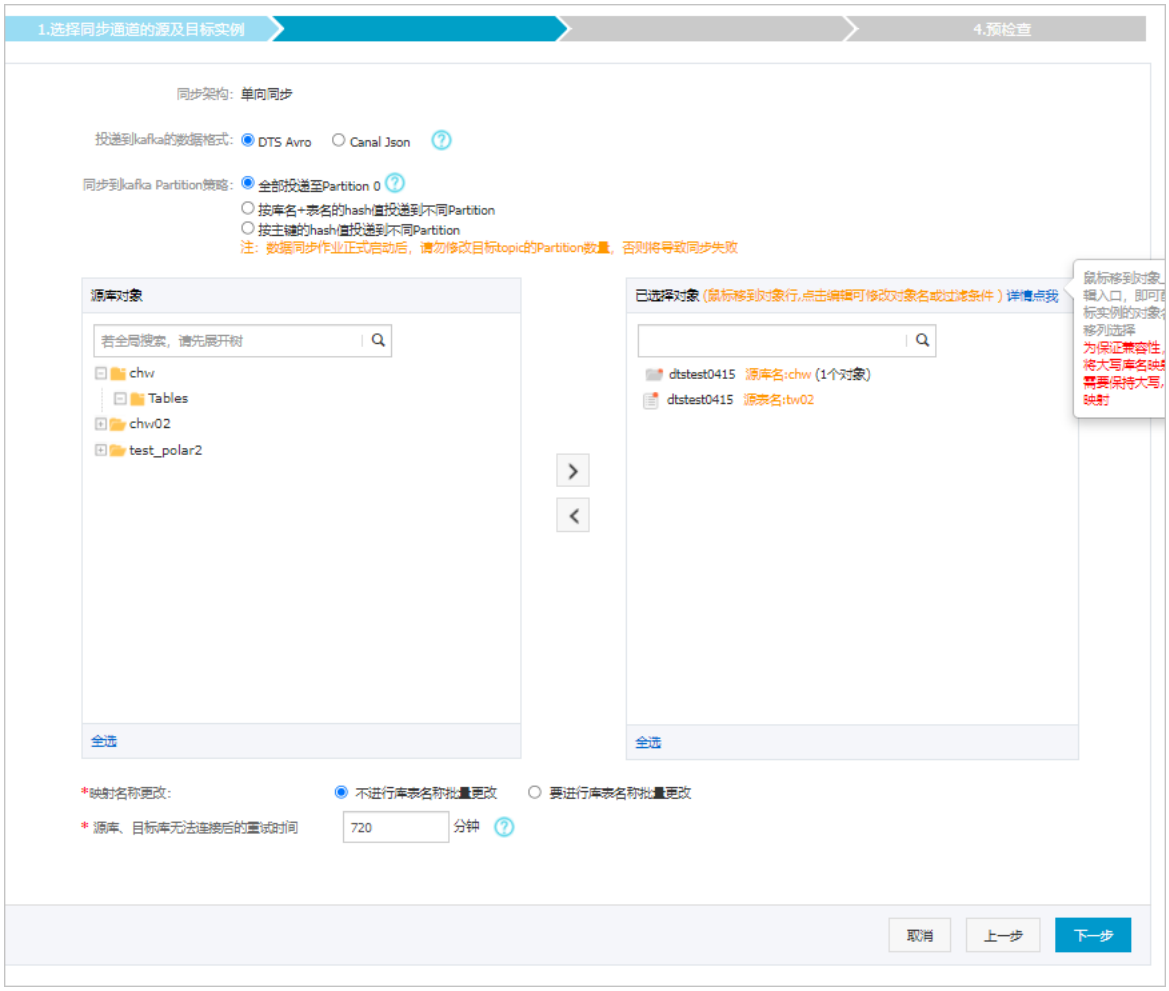
类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	固定为**PolarDB实例**，不可变更。
	实例地区	购买数据同步实例时选择的源实例地域，不可变更。
	PolarDB实例ID	选择PolarDB集群ID。
	数据库账号	填入PolarDB集群的数据库账号，需要具备待同步数据库的读权限。
	数据库密码	填入该数据库账号的密码。

类别	配置	说明
目标实例信息	实例类型	根据Kafka集群的部署位置选择，本文以**ECS上的自建数据库**为例介绍配置流程。 **说明** 当选择为其他实例类型时，您还需要执行相应的准备工作，详情请参见准备工作概览。
	实例地区	购买数据同步实例时选择的目标实例地域，不可变更。
	ECS实例ID	选择部署了Kafka集群的ECS实例ID。
	数据库类型	选择为**Kafka**。
	端口	Kafka集群对外提供服务的端口，默认为9092。
	数据库账号	填入Kafka集群的用户名，如Kafka集群未开启验证可不填写。
	数据库密码	填入Kafka集群用户名对应的密码，如Kafka集群未开启验证可不填写。
	Topic	单击右侧的**获取Topic列表**，然后在下拉框中选择具体的Topic。
	Kafka版本	根据目标Kafka集群版本，选择对应的版本信息。
	连接方式	根据业务及安全需求，选择**非加密连接**或**SCRAM-SHA-256**。

7. 单击页面右下角的**授权白名单并进入下一步**。

   说明 此步骤会将DTS服务器的IP地址自动添加到源PolarDB集群的白名单和目标ECS实例的内网入方向安全组规则中，用于保障DTS服务器能够正常连接源和目标实例。

8. 配置同步策略和同步对象信息。



配置	说明
投递到kafka的数据格式	同步到Kafka集群中的数据以avro格式或者Canal Json格式存储，定义详情请参见Kafka集群的数据存储格式。
同步到Kafka Partition策略	根据业务需求选择同步的策略，详细介绍请参见Kafka Partition同步策略说明。
同步对象	在**源库对象**区域框中，选择需要同步的对象（选择的粒度为表），然后单击 > 图标将其移动到**已选对象**区域框中。 **说明** DTS会自动将表名映射为步骤6选择的Topic名称。如果需要更换同步的目标Topic，请参见步骤9 。
映射名称更改	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。

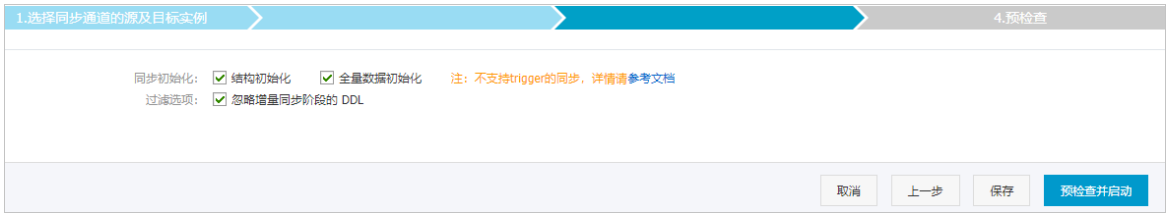
配置	说明
源、目标库无法连接重试时间	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

9. （可选）在**已选择对象**区域框中，将鼠标指针放置在目标Topic名上，然后单击Topic名后出现的**编辑**，在弹出的对话框中设置源表在目标Kafka集群中的Topic名称、Topic的Partition数量、Partition Key等信息。

配置	说明

配置	说明
数据库表名称	设置源表同步到的目标Topic名称。 **说明** 如果设置的Topic名称在目标Kafka集群中不存在，您还需要设置该Topic的Partition数量。
过滤条件	○ 过滤条件支持标准的SQL WHERE语句（仅支持 `=` 、 `!=` 、 `<` 和 `>` 操作符），只有满足WHERE条件的数据才会被同步到目标Topic。本案例填入 `id>1000` 。 ○ 过滤条件中如需使用引号，请使用单引号（'），例如 `address in('hangzhou','shanghai')` 。
设置新Topic的Partition数量	在下拉列表中，选择新Topic的Partition数量。 **说明** 只有当设置的目标Topic名称在目标Kafka集群中不存在时，您才需要配置本参数。
设置Partition Key	当您在步骤8中选择同步策略为**按主键的hash值投递到不同Partition**时，您可以配置本参数，指定单个或多个列作为Partition Key来计算Hash值，DTS将根据计算得到的Hash值将不同的行投递到目标Topic的各Partition中。

10. 上述配置完成后单击页面右下角的**下一步**。
11. 配置同步初始化的高级配置信息。



配置	说明
同步初始化	默认选择**结构初始化**和**全量数据初始化**，DTS会在增量数据同步之前，将源库中待同步对象的结构和存量数据，同步到目标库。
过滤选项	默认选择**忽略增量同步阶段的 DDL**，即增量同步阶段源库执行的DDL操作不会被DTS同步至目标库。

12. 上述配置完成后，单击页面右下角的**预检查并启动**。

> 说明
> - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
> - 如果预检查失败，单击具体检查项后的，查看失败详情。
>   - 您可以根据提示修复后重新进行预检查。
>   - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

13. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，数据同步作业正式开始。

    您可以在**数据同步**页面，查看数据同步状态。

3.8.7. 从ECS上的自建MySQL同步至PolarDB MySQL

PolarDB是阿里巴巴自主研发的下一代关系型分布式云原生数据库，可完全兼容MySQL，具备简单易用、高性能、高可靠、高可用等优势。通过数据传输服务DTS（Data Transmission Service），您可以将自建的MySQL数据库同步至PolarDB MySQL，本文以ECS上的自建MySQL为例介绍配置流程。

前提条件

已创建PolarDB MySQL集群，详情请参见创建PolarDB MySQL集群。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据同步前评估源库和目标库的性能，同时建议您在业务低峰期执行数据同步（例如源库和目标库的CPU负载在30%以下）。
-
- 全量初始化过程中，并发INSERT会导致目标集群的表碎片，全量初始化完成后，目标集群的表空间比源库的表空间大。
- 如果数据同步的源库没有主键或唯一约束，且记录的全字段没有唯一性，可能会出现重复数据。

支持同步的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE、REPLACE

操作类型	SQL操作语句
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

功能限制

- 不兼容触发器

  当同步对象为整个库，且库中的触发器（TRIGGER）会更新库内某个表时，可能导致源和目标库的数据不一致。相关解决方案请参见源库存在触发器时如何配置同步作业。

- RENAME TABLE限制

  RENAME TABLE操作可能导致同步数据不一致。例如同步对象只包含某个表，如果同步过程中源实例对该表执行了重命名操作，那么该表的数据将不会同步到目标库。为避免该问题，您可以在数据同步配置时将该表所属的整个数据库作为同步对象。

准备工作

为自建MySQL创建账号并设置binlog

> **说明** 用于数据同步的数据库账号需具备待同步对象的REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和所有同步对象的SELECT权限。

支持的同步架构

- 一对一单向同步
- 一对多单向同步
- 级联单向同步
- 多对一单向同步

关于各类同步架构的介绍及注意事项，请参见数据同步拓扑介绍。

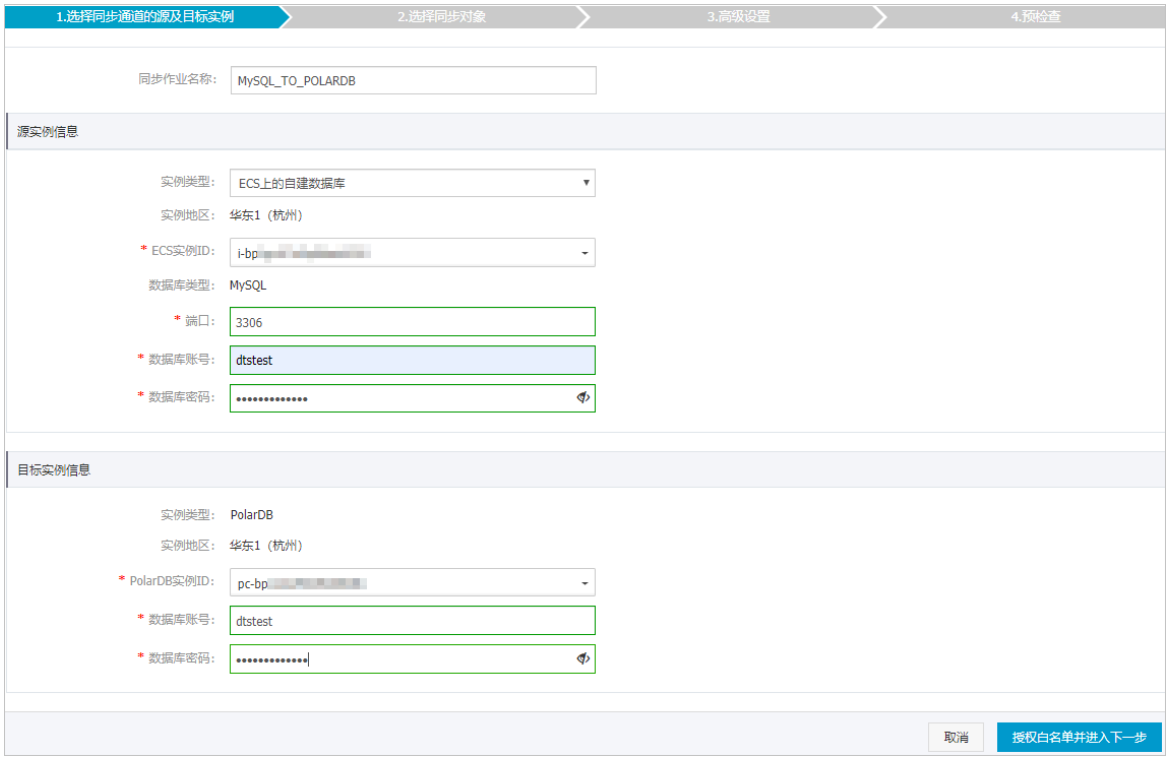
操作步骤

1. 购买数据同步作业，详情请参见购买流程。

   > **说明** 购买时，选择源实例为MySQL、目标实例为PolarDB，并选择同步拓扑为**单向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。

5. 定位至已购买的数据同步实例，单击**配置同步链路**。
6. 配置同步通道的源实例及目标实例信息。



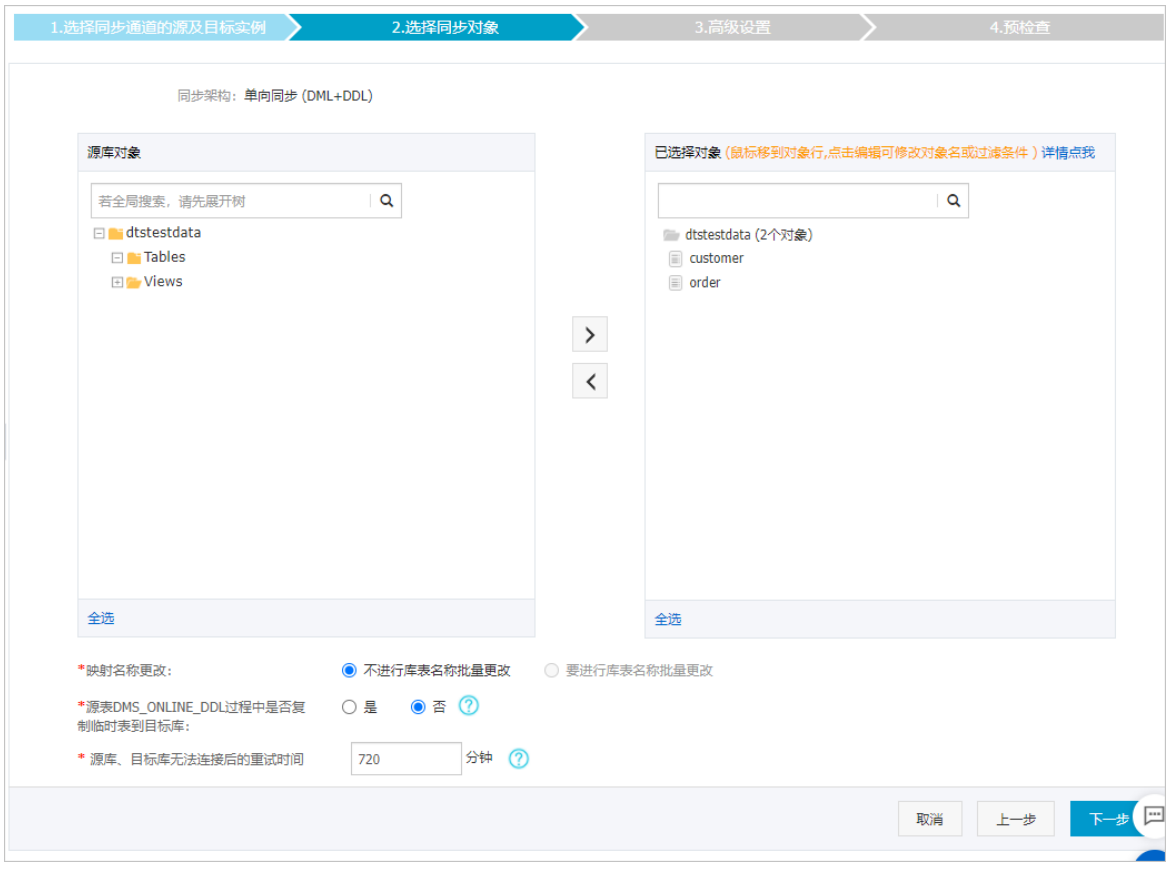
类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	选择**ECS上的自建数据库**。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	ECS实例ID	选择自建MySQL数据库所属的ECS实例ID。
	数据库类型	固定为**MySQL**，不可变更。
	端口	填入自建MySQL的数据库服务端口。
	数据库账号	填入连接自建MySQL的数据库账号。 **说明** 用于数据同步的数据库账号需具备REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和所有同步对象的SELECT权限。
	数据库密码	填入数据库账号的密码。
	实例类型	固定为**PolarDB**，不可变更。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。

类别	配置	说明
目标实例信息	PolarDB实例ID	选择目标PolarDB集群ID。
	数据库账号	填入连接PolarDB集群的数据库账号。 说明 用于数据同步的数据库账号需具备目标同步对象的ALL权限。
	数据库密码	填入数据库账号的密码。

7. 单击页面右下角的**授权白名单并进入下一步**。

   说明 此步骤会将DTS服务器的IP地址自动添加到ECS实例的内网入方向安全组规则和目标PolarDB集群的白名单中，用于保障DTS服务器能够正常连接源实例和目标集群。

8. 配置目标已存在表的处理模式和同步对象。

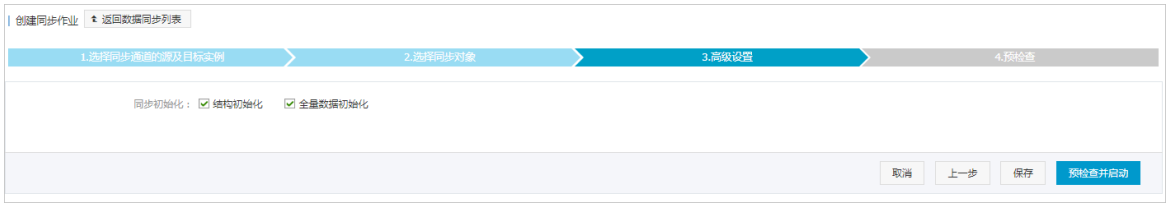


配置项目	配置说明

配置项目	配置说明
目标已存在表的处理模式	○ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据同步作业不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以设置同步对象在目标实例中的名称来避免表名冲突。 ○ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，如果在目标库遇到与源库主键的值相同的记录，在初始化阶段会保留目标库中的该条记录；在增量同步阶段则会覆盖目标库的该条记录。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。
选择同步对象	在**源库对象**框中单击待同步的对象，然后单击 › 图标将其移动至**已选择对象**框。 同步对象的选择粒度为库、表。 **说明** ○ 如果选择整个库作为同步对象，那么该库中所有对象的结构变更操作都会同步至目标库。 ○ 默认情况下，同步对象的名称保持不变。如果您需要改变同步对象在目标集群中的名称，请使用对象名映射功能，详情请参见设置同步对象在目标实例中的名称。
映射名称更改	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。

配置项目	配置说明
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ○ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ○ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。
源、目标库无法连接重试时间	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

9. 上述配置完成后，单击页面右下角的**下一步**。
10. 配置同步初始化的高级配置信息。



说明 同步初始化类型细分为：结构初始化，全量数据初始化。选中**结构初始化**和**全量数据初始化**后，DTS会在增量数据同步之前，将源数据库中待同步对象的结构和存量数据，同步到目标数据库。

11. 上述配置完成后，单击页面右下角的**预检查并启动**。

说明
- 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
- 如果预检查失败，单击具体检查项后的，查看失败详情。
  - 您可以根据提示修复后重新进行预检查。
  - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

12. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，同步作业将正式开始。

13. 等待同步作业的链路初始化完成，直至处于**同步中**状态。

    您可以在**数据同步**页面，查看数据同步作业的状态。

3.8.8. 从通过专线、VPN网关或智能接入网关接入的自建MySQL同步至PolarDB MySQL

PolarDB是阿里巴巴自主研发的下一代关系型分布式云原生数据库，可完全兼容MySQL，具备简单易用、高性能、高可靠、高可用等优势。通过数据传输服务DTS（Data Transmission Service），您可以将自建的MySQL数据库同步至PolarDB MySQL，本文以通过专线、VPN网关或智能接入网关接入的自建MySQL为例介绍配置流程。

前提条件

- 自建MySQL数据库版本为5.1、5.5、5.6、5.7或8.0版本。
- 已经将自建MySQL数据库通过专线、VPN网关或智能接入网关接入至阿里云专有网络，详情请参见本地IDC接入至阿里云方案概览。

  > **说明** 接入至阿里云专有网络后，还需要允许DTS服务器的IP地址访问自建数据库所属的网络，详情请参见通过VPN网关实现本地IDC与DTS云服务互通。

- 已创建PolarDB MySQL集群，详情请参见创建PolarDB MySQL集群。

  > **说明** PolarDB MySQL集群的存储空间须大于自建MySQL数据库占用的存储空间。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，在数据库性能较差、规格较低或业务量较大的情况下（例如源库有大量慢SQL、存在无主键表或目标库存在死锁等），可能会加重数据库压力，甚至导致数据库服务不可用。因此您需要在执行数据同步前评估源库和目标库的性能，同时建议您在业务低峰期执行数据同步（例如源库和目标库的CPU负载在30%以下）。
-
- 全量初始化过程中，并发INSERT会导致目标集群的表碎片，全量初始化完成后，目标集群的表空间比源库的表空间大。
- 如果数据同步的源集群没有主键或唯一约束，且记录的全字段没有唯一性，可能会出现重复数据。

支持同步的SQL操作

操作类型	SQL操作语句
DML	INSERT、UPDATE、DELETE、REPLACE

操作类型	SQL操作语句
DDL	• ALTER TABLE、ALTER VIEW • CREATE FUNCTION、CREATE INDEX、CREATE PROCEDURE、CREATE TABLE、CREATE VIEW • DROP INDEX、DROP TABLE • RENAME TABLE • TRUNCATE TABLE

支持的同步架构

- 一对一单向同步
- 一对多单向同步
- 级联单向同步
- 多对一单向同步

关于各类同步架构的介绍及注意事项，请参见数据同步拓扑介绍。

功能限制

- 不兼容触发器

  当同步对象为整个库，且库中的触发器（TRIGGER）会更新库内某个表时，可能导致源和目标库的数据不一致。相关解决方案请参见源库存在触发器时如何配置同步作业。

- RENAME TABLE限制

  RENAME TABLE操作可能导致同步数据不一致。例如同步对象只包含某个表，如果同步过程中源实例对该表执行了重命名操作，那么该表的数据将不会同步到目标库。为避免该问题，您可以在数据同步配置时将该表所属的整个数据库作为同步对象。

准备工作

为自建MySQL创建账号并设置binlog

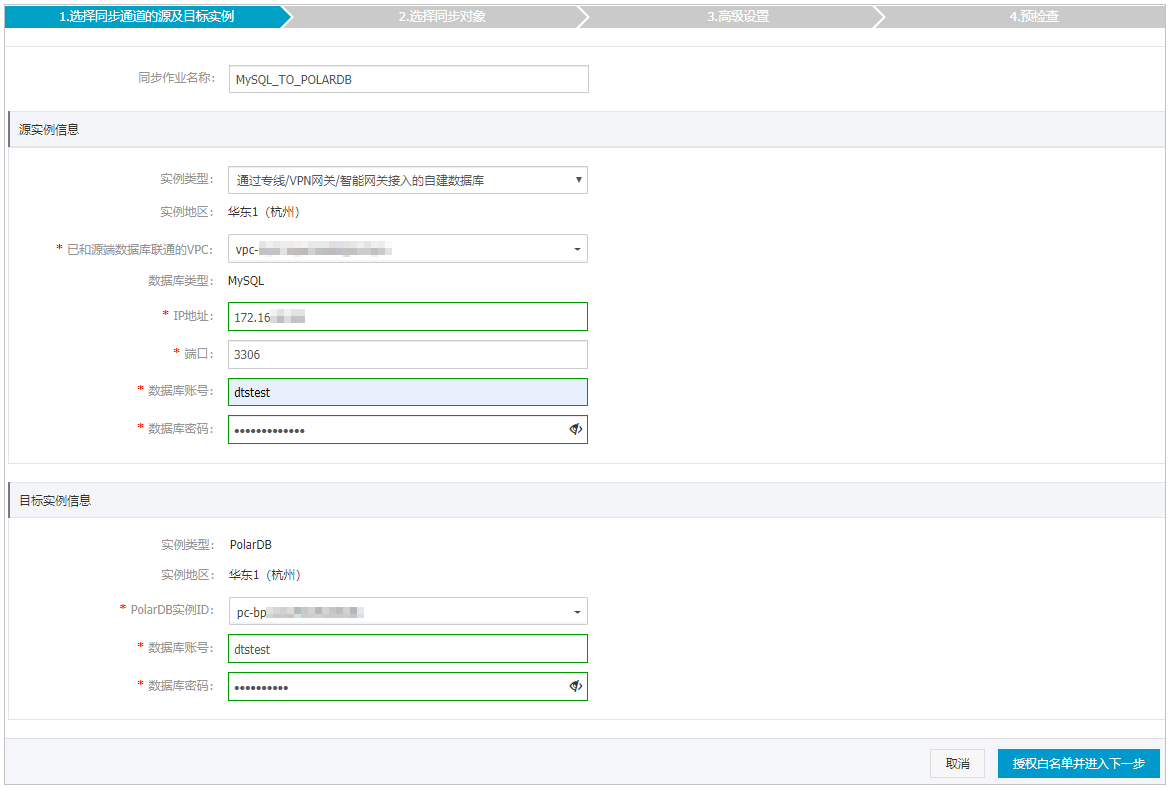
> **说明** 用于数据同步的数据库账号需具备REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和所有同步对象的SELECT权限。

操作步骤

1. 购买数据同步作业，详情请参见购买流程。

   > **说明** 购买时，选择源实例为MySQL、目标实例为PolarDB，并选择同步拓扑为**单向同步**。

2. 登录数据传输控制台。
3. 在左侧导航栏，单击**数据同步**。
4. 在**同步作业列表**页面顶部，选择同步的目标实例所属地域。
5. 定位至已购买的数据同步实例，单击**配置同步链路**。
6. 配置同步通道的源实例及目标实例信息。



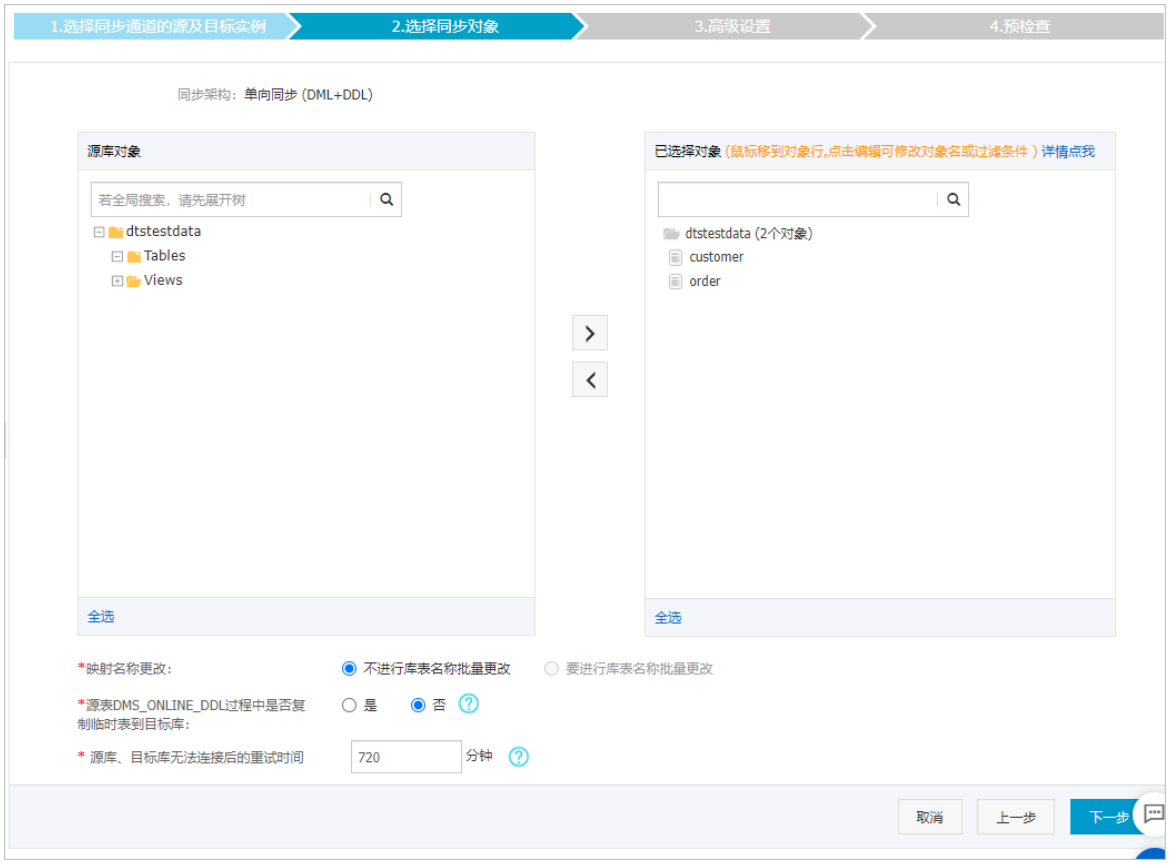
类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	选择**通过专线/VPN网关/智能接入网关接入的自建数据库**。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	已和源端数据库联通的VPC	选择自建数据库接入的VPC ID。
	数据库类型	固定为**MySQL**，不可变更。
	IP地址	填入自建MySQL数据库的服务器IP地址。
	端口	填入自建MySQL的数据库服务端口。
	数据库账号	填入为自建MySQL创建的数据库账号，详情请参见准备工作。
	数据库密码	填入该数据库账号的密码。
目标实例信息	实例类型	固定为**PolarDB**，不可变更。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	PolarDB实例ID	选择目标PolarDB集群ID。

目标实例信息 类别	配置	说明
	数据库账号	填入PolarDB集群的数据库账号。 说明 该数据库账号需具备目标同步对象的ALL权限。
	数据库密码	填入该数据库账号的密码。

7. 单击页面右下角的**授权白名单并进入下一步**。

> 说明
> - 如果源或目标数据库是阿里云数据库实例（例如RDS MySQL、云数据库MongoDB版等）或ECS上的自建数据库，DTS会自动将对应地区DTS服务的IP地址添加到阿里云数据库实例的白名单或ECS的安全规则中，您无需手动添加，请参见DTS服务器的IP地址段。
> - DTS任务完成或释放后，建议您手动删除添加的DTS服务器IP地址段。

8. 配置目标已存在表的处理模式和同步对象。

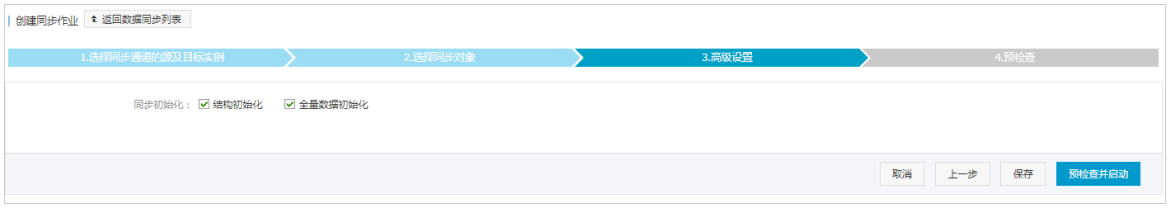


配置项目	配置说明

配置项目	配置说明
目标已存在表的处理模式	○ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据同步作业不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以设置同步对象在目标实例中的名称来避免表名冲突。 ○ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，如果在目标库遇到与源库主键的值相同的记录，在初始化阶段会保留目标库中的该条记录；在增量同步阶段则会覆盖目标库的该条记录。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。
选择同步对象	在**源库对象**框中单击待同步的对象，然后单击 › 图标将其移动至**已选择对象**框。 同步对象的选择粒度为库、表。 **说明** ○ 如果选择整个库作为同步对象，那么该库中所有对象的结构变更操作都会同步至目标库。 ○ 默认情况下，同步对象的名称保持不变。如果您需要改变同步对象在目标集群中的名称，请使用对象名映射功能，详情请参见设置同步对象在目标实例中的名称。
映射名称更改	如需更改同步对象在目标实例中的名称，请使用对象名映射功能，详情请参见库表列映射。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ○ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ○ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。

配置项目	配置说明
源、目标库无法连接重试时间	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

9. 上述配置完成后，单击页面右下角的**下一步**。
10. 配置同步初始化的高级配置信息。



说明 同步初始化类型细分为：结构初始化，全量数据初始化。选中**结构初始化**和**全量数据初始化**后，DTS会在增量数据同步之前，将源数据库中待同步对象的结构和存量数据，同步到目标数据库。

11. 上述配置完成后，单击页面右下角的**预检查并启动**。

说明

- 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
- 如果预检查失败，单击具体检查项后的，查看失败详情。
  - 您可以根据提示修复后重新进行预检查。
  - 如无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

12. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，同步作业将正式开始。
13. 等待同步作业的链路初始化完成，直至处于**同步中**状态。

    您可以在**数据同步**页面，查看数据同步作业的状态。

4.内核功能

4.1. 内核说明

4.1.1. 兼容性说明

PolarDB MySQL引擎100%兼容原生MySQL和RDS MySQL，您可以在不修改应用程序任何代码和配置的情况下，将MySQL数据库迁移至PolarDB。

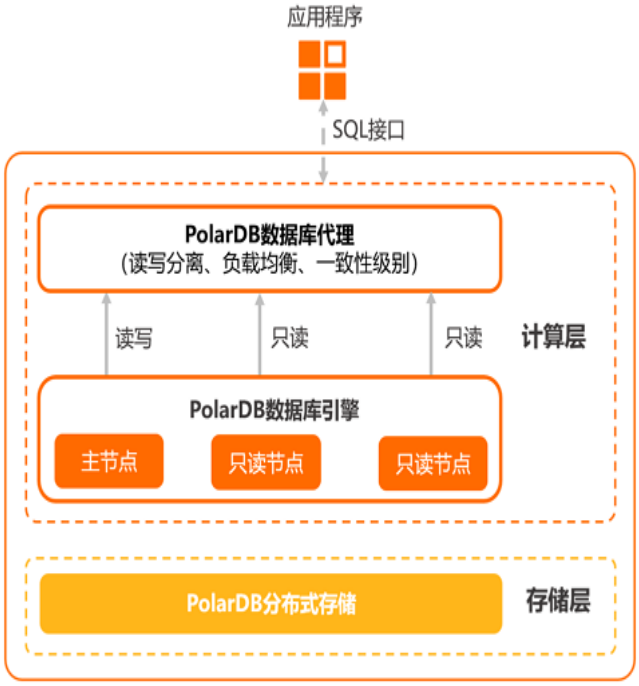
PolarDB MySQL引擎内核兼容性说明如下：

- 兼容ANSI/ISO SQL标准，PolarDB MySQL引擎支持修改SQL兼容模式为ANSI，您可以将集群参数sql_mode修改为*ANS*实现该需求。如何修改集群参数，请参见设置集群参数。
- 支持0至3.51版本的ODBC。
- 支持W3C和XPath标准的XML功能。
- PolarDB MySQL引擎5.7和8.0版本支持RFC 7159和ECMAScript标准（ECMA-262）的原生JSON数据类型。

4.1.2. 内核发布记录

本文将详细介绍PolarDB MySQL引擎内核模块的更新日志和全部的功能特性。

PolarDB数据库引擎在PolarDB架构中的位置如下图所示。



更多关于PolarDB产品架构及特点，请参见产品架构。

查询版本号

您可以通过如下方式查看集群的内核版本信息：

- 登录PolarDB控制台，在目标集群的**基本信息**页直接查看**内核版本**信息。

- 若在控制台上未看见**内核版本**信息，您可以通过 `show variables like "%polardb_version%";` 或 `show variables like '%rds_release_date%';` 命令查询具体的版本号。

> **说明**
> - PolarDB MySQL引擎5.6版本集群仅支持通过 `show variables like '%rds_release_date%';` 命令查询具体的版本号。
> - 关于PolarDB数据库引擎版本号的详细说明，请参见数据库引擎版本号说明。

PolarDB MySQL与社区MySQL版本的兼容关系

内核版本	完全兼容的MySQL小版本
8.0.2	8.0.18
8.0.1	8.0.13
5.7	5.7.28
5.6	5.6.16

PolarDB MySQL引擎8.0版本发布日志

PolarDB MySQL引擎8.0支持如下版本：

- 8.0.2.1.4.1（发布日期：20211026）

类别	说明
问题修复	支持修改数据库缓存大小。

- 8.0.2.1.4（发布日期：20201201）

类别	说明
新增功能和性能优化	支持官方MySQL 8.0.20的Index Hints功能。

类别	说明
问题修复	○ 修复Hash Join并行的一个稳定性问题。 ○ 修复统计信息中关于每个索引key对应行数估算错误的问题。 ○ 修复并行扫描共享临时表时，处理空数据出现错误的问题。 ○ 修复官方REGEXP_REPLACE函数处理数据出现错误的问题。 ○ 修复官方对于复杂查询中包含子查询常量的崩溃问题。 ○ 修复filesort返回错误数据的问题。

- 8.0.2.1.3（发布日期：20201026）

类别	说明
新增功能和性能优化	○ 增强Hash Join的代价估算模型性能。 ○ 支持并行Hash Join，即Build和Probe阶段的并行。
问题修复	○ 修复并行执行计划中并行分区数量错误显示的问题。 ○ 修复并行子查询的一个异常崩溃问题。 ○ 修复在并行查询中使用RAND() 函数时，多个工作线程不能产生随机结果的问题。

- 8.0.2.1.2（发布日期：20200927）

类别	说明
新增功能和性能优化	PolarDB并行全面支持最新的火山模型迭代执行器。

- 8.0.2.1.1（发布日期：20200826）

类别	说明
新增功能和性能优化	优化集群的写性能。
问题修复	修复内存泄漏问题。

- 8.0.2.1.0（发布日期：20200722）

类别	说明
新增功能和性能优化	○ EXPLAIN支持并行计划的展示，支持通过FORMAT=TREE查看并行的执行计划树。 ○ 并行查询支持哈希连接，增强了无索引的JOIN性能，详情请参见Hash Join的并行执行。 ○ 支持Resource Manager功能，能够实现对CPU和内存资源的监控，详情请参见Resource Manager。 ○ 支持Performance Agent，能够实现PolarDB MySQL引擎集群中的节点内部各项性能数据的采集与统计，您可以通过直接查询内存表PERF_STATISTICS获取相关指标的性能数据，详情请参见Performance Agent。 ○ 支持热重启（Warm buffer pool），在Crash重启和升级时，Buffer Pool中的数据依然存在，大幅度加快重启速度，并保持重启后性能无衰减。

- 8.0.1.1.23（发布日期：20220120）

类别	说明
新增功能和性能优化	○ 支持Strict Consistency Cluster（RO节点强一致性读）功能。 ○ Statement Outline支持PrepareStatement。 ○ 支持热备节点功能，进一步优化高可用效率。
问题修复	○ 修复Fast Query Cache在RO节点获取MDL锁导致Redo日志同步阻塞的问题。 ○ 支持SELECT FOR UPDATE/SHARE WAIT N语法。 其中，N表示等待超时的秒数，针对单个行锁。如果一个查询需要锁定多行，不会对时间进行累计，仅对单行进行超时检测。等待时间超过N秒，则返回锁等待超时错误 `Lock wait timeout exceeded; try restarting transaction` 。

- 8.0.1.1.22（发布日期：20211222）

类别	说明
问题修复	○ 优化并行计划在分区表索引等值访问时的并行度估算精度。 ○ 修复特定场景下，优化器进行统计信息估算的代价耗时长的问题。 ○ 修复优化器针对部分GROUP BY语句未选择更优的索引范围路径的问题。 ○ 修复Standby节点通过HA切换为主节点后，创建新的Redo日志文件失败的问题。

- 8.0.1.1.21（发布日期：20211126）

类别	说明

类别	说明
问题修复	○ 修复group_concat_max_len参数取值较大时，出现算术溢出导致GROUP_CONCAT函数结果错误的问题。 ○ 修复备可用区的Standby节点，从故障中恢复后出现数据错误的问题。 ○ 修复备可用区的Standby节点切换为主节点（RW节点）时，出现的数据异常问题。 ○ 修复优化器在选定了多列组合索引过滤条件时，只能进行单列索引的过滤，不能进行组合索引过滤的问题。

- 8.0.1.1.19（发布日期：20210918）

类别	说明
新增功能和性能优化	○ 针对DDL操作增加了新的并发控制CCL规则。 ○ 增加参数restrict_on_limit_without_order，控制存在limit子句的并行查询中在没有order by子句的情况下，是否允许执行并行查询。
问题修复	○ 修复并行查询中的Group by隐式排序，在选择group by列的索引的时候，并发执行结果无序的问题。 ○ 修复在使用线程池（Thread Pool）时，审核日志中事务ID字段始终为0的错误。

- 8.0.1.1.18（发布日期：20210814）

类别	说明
新增功能和性能优化	数据库内核支持事务断点续传和只读节点读取未提交的事务。
问题修复	优化 `master_key_id_mutex` ，使得DDL操作可以并行执行。

- 8.0.1.1.17（发布日期：20210723）

类别	说明

类别	说明
新增功能和性能优化	○ 新增polar_replica_work_on_nonblock_mdl_mode参数。开启该参数时，只读节点上的RU/RC隔离级别的未提交事务将不再阻塞主节点上的DDL操作，同时只读节点上将不在保证表定义的事务特性。 ○ 优化了海量表场景下（如需要提供SaaS服务的场景）对统计信息的维护机制，大幅提升高并发下的查询表结构（desc table）和读写性能。
问题修复	○ 修复了只读节点在进行物理复制，高并发压力很大时，崩溃在m_recv_bits.is_set(fold)的问题。 ○ 修复了只读节点在高并发压力很大时，replay log崩溃的问题。

- 8.0.1.1.16（发布日期：20210624）

类别	说明
问题修复	在ACL Cache Lock请求发生等待的时候，在master error log中打印请求线程和最早的锁持有线程的信息来辅助诊断问题。

- 8.0.1.1.15（发布日期：20210525）

类别	说明
新增和优化功能	○ TDE支持对集群中MySQL的新建表自动加密。 ○ MySQL数据表支持utf8mb4_0900_bin字符集。
问题修复	○ 修复了instant add column后回滚Update操作时产生的记录过长，进而导致数据库崩溃的问题。 ○ 修复了避免mysql.slow_log、mysql.general_log被DDL语句强制使用innodb引擎造成混乱的问题。 ○ 修复了REGEXP函数的元数据信息错误导致结果集错误的问题。 ○ 修复了在虚拟列上回滚Update操作导致数据库崩溃的问题。 ○ 调整RO节点在首次注册到主节点时是否要立即触发checkpoint的策略。当LSN差值小于规定的阈值时，可不触发checkpoint。

- 8.0.1.1.14（发布日期：20210423）

类别	说明

类别	说明
新增功能和性能优化	○ 优化了海量表场景（如需要提供SaaS服务的场景）下对内部索引信息的维护，来提升只读节点的启动速度。 ○ 优化GDN同步连接，减少同步线程对CPU资源的消耗，提升了小规格（即8核以下）从集群同步Redo日志的速度。 ○ 优化了并行度控制参数中的AutoDop策略，以避免对PARALLEL HINT和 `force_parallel_mode` 的使用造成影响。
问题修复	○ 修复当RANGE查询范围较广时，由于 `records_in_range` 统计信息不准确，导致使用了错误索引的问题。 ○ 修复当进行按时间点恢复全量数据时，由于缓冲池过小导致Standby节点崩溃问题。 ○ 修复X-Engine引擎内部元信息内存占用过高的问题。 ○ 修复官方MySQL中2个关于ACL DDL的问题，避免由于ACL DDL操作带来的死锁导致集群不可用的问题。 ○ 修复当并行查询中包含了SQL_BUFFER_RESULT关键字，且使用了聚集函数但没有GROUP BY时，会返回错误结果集的问题。

- 8.0.1.1.13.2（发布日期：20210419）

类别	说明
问题修复	修复主备切换时，TDE加密表中的加密信息可能丢失导致解密失败的问题。

- 8.0.1.1.13.1（发布日期：20210408）

类别	说明
问题修复	修复并行查询中，由于未将Block Nested-Loop Join（BNL）算法中的常量过滤条件推到单表上，导致查询速度变慢的问题。

- 8.0.1.1.13（发布日期：20210330）

类别	说明
新增功能和性能优化	○ 只读节点支持polar_use_statement_mdl_on_replica参数，避免只读节点上的事务（RC隔离级别）堵塞主节点的DDL操作。同时，当只读节点上的读事务与主节点上的DDL并发时，读事务将会看到不同的表定义（例如在只读节点读事务的两条语句中间，主节点上存在ADD COLUMN操作，那么只读节点上的第二条语句将读到比第一条语句更多的列）。 ○ 移除索引等值查找路径中不必要的等值条件，便于在ORDER BY LIMIT场景下将Limit offset下推执行。 ○ 新增 `dbms_stats.gather_stats(timeout, workers)` 命令，支持通过事件调度或手动执行该命令来更新过时的直方图。 ○ `mysql.slow_log` 新增支持查看 `log_version` 、 `log_id` 、 `origin_start_time` 和 `rds_ext` 字段。

类别	说明
问题修复	○ 修复当对历史库的X-Engine表执行CHECK TABLE、COUNT(*)或DDL等命令时，无法终止查询的问题。 ○ 将KICKOUT修改为非保留关键字。 ○ 修复特定情况下，由于生成查询计划时评估扫描行数少于实际扫描行数，导致未充分进行并行查询的问题。

- 8.0.1.1.12.2（发布日期：20210312）

类别	说明
问题修复	○ 修复在打开 `session_track_temporary_tables` 系统变量的情况下，在存储过程中创建或删除临时表会导致集群不可用的问题。 ○ 引入官方MySQL 8.0.14的补丁，用于解决执行CREATE USER命令时，由于无法获取MySQL数据库系统表的元数据锁（MDL），导致CREATE USER语句被阻塞的问题。

- 8.0.1.1.12.1（发布日期：20210302）

类别	说明
新增功能和性能优化	优化在多表场景下持续导入数据时，PolarDB历史库X-Engine的写性能。
问题修复	修复并行查询中，由于Leader线程缺失互斥锁（Mutex）保护，导致与Worker线程上的元数据锁（MDL）状态可能不一致的问题。

- 8.0.1.1.12（发布日期：20210220）

类别	说明
新增功能和性能优化	○ 并行度控制策略新增auto_dop_low_degree_cost参数，用于设置并行查询的并行度选择策略，详情请参见并行度控制策略。 ○ 新增支持通过 `restore_table` 命令，快速恢复回收站内的表，详情请参见表回收站。 ○ 支持从只读节点上获取Binlog，详情请参见远程获取并解析PolarDB MySQL引擎Binlog日志。 ○ 支持在 `opt trace` 中打印 `in_memory` 等关键信息，便于在执行计划出现问题时，定位问题原因。
问题修复	○ 引入Port Percona补丁，用于解决高并发场景下，ACL CACHE 元数据锁冲突检测较慢的问题。 ○ 在 `optimizer_switch` 系统变量中新增 `preferred_ordering_index` 参数，用于修复某些场景下（如使用了LIMIT子句的ORDER BY或GROUP BY查询），由于使用了有序索引，导致没有选择最优计划的问题。 ○ 修复部分场景下SHOW PROCESSLIST结果不正确的问题。 ○ 修复8.0.1.1.10之前的版本在执行小版本升级后，由于未更新系统表 `information_schame.KEY_COLUMN_USAGE` 的定义，导致系统表访问较慢的问题。

- 8.0.1.1.11（发布日期：20210129）

类别	说明
新增功能和性能优化	○ 新增支持通过parallel_degree_policy参数来设置并行查询中并行度的配置策略，详情请参见并行度控制策略。 ○ 新增支持通过SET GLOBAL语句设置max_digest_length参数值来限制可识别语句的长度。 **说明** max_digest_length参数值修改后，客户端需要重新连接集群，新参数值才会生效。
问题修复	○ 修复主节点和只读节点权限不一致的问题。 ○ 修复主备切换后，只读节点无法连接到主节点的问题。 ○ 修复当特定执行计划失效时，SPM的处理逻辑不正确的问题。

- 8.0.1.1.10（发布日期：20210112）

类别	说明
新增功能和性能优化	○ 新增支持 `Group By` 的隐式排序功能，与在PolarDB MySQL引擎5.7版本上的用法一致。 ○ 新增当存在Blob字段时禁用并行查询的功能。 ○ 新增只读节点上自动更新语句级并发控制缓存信息的功能。 ○ 新增热点行优化功能，详情请参见热点行优化。 ○ 新增支持DDL物理复制优化，详情请参见DDL物理复制优化。 ○ 新增支持并行元数据锁同步，详情请参见并行元数据锁同步。 ○ 新增当计算下推时快速反向遍历的功能。 ○ 优化了文件系统，加快多表场景下表的打开速度。 ○ 缩短了多表场景下的主备切换时间，加速新主节点的恢复。

类别	说明
问题修复	○ 修复当只读节点升级为主节点时出现的系统表丢失的问题。 ○ 修复开启并行查询后，使用范围查询时会导致评估扫描行数过多的问题。 ○ 修复当字段类型为BIT时，聚合查询的结果为整型的问题。 ○ 修复使用枚举字段后，通过SELECT DISTINCT查询返回的结果不正确的问题。 ○ 修复使用EXISTS条件的并行查询结果出现异常的问题。 ○ 修复某些情况下只读节点重启失败的问题。 ○ 修复当只读节点执行DDL时，因外键关联表在打开表时把正在执行的DDL表也重新打开，导致数据字典中表信息异常的问题。 ○ 修复因主备切换后未正确设置节点重启标志，导致全文索引查询失败的问题。 ○ 修复因元数据锁（MDL）导致只读节点上日志应用线程被阻塞的问题。 ○ 修复因释放的内存被复用导致主备切换后，新的主节点不可用的问题。 ○ 修复因 `polar.info` 数据问题导致所有节点不可用的问题。 ○ 修复分区表自增列异常的问题。 ○ 修复主节点出现Redo log被覆盖写导致数据出错的问题。 ○ 修复当主节点等待元数据锁（MDL）时，主节点不可用的问题。 ○ 修复在使用透明数据加密（TDE）时的相关问题。 ○ 修复在执行Lock Table并开启表回收站功能时出现集群不可用的问题。 ○ 修复主节点在执行DDL时出现死锁的问题。 ○ 修复线程池和连接控制无法同时生效的问题。

- 8.0.1.1.9（发布日期：20201218）

类别	说明
新增功能和性能优化	取消将 `SPM` 和 `PLAN` 设置为关键字，避免出现因表名包含这两个词而无法操作的问题。

- 8.0.1.1.8（发布日期：20201209）

类别	说明
新增功能和性能优化	○ 执行计划管理器新增多计划模式。 ○ 新增系统变量rds_ap_threshold参数，来阻拦优化器评估扫描记录数太多的请求。 ○ 提升了主节点的脏页落盘效率。 ○ 新增Redo多分片写入机制。
问题修复	○ 修复并行查询执行过程中出现元数据锁（MDL） key空指针的问题。 ○ 修复当创建并线线程缓存时出现查询失败的问题。 ○ 修复并行查询MRR（Multi-Range Read）返回结果异常的问题。

- 8.0.1.1.7（发布日期：20201116）

类别	说明
新增功能和性能优化	○ 提升了JOIN查询等场景中，被驱动表并行扫描的效率。 ○ 新增支持在关闭Binglog情况下，依然能够清理残留的Binglog文件。 ○ 新增Slave节点物理复制断开自动检查重连机制，避免出现长时间的物理复制断开。 ○ 提升了主节点和只读节点间的切换效率。 ○ 支持快速启动包含大量表的集群，方便快速扫描表数据文件。
问题修复	○ 修复在获取 `trx->wait_lock` 的类型时出现集群崩溃的问题。 ○ 修复在打开多队列Simulated AIO时，AIO线程存在数量上限的问题。 ○ 修复当查询索引遇到初始化查询失败时，无法直接结束查询的问题。 ○ 修复Slave节点在SMO（Split Merge Operation）过程中，当前游标的Next Page指向了一个不存在的Page的问题。 ○ 修复只读节点会读取到已被主节点覆盖的日志信息的问题。 ○ 修复因Redo日志中时间戳间隔过大导致清理Redo文件失败的问题。 ○ 修复释放元数据锁（MDL）时未清除相关缓存中表缓存信息的问题。

- 8.0.1.1.6（发布日期：20200921）

类别	说明
新增功能和性能优化	○ 提升了SPM（SQL plan manager）和并行查询的兼容性。 ○ 提高了并行查询归并排序的效率。 ○ 支持删除操作的计算下推工作。 ○ 支持PolarDB Commit Timestamp（CTS）功能。
问题修复	○ 修复 `pq_optimize_switch` 描述不正确的问题。 ○ 修复子查询不能被稳定执行的问题。

- 8.0.1.1.5（发布日期：20200819）

类别	说明
新增功能和性能优化	○ 当只读节点和主节点建立复制关系时，新增主节点是否需要立即执行checkpoint策略的功能。 ○ 支持简单的范围查询和计算下推工作。 ○ PFS（Polar file system）文件系统新增pfs_remount功能，避免了因文件未关闭导致无法挂载PFS文件的问题。 ○ 解决了只读节点上由于Parse线程强制停顿造成的性能瓶颈问题，提升了物理复制过程中数据同步的效率。 ○ 优化了多连接场景下的Early Lock Release性能，多连接场景下的集群性能提升至原来的10倍。

类别	说明
问题修复	○ 修复只读节点在连接主节点失败后的不可用问题。 ○ 修复在使用全文索引并执行DDL查询语句后，会导致的只读节点在主备切换后不可用问题。 ○ 修复执行UNDO TRUNCATE命令后导致的无法进行purge binlog问题。 ○ 修复只读节点和主节点间统计信息不一致的问题。

- 8.0.1.1.4（发布日期：20200704）

类别	说明
新增功能和性能优化	○ 支持并行DDL，提升DDL执行效率。 ○ 支持Simulated AIO动态调整多队列的长度。 ○ 支持全文搜索（Full Text Search，FTS）缓存一致性。 ○ WHERE条件中支持含有聚集函数的子查询，且若子查询支持基于索引的扫描，那么该子查询还可以支持并行执行。 ○ 临时表和普通表一样支持进行lock mode检查。
问题修复	○ 修复当主节点降为备节点时，因部分DDL仍在复制中而导致的集群不可用问题。 ○ 修复因为开启线程池导致的性能下降问题。 ○ 修复purge binlog导致死锁的问题。 ○ 修复若干内存泄漏的问题。 ○ 修复若干在高可用场景中出现的问题。

- 8.0.1.1.3（发布日期：20200529）

类别	说明
新增功能和性能优化	○ 增强安全性（如密码管理）。 ○ 提升如下场景中并行查询（Parallel Query）的性能： ■ 增强GROUP BY、UNION、SELECT COUNT(*) FROM <table>场景中的并行查询性能。 ■ 并行子查询中，执行计划使用了共享InnoDB临时表的场景。 ■ 计划中使用VIEW/DERIVED TEMP TABLE的场景。 ■ 并行查询支持定义临时表的场景， 但有如下限制： ■ 不支持临时表上不带条件的SELECT COUNT(*) ■ 不支持在Memory Engine临时表的并行。 ○ 支持新版本的审计日志格式，增加了VIP信息。 ○ 支持索引页面空闲比率控制，减少SMO概率和latch竞争，提升写入性能。 ○ 支持多队列的模拟AIO，增强刷脏和写入性能。 ○ 支持在core文件中不记录buffer pool的内容，降低core文件的大小，避免影响线上服务。

类别	说明
问题修复	○ 修复当达到TempTable最大内存限制的时候，原来逻辑中TempTable存储引擎会误报out-of-memory的错误，而不是回退到基于磁盘存储上的问题。 ○ 修复当排序缓存（sort buffer size）参数设置过小时，在InnoDB全文搜索中使用ORDER BY会导致集群不可用的问题。 ○ 修复在临时表出现同名列情况下，无法找到对应的正确Field的问题。 ○ 修复并行查询中，当使用MAX/MIN函数结合GROUP BY并使用松散扫描数据时，无法被Kill的问题。 ○ 修复故障切换时的若干问题。 ○ 修复并行查询下的若干问题。 ○ 修复使用SHOW BINARY LOGS命令可能会阻碍事务提交的问题。

- 8.0.1.1.2（发布日期：20200409）

类别	说明
新增功能和性能优化	○ 支持基于字符串前缀的排序优化方法（即优先使用字符串前缀进行排序，如果前缀相同，再使用字符串的全长进行排序）。对长字符类型的列进行排序时，可以通过指定该列值最大不同的前缀长度来加速比较，减少排序时间。 ○ 新增在如下场景中支持并行查询的能力： ■ 支持Range Cost的估算模型的并行。 ■ 支持Temporary Table表的并行。 ■ 支持Semijoin物化Lookup和Scan的策略下的并行。 ○ 增加了3类可以被PolarDB智能路由用来支持连接保持功能的会话状态tracker；同时，tracker打开后可以跟踪会话中user variable的改变、临时表的创建和删除、SQL语句中的prepare和dealloction操作等。 ○ 优化DDL过程中Drop AHI的性能，降低DDL对集群性能的影响。 ○ 增加表回收站的功能，避免出现因误删导致数据丢失的情况。 ○ 优化在大缓冲池临时表空间truncate的表现，降低临时表操作对集群性能的影响。
问题修复	○ 修复当聚合函数存在于IF函数中的场景下执行ROLLUP的问题。 ○ 修复Blob类型在排序过程中的问题。 ○ 修复并行中使用Prepare中包含聚合函数SQL的特定问题。 ○ 修复并行查询下的若干问题。 ○ 修复可能会清除过多Redo日志的问题。 ○ 修复RO节点上Redo日志的相关问题。

- 8.0.1.1.1（发布日期：20200328）

类别	说明

类别	说明
新增功能和性能优化	○ 支持在子查询含有ROLLUP的场景中使用并行。 ○ 支持语句并发控制功能。 ○ 增加 `POLARDB_INDEX` 的Hint。 ○ 优化主节点和只读节点同步延迟。 ○ 支持线程池（Thread Pool）。 ○ 支持TDE keyring_rds插件。 ○ 支持全球数据库（GDN）。 ○ 优化无锁事务系统， 优化读写性能。
问题修复	○ 修复并行查询下的若干问题。 ○ 修复ONLINE DDL过程中统计信息可能为0的问题。 ○ 优化用户态文件系统，加速集群启动。 ○ 修复innodb_flush_method被设置为all_o_direct时可能会导致集群不可用的问题。 ○ 修复事务提交放锁时可能会导致集群不可用的问题。 ○ 修复慢日志truncate时可能会阻塞用户请求的问题。 ○ 修复压缩页在RO上可能会导致集群不可用的问题。 ○ 修复线程池可能会错误断开杀复制连接的问题。

- 8.0.1.1.0（发布日期：20200205）

类别	说明
新增功能和性能优化	○ 增强并行查询的能力，支持企业级分析ROLLUP函数的并行计算。 ○ 增加优化器的估算模型能力（如增强条件过滤的选择率和对并行查询的代价估算模型），被执行的SQL可以根据Selectivity更准确地选择并行计划还是串行计划。 ○ 支持对按照FIFO模式分配工作线程的并行工作线程进行统一管理，避免大量并行查询造成系统资源耗尽的问题。
问题修复	○ 修复并行查询的内存相关系列问题。 ○ 修复并行查询下的若干不稳定问题。

- 8.0.1.0.6（发布日期：20200101）

类别	说明
问题修复	○ 修复在主节点降为备节点时Binlog Index文件未关闭的问题。 ○ 修复RO节点在访问已经被清除的Undo页时会出现集群不可用的问题。 ○ 修复在RO节点进行主备切换时，后台线程访问到不存在表空间页面的问题。 ○ 修复在集群关闭时由于日志线程已经退出之后还在写Redo日志导致的集群不可用问题。

- 8.0.1.0.5（发布日期：20191203）

类别	说明
新增功能和性能优化	○ Optimizer trace中支持并行查询的相关信息（例如您可以通过Optimizer trace的工具分析为何使用并行或者未使用并行的原因）。 ○ 增加并行查询相关的Hint，支持通过SQL Hint的方式显性启用并行和指定并行度。 ○ 支持在READ COMMITTED下INSERT..SELECT的并行扫描，您可以使用INSERT..SELECT语句将导入的数据加入到另一个表。
问题修复	○ 修复并行查询下的若干问题。 ○ 修复在主备切换时，备节点升级为主节点时出现的节点不可用的问题。 ○ 修复在主备切换时因使用部分DDL语句引起故障的问题。 ○ 修复锁限制导致报错too many connection error的问题。

PolarDB MySQL引擎5.7版本发布日志

PolarDB MySQL引擎 5.7支持如下版本：

- 5.7.1.0.17（发布日期：20220114）

类别	说明
新增功能和性能优化	○ 支持热点行优化功能。具体请参见热点行优化。 ○ 支持热备节点功能，进一步优化高可用效率。

- 5.7.1.0.16（发布日期：20211214）

类别	说明
问题修复	优化了手动触发checkpoint的策略。

- 5.7.1.0.15（发布日期：20211117）

类别	说明
新增功能和性能优化	新增Fast Query Cache功能。具体请参见Fast Query Cache。

- 5.7.1.0.14（发布日期：20211019）

类别	说明

类别	说明
问题修复	○ 提供一个optimizer hint使得用户可以控制是否强制视图进行merge。具体请参考MySQL官方文档。 ○ 支持 `SELECT FOR UPDATE WAIT N` 语法。 其中N表示等待超时的秒数，仅针对单个行锁。如果一个查询需要锁定多行时，系统不会对多个行进行时间累计，仅对单行进行超时检测。等待时间超过N秒，系统则返回锁等待超时错误： `Lock wait timeout exceeded; try restarting transaction` 。

- 5.7.1.0.13（发布日期：20210910）

类别	说明
新增功能和性能优化	GDN从集群中的只读节点支持通过alter polar to slave命令切换成主节点，从而实现从集群的高可用。

- 5.7.1.0.12（发布日期：20210827）

类别	说明
新增功能和性能优化	○ GDN中的主集群与同步节点之间支持多连接，提高物理复制的吞吐量。 ○ 支持从只读节点上获取Binlog，详情请参见远程获取并解析PolarDB MySQL引擎Binlog日志。
问题修复	○ 修复Statement Concurrency Control指定库表的规则无法匹配的问题。 ○ 加快只读节点和从集群应用redo log，提升主节点的同步效率。

- 5.7.1.0.11（发布日期：20210708）

类别	说明
新增功能和性能优化	新增polar_replica_work_on_nonblock_mdl_mode参数。开启该参数时，只读节点上的RU/RC隔离级别的未提交事务将不再阻塞主节点上的DDL操作，同时只读节点上将不在保证表定义的事务特性。

- 5.7.1.0.10（发布日期：20210615）

类别	说明
新增功能和性能优化	兼容MySQL 8.0，支持SELECT FOR UPDATE/SHARE SKIP LOCKED/NOWAIT语法。具体请参见MySQL 8.0。
问题修复	RO节点支持将innodb temp table数据落盘。

- 5.7.1.0.9（发布日期：20210513）

类别	说明
问题修复	○ 多表场景下，支持存储引擎快速启动。 ○ 修复了在虚拟列上回滚Update操作导致数据库崩溃的问题。

- 5.7.1.0.8（发布日期：20210419）

类别	说明
新增功能和性能优化	当只读节点和主节点建立复制关系时，新增主节点是否需要立即执行checkpoint策略的功能。
问题修复	○ 将KICKOUT修改为非保留关键字。 ○ 修复主备切换时，TDE加密表中的加密信息可能丢失导致解密失败的问题。

- 5.7.1.0.7（发布日期：20210310）

类别	说明
新增功能和性能优化	○ 新增并行DDL功能，提升DDL性能，详情请参见并行DDL。 ○ 新增innodb_buffer_pool_in_core_file参数，以便将buffer pool从core file中移除。 ○ Index Hints新增支持如下关键字，以便优化器处理查询时使用或忽略指定的索引： ■ `GROUP_INDEX` 或 `NO_GROUP_INDEX` ：使用或忽略指定的索引以进行带有GROUP BY操作的索引扫描。 ■ `INDEX` 或 `NO_INDEX` ：强制服务器使用或忽略指定索引。 ■ `JOIN_INDEX` 或 `NO_JOIN_INDEX` ：强制MySQL对任何访问方法使用或忽略指定的索引。 ○ Join Order Hint新增支持如下关键字，以便优化器选择合适的表连接顺序： ■ `JOIN_FIXED_ORDER` ：强制优化器使用FROM子句中出现的顺序来连接表。 ■ `JOIN_ORDER` ：指导优化器使用指定的表顺序连接表。 该Hint适用于命名表。 优化器可以将未命名的表放在连接顺序中的任何位置，包括指定表之间。 ■ `JOIN_PREFIX` ：指导优化器为连接执行计划的前几个表使用指定的表顺序来连接表。 该Hint适用于命名表。 优化器会将所有的其他表放在命名表之后。 ■ `JOIN_SUFFIX` ：指导优化器为连接执行计划的最后几张表使用指定的表顺序来连接表。 该Hint适用于命名表。 优化器会将所有的其他表放在命名表之前。

类别	说明
问题修复	○ 将token number的对照表进行分区并预留部分token，以修复插入新token后，同一语句digest hash值不一致的问题。 ○ 修复变配过程中，由于只读节点无法在主节点上成功注册，导致变配失败的问题。 ○ 修复在打开 `session_track_temporary_tables` 系统变量的情况下，在存储过程中创建或删除临时表会导致集群异常退出的问题。

- 5.7.1.0.6.3（发布日期：20210222）

类别	说明
问题修复	修复部分场景下SHOW PROCESSLIST结果不正确的问题。

- 5.7.1.0.6.2（发布日期：20210210）

类别	说明
问题修复	修复在只读节点上执行使用了Index Merge优化的查询时，偶发性地出现1032错误码 `Can't find record in TABLE` 的问题。

- 5.7.1.0.6.1（发布日期：20210202）

类别	说明
问题修复	○ 修复读写节点上刷脏异常的问题。 ○ 修复在已执行过主备切换的集群上，可能无法再进行更换主可用区操作的问题。 ○ 修复由于执行SHOW PROCESSLIST时错误调用了THD（Thread Descriptor），导致的数据库崩溃问题。

- 5.7.1.0.6（发布日期：20210129）

类别	说明
新增功能和性能优化	○ 新增支持Statement Queue功能，详情请参见Statement Queue。 ○ 新增支持秒级加字段（ `Instant add column` ）功能，详情请参见秒级加字段。 ○ 新增支持Returning功能，详情请参见Returning。
问题修复	修复升级PolarDB内核版本时，系统表丢失的问题。

- 5.7.1.0.5（发布日期：20201231）

类别	说明

类别	说明
问题修复	○ 修复主节点修改密钥后，在只读节点上进行查询时，只读节点会崩溃的问题。 ○ 修复在只读节点上分析分区表时，只读节点会崩溃的问题。 ○ 修复主备切换后，由于表空间不一致导致新的主节点会崩溃的问题。

- 5.7.1.0.4（发布日期：20201117）

类别	说明
问题修复	○ 修复主备切换后，无法将数据插入临时表的问题。 ○ 修复当在只读节点上扩展临时表空间时，只读节点不可用的问题。 ○ 修复Simulated AIO不能正常工作的问题。

- 5.7.1.0.3（发布日期：20201021）

类别	说明
新增功能和性能优化	支持透明数据加密TDE功能。
问题修复	○ 修复只读节点和主节点间统计信息不一致的问题。 ○ 修复SHOW PROCESSLIST返回结果会显示错误信息的问题。

- 5.7.1.0.2（发布日期：20200828）

类别	说明
问题修复	○ 修复从RDS MySQL 5.7迁移后不能扩展.IDB文件的问题。 ○ 禁止在主备切换时执行CF_STATUS_COMMAND相关命令，以保证主备切换的正常运行。 ○ 修复因后台更新统计信息线程和Truncate逻辑出现Page争用，导致主节点不可用的问题。

- 5.7.1.0.1（发布日期：20200730）

类别	说明
问题修复	○ 修复当部分PolarDB专属命令缺少对应命令数字时，导致maxscale不能正常访问的问题。 ○ 修复通过从回收站恢复方式创建的集群由于找不到表空间，导致集群不可用的问题。 ○ 修复执行 `promote replica` 命令时，有文件未关闭导致unmount PFS文件崩溃的问题。

PolarDB MySQL引擎5.6版本发布日志

PolarDB MySQL引擎5.6支持如下版本：

- 5.6.1.0.30（发布日期：20211110）

类别	说明
问题修复	○ 修复分区表统计信息不稳定的问题。 ○ 修复报文长度为251时，数字长度编码错误的问题。

- 5.6.1.0.29（发布日期：20210909）

类别	说明
新增功能和性能优化	○ 数据库内核支持事务断点续传。 ○ 支持Fast Query Cache。具体请参见Fast Query Cache。
问题修复	加快只读节点和从集群应用redo log，提升主节点的同步效率。

- 5.6.1.0.28（发布日期：20210723）

类别	说明
新增功能和性能优化	新增polar_replica_work_on_nonblock_mdl_mode参数。开启该参数时，只读节点上的RU/RC隔离级别的未提交事务将不再阻塞主节点上的DDL操作，同时只读节点上将不在保证表定义的事务特性。
问题修复	优化表空间元信息的加载速度。对于拥有百万级以上表文件的数据库实例，能大幅缩短主节点崩溃的恢复时间以及从节点的启动时间。

- 5.6.1.0.27（发布日期：20210601）

类别	说明
问题修复	○ 优化Standby节点在执行truncate polar logs lsn命令删除文件时按照4K对齐。 ○ 将KICKOUT修改为非保留关键字。 ○ 调整只读节点在首次注册到主节点时是否要立即触发checkpoint策略。当LSN差值小于特定的阈值时，可不触发checkpoint策略。 ○ load polar logs支持添加条件语句。 ○ 修复autoinc重复的问题。

- 5.6.1.0.26（发布日期：20210319）

类别	说明

类别	说明
问题修复	○ 修复当执行FLUSH PRIVILEGES或FLUSH GRANT命令来批量授权时，可能出现连接失败的问题。 ○ 修复部分情况下由于分区表估计逻辑提前中止，导致的分区表估计错误的问题。 ○ 修复部分场景下SHOW PROCESSLIST结果不正确的问题。 ○ 修复在打开 `session_track_temporary_tables` 系统变量的情况下，在存储过程中创建或删除临时表会导致集群不可用的问题。

- 5.6.1.0.25（发布日期：20210205）

类别	说明
新增功能和性能优化	优化库表级恢复功能，提升数据恢复速度。
问题修复	○ 修复只读节点读取到已经TRUNCATE的undo page后会导致节点不可用的问题。 ○ 修复在已执行过主备切换的集群上，可能无法再进行更换主可用区操作的问题。

- 5.6.1.0.24（发布日期：20210122）

类别	说明
新增功能和性能优化	○ 优化了PolarDB引擎初始化进程，缩短大表场景下引擎的启动时间。 ○ 新增支持在**基本信息**页查看集群的**内核版本**信息。
问题修复	○ 修复从RDS迁移至PolarDB过程中，无法TRUNCATE Undo Log的问题。 ○ 修复无法新增系统表的问题。 ○ 修复库表恢复时主节点不可用的问题。 ○ 修复当查询结果为DECIMAL类型时，排序不正确的问题。 ○ 修复若干在特殊情况下可能出现的MySQL服务进程崩溃的问题。

- 5.6.1.0.23（发布日期：20210104）

类别	说明
问题修复	修复只读节点上的内存泄漏问题。

- 5.6.1.0.22（发布日期：20201225）

类别	说明
新增功能和性能优化	PFS新增支持目录索引，以提升海量表场景下的集群性能。

类别	说明
问题修复	○ 修复某些情况下，主备切换后节点角色不对导致集群不可用的问题。 ○ 修复Statement Queue未初始化导致只读节点崩溃的问题。 ○ 修复新增系统表在主备切换后没有初始化的问题。 ○ 修复线程池和连接控制（Connection Control ）功能会同时开启的问题。 ○ 修复全文索引存在重复ID导致集群不可用的问题。 ○ 修复某些情况下只读节点查询失败的问题。 ○ 修复日志复制线程退出异常造成复制中断的问题。

- 5.6.1.0.21（发布日期：20201112）

类别	说明
新增功能和性能优化	○ 支持全量恢复方式2：恢复到过去时间点功能。 ○ 支持透明数据加密TDE功能。 ○ PFS支持单表恢复所需的多盘挂载能力。 ○ 优化在物理复制中AHI（Adaptive Hash Index）锁的资源占用问题。
问题修复	○ 修复SELECT语句在使用DYNAMIC RANGE AND INDEX MERGE情况下出现OOM（Out Of Memory）的问题。 ○ 修复某些情况下创建或删除账号导致集群崩溃的问题。 ○ 修复某些情况下无法重连STANDBY节点的问题。 ○ 修复当主备切换发生异常时导致集群无法启动的问题。 ○ 修复某些情况下Binlog线程状态不正确的问题。

- 5.6.1.0.20（发布日期：20201027）

类别	说明
新增功能和性能优化	提升某些情况下物理复制的效率。
问题修复	○ 修复某些情况下执行 `CREATE TABLE... SELECT` 命令会导致集群崩溃的问题。 ○ 修复存储过程中，因派生表使用次数过多导致内存泄露的问题。 ○ 修复使用按时间点恢复数据时，恢复时间不准或恢复失败的问题。 ○ 修复PolarFS异常日志输出过多的问题。 ○ 修复关闭外键检查后，执行DDL导致表丢失的问题。 ○ 修复同时TRUNCATE多个临时表导致只读节点崩溃的问题。

- 20200831（发布日期：20200922）

类别	说明
新增功能和性能优化	PFS支持本地盘，支持挂载可写快照及性能优化。

类别	说明
问题修复	○ 修复某些情况下使用Statement Queue功能会导致集群崩溃的问题。 ○ 修复Corefiles占用过多空间的问题。 ○ 修复只读节点和主节点间统计信息不一致的问题。 ○ 修复只读节点切换为主节点后，其它只读节点无法连接新主节点的问题。 ○ 修复只读节点和主节点间全文索引缓存不一致的问题。

- 20200616（发布日期：20200701）

类别	说明
新增功能和性能优化	○ 支持Statement Queue功能，详情请参见Statement Queue。 ○ 优化RO和RW之间的复制延迟。 ○ 支持复制LOCK TABLE时的MDL锁。
问题修复	○ 修复MDL锁复制中的问题。 ○ 修复FTS INDEX创建在系统表的问题。 ○ 修复热点更新优化的一些问题。

- 20200601（发布日期：20200605）

类别	说明
新增功能和性能优化	○ 支持热点行更新优化，详情请参见热点行优化。 ○ 支持Statement Concurrency Control功能，详情请参见Statement Concurrency Control。
问题修复	修复线程池（Thread Pool）带来的写性能下降的问题。

- 20200507（发布日期：20200513）

类别	说明
新增功能和性能优化	○ 增加并发控制功能。 ○ 增加一个参数控制索引页的空闲空间。 ○ 优化Simulate AIO。
问题修复	○ 修复bool flag类导致的性能退化问题。 ○ 修复pfs_umount表没有关闭导致集群不可用的问题。

版本升级

- DB Version和Minor Version升级

PolarDB暂时不提供直接升级DB Version和Minor Version版本（如您无法将PolarDB MySQL引擎 5.6版本直接升级到PolarDB MySQL引擎 8.0版本，同时您也无法将PolarDB MySQL引擎 8.0.1版本直接升级到8.0.2版本），但您可以通过DTS将数据从源版本迁移或同步到目标版本的集群完成升级，关于如何迁移或同步数据，请参见数据迁移或同步方案概览。

- Revision Version升级

  关于如何升级Revision Version版本，请参见Revision Version升级。

4.1.3. 内核版本说明

本文将详细介绍PolarDB数据库内核版本的组成部分及各部分间的关系。

一个完整的PolarDB数据库内核版本号由大版本（DB version）号、小版本（Minor version）号和修订版本（Revision version）号三者组成，三者间的关系如下所示（以PolarDB MySQL引擎 8.0版本为例）：

大版本号是PolarDB数据库内核版本最重要的标识号，一个大版本下通常包含一个或多个小版本，如大版本5.6仅包含一个小版本5.6.16，而大版本8.0包含8.0.1和8.0.2两个小版本。不同小版本支持的功能差异较大，建议您在购买集群前先根据业务需要确定适合的小版本号。通常情况下，一个小版本下还包含多个修订版本。新版本的修订版本会优化或改进现有功能，或者增加一些新的简单功能。同时，小版本和修订版本均会包含安全、性能等方面的优化和改进。

> **说明** PolarDB数据库内核版本整体向下兼容，高版本包含低版本的全部功能，即当您从低版本升级到高版本后，应用程序不需要做任何修改，完全兼容。关于如何升级版本，详情请参见版本升级。

4.1.4. 内核高级功能

本文汇总了PolarDB支持的内核高级功能。

功能	8.0	5.7	5.6
并行度控制策略	支持（内核版本需为8.0.1.1.11或以上）	不支持	不支持
Returning	不支持	支持（内核版本需为5.7.1.0.6或以上）	不支持
秒级加字段	支持（所有版本均支持）	支持（内核版本需为5.7.1.0.6或以上）	不支持

功能		8.0	5.7	5.6
并行元数据锁同步		支持（内核版本需为8.0.1.1.10或以上）	支持（所有版本均支持）	支持（所有版本均支持）
DDL物理复制优化		支持（内核版本需为8.0.1.1.10或以上）	不支持	不支持
并行DDL		支持（内核版本需为8.0.1.1.7或以上）	不支持	不支持
Fast Query Cache		支持（内核版本需为8.0.1.1.5或以上）	不支持	不支持
Statement Outline		支持（内核版本需为8.0.1.1.1或以上）	支持（内核版本需为5.7.1.0.2或以上）	不支持
ROLLUP性能增强		支持（内核版本需为8.0.1.1.0或以上）	不支持	不支持
表回收站（Recycle Bin）		支持（内核版本需为8.0.1.1.2或以上）	不支持	不支持
透明数据加密TDE		支持（内核版本需为8.0.1.1.1或以上）	支持（内核版本需为5.7.1.0.3或以上）	支持（内核版本需为5.6.1.0.21或以上）
大查询优化（Parallel Query）	并行查询（Parallel Query）	支持（内核版本需为8.0.1.0.5或以上）	不支持	不支持
	Hash Join的并行执行	支持（内核版本需为8.0.2.1.0或以上）	不支持	不支持
	Semi-Join的并行执行	支持（内核版本需为8.0.1.1.2或以上）	不支持	不支持
	通过Hint控制	支持（内核版本需为8.0.1.0.5或以上）	不支持	不支持
	Statement Concurrency Control	支持（内核版本需为8.0.1.1.7以上）	不支持	支持（内核版本需为20200601或以上）

功能		8.0	5.7	5.6
高并发优化	Inventory Hint	支持（内核版本需为8.0.1.1.1或以上）	不支持	不支持
	Statement Queue	支持（内核版本需为8.0.1.1.1或以上）	支持（内核版本需为5.7.1.0.6或以上）	支持（内核版本需为20200616或以上）
	热点行优化（适用于秒杀业务场景）。	支持（内核版本需为8.0.1.1.10或以上）	不支持	支持（内核版本需为20200601或以上）
	线程池（Thread Pool）。	支持（内核版本需为8.0.1.1.1或以上）	不支持	支持（内核版本需为20200423以上）

4.2. DDL优化

4.2.1. 秒级加字段

使用传统方法执行加列操作时，需要重建整个表数据，占用大量系统资源。PolarDB MySQL引擎新增支持秒级加字段（`Instant add column`）功能，在加列操作时只需变更表定义信息，无需修改已有数据，帮助您快速完成对任意大小的表的加列操作。本文介绍如何使用秒级加字段功能。

前提条件

集群版本需为以下版本之一：

- PolarDB MySQL引擎5.7版本且修订版本为5.7.1.0.6或以上，您可以通过查询版本号确认集群的修订版本。

  > **说明** 您需要先配置innodb_support_instant_add_column参数才能在PolarDB MySQL引擎5.7版本的集群上使用该功能。

- PolarDB MySQL引擎8.0版本。

  > **说明** PolarDB MySQL引擎8.0版本的集群默认支持秒级加字段功能，无需配置任何参数。

使用限制

- 新增列只能为表的最后一列。
- 不支持虚拟列（PolarDB MySQL引擎8.0版本支持）。
- 不支持分区表（PolarDB MySQL引擎8.0版本支持）。
- 不支持使用了全文索引的表。
- 不支持开启了`Implicit primary key`选项且未自定义主键的表。
- 不支持在同一条SQL中同时执行其它DDL操作和`Instant add column`操作。

使用方法

- 参数说明

  针对PolarDB MySQL引擎5.7版本的集群，您需要开启innodb_support_instant_add_column参数来使用秒级加字段功能。

  > **说明** PolarDB MySQL引擎8.0版本的集群无需配置该参数即可直接使用秒级加字段功能。

参数	级别	说明
innodb_support_instant_add_column	Global	秒级加字段功能的开关，取值范围如下： ○ ON：开启秒级加字段功能 ○ OFF：关闭秒级加字段功能（默认值）

- 语句
  - 指定 `ALGORITHM=INSTANT` 以强制使用秒级加字段功能，语句示例如下：

    ```
    ALTER TABLE test.t ADD COLUMN test_column int, ALGORITHM=INSTANT;
    ```

    使用上述语句时，若返回 `ERROR 0A000: ALGORITHM=INSTANT is not supported for this operation. Try ALGORITHM=COPY/INPLACE.` 的错误，表示当前加列操作不能以Instant算法执行，建议您查看innodb_support_instant_add_column参数是否已开启，并仔细核对使用限制。
  - 不指定 `ALGORITHM` 或指定 `ALGORITHM=DEFAULT` ，PolarDB会自行选择执行速度最快的算法来执行加列操作，语句示例如下：

    ```
    ALTER TABLE test.t ADD COLUMN test_column int, ALGORITHM=DEFAULT;
    ALTER TABLE test.t ADD COLUMN test_column int;
    ```

    > **说明** PolarDB算法选择的优先级为INSTANT > INPLACE > COPY。

- 查看通过Instant算法增加的列信息

  `INFORMATION_SCHEMA` 数据库中新增了 `INNODB_SYS_INSTANT_COLUMNS` 表。该表记录了使用Instant算法增加的列信息，例如列名、列序号和默认值（二进制方式存储）等。您可通过如下语句查看该表详情来确认新增的列信息。

  ```
  SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_INSTANT_COLUMNS;
  ```

  > **说明** 对目标表使用了Instant算法增加列后，如果执行了需要重建表的DDL操作（如DROP COLUMN），系统将会删除 `INNODB_SYS_INSTANT_COLUMNS` 表中目标表的相关列信息。

相关视频

4.2.2. 并行DDL

PolarDB新增支持并行DDL的功能。当数据库硬件资源空闲时，您可以通过并行DDL功能加速DDL执行，避免阻塞后续相关的DML操作，缩短执行DDL操作的窗口期。

前提条件

PolarDB集群版本需满足如下条件之一：

- PolarDB MySQL引擎8.0版本且修订版本为8.0.1.1.7或以上。
- PolarDB MySQL引擎5.7版本且修订版本为5.7.1.0.7或以上

如何确认集群版本，详情请参见查询版本号。

注意事项

开启并行DDL功能后，由于并行线程数的增加，硬件资源（如CPU、内存、IO等）的占用也会随之增加，可能会影响同一时间内执行的其他SQL操作，因此建议在业务低峰或硬件资源充足时使用并行DDL。

使用限制

目前并行DDL加速仅支持创建二级索引（不包括聚簇索引、全文索引、空间索引和虚拟列上的二级索引）的DDL操作。

背景信息

传统的DDL操作基于单核和传统硬盘设计，导致针对大表的DDL操作耗时较久，延迟过高。以创建二级索引为例，过高延迟的DDL操作会阻塞后续依赖新索引的DML查询操作。多核处理器的发展为并行DDL使用更多线程数提供了硬件支持，而固态硬盘（Solid State Disk，简称SSD）的普及使得随机访问延迟与顺序访问延迟相近，使用并行DDL加速大表的索引创建显得尤为重要。

使用方法

- innodb_polar_parallel_ddl_threads

  您可以通过如下innodb_polar_parallel_ddl_threads参数开启并行DDL功能：

参数	级别	取值范围	说明
innodb_polar_parallel_ddl_threads	Session	[1,8] 默认值为1。	控制每一个DDL操作的并行线程数。默认值为1，即执行单线程DDL。 若该参数值不为1，当执行创建二级索引操作时将自动开启并行DDL。

- innodb_polar_use_sample_sort

  若仅开启并行DDL功能仍不能满足您的需求，您可以通过innodb_polar_use_sample_sort参数对创建索引过程中的排序进行进一步优化。

参数	级别	说明
innodb_polar_use_sample_sort	Session	sample sort优化功能开关，取值范围如下： ○ ON：开启sample sort优化功能开关 ○ OFF：关闭sample sort优化功能开关（默认值）

- `Optimization A`

  若上述功能仍不能满足您的需求，您还可以通过`Optimization A`参数对创建索引树的过程进行进一步优化。

> **说明** `Optimization A` 功能尚在内测中，暂不支持自定义开启，如需使用，请提交工单联系技术支持。

性能测试

- 测试环境
  - 一个规格为16核128 GB的标准版PolarDB MySQL引擎8.0版本的集群。
  - 集群存储空间为50 TB。
- 测试表结构

  通过如下语句创建一张名为 `t0` 的表：

```
CREATE TABLE t0(
a INT PRIMARY KEY,
b INT) ENGINE=InnoDB;
```

- 测试表数据

  通过如下语句生成测试数据：

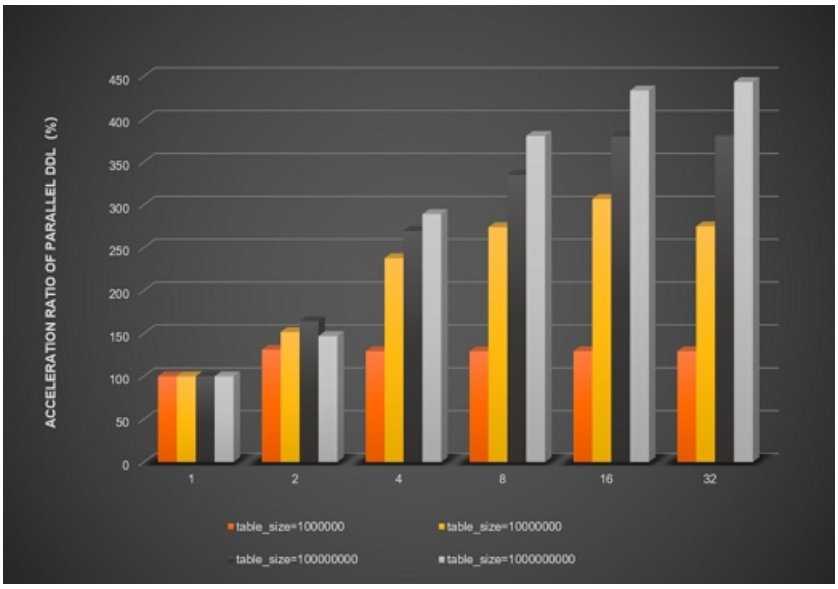
```
DELIMITER //
CREATE PROCEDURE populate_t0()
BEGIN
     DECLARE i int DEFAULT 1;
     WHILE (i <= $table_size) DO
             INSERT INTO t0 VALUES (i, 1000000 * RAND());
             SET i = i + 1;
     END WHILE;
END //
DELIMITER ;
CALL populate_t0() ;
```

> **说明**
>
> - 实际测试时请将 `$table_size` 替换成具体的表内记录数，如 `1000000` 。
> - 本测试分别使用了包含1000000行、10000000行、100000000行、1000000000行记录数的表，以及一张包含1 TB数据量的表。
> - 1 TB数据量的测试用表通过SysBench工具生成。如何使用SysBench工具，请参见测试工具。

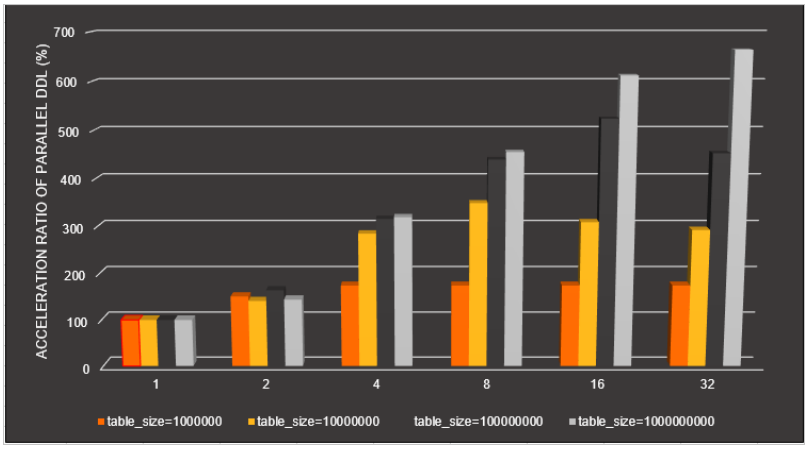
- 测试方法

  当使用不同的并行线程数（即设置innodb_polar_parallel_ddl_threads参数为1、2、4、8、16和32）时，测试在不同数据量的表中开启并行DDL后，在数据类型为 `INT` 的字段 `b` 上创建二级索引带来的DDL执行效率的提升比例。
- 测试结果

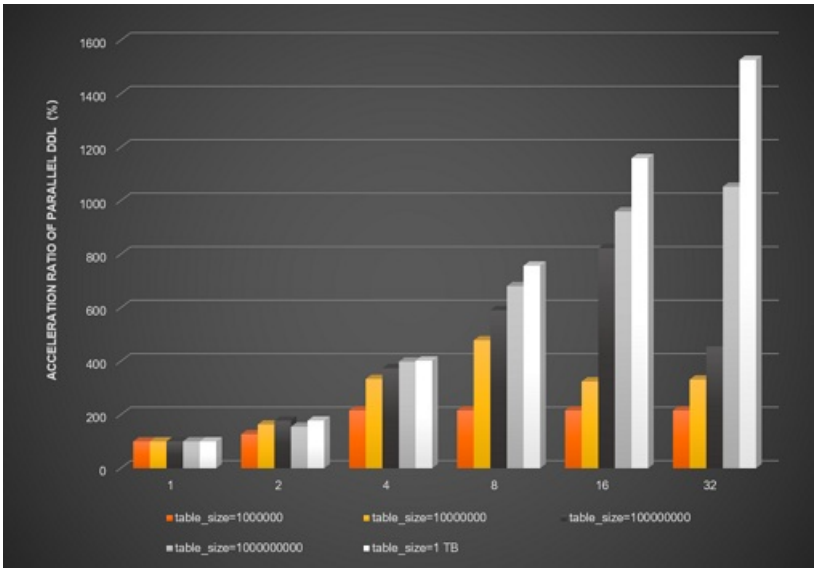
- 仅开启innodb_polar_parallel_ddl_threads参数后，并行DDL加速比结果如下图所示。



- 同时开启innodb_polar_parallel_ddl_threads和innodb_polar_use_sample_sort参数后，并行DDL加速比结果如下图所示。



- 开启innodb_polar_parallel_ddl_threads和innodb_polar_use_sample_sort参数后，同时使用 `Optimization A` 优化后，并行DDL加速比结果如下图所示。



相关视频

4.2.3. DDL物理复制优化

PolarDB通过DDL物理复制优化功能，在主节点写物理日志和只读节点应用物理日志的关键路径上进行了全面优化，大大缩短了主节点上DDL操作的执行时间和只读节点上解析DDL的物理日志复制延迟时间。本文介绍如何使用DDL物理复制优化功能。

前提条件

PolarDB集群版本需为以下版本之一：

- PolarDB MySQL引擎8.0.1版本且Revision version为8.0.1.1.10及以上
- PolarDB MySQL引擎5.7版本且Revision version为5.7.1.0.10及以上

> **说明** 您可以通过查询版本号确认集群版本。

使用限制

目前并行DDL物理复制优化仅支持创建主键或二级索引（不包括全文索引和空间索引）的DDL操作。

对于只需修改元数据的DDL操作（如rename），因其本身执行速度已经很快，无需使用该优化功能。

背景信息

PolarDB通过存储计算分离架构，实现了主节点和只读节点共享同一份存储数据，既降低了存储成本，又提高了集群的可用性和可靠性。为实现这一架构，PolarDB采用了业界领先的物理复制技术，不仅实现了共享存储架构上主节点和只读节点间的数据一致性，而且减少了Binlog fsync操作带来的I/O开销。

InnoDB中的数据是通过B-Tree来维护索引的，然而大部分Slow DDL操作（如增加主键或二级索引、optimize table等）往往需要重建或新增B-Tree索引，导致大量物理日志的产生。而针对物理日志进行的操作往往出现在DDL执行的关键路径上，增加了DDL操作的执行时间。此外，物理复制技术要求只读节点解析和应用这些新生成的物理日志，由于DDL操作而产生的大量物理日志可能严重影响只读节点的日志同步进程，甚至导致只读节点不可用等问题。

针对上述问题，PolarDB提供了DDL物理复制优化功能，在主节点写物理日志和只读节点应用物理日志的关键路径上做了全面的优化，使得主节点在执行创建主键DDL操作的执行时间最多约可减少至原来的20.6%，只读节点解析DDL的复制延迟时间最多约可减少至原来的0.4%。

使用方法

您可以通过设置如下参数开启DDL物理复制优化功能。

参数	级别	说明
innodb_bulk_load_page_grained_redo_enable	Global	DDL物理复制优化功能开关，取值范围如下： • ON：开启DDL物理复制优化 • OFF：关闭DDL物理复制优化（默认值）

性能测试

- 测试准备
  - 测试环境
    - PolarDB MySQL引擎8.0版本的集群（包含1个主节点和1个只读节点）规格为16核128 GB。
    - 集群存储空间为50 TB。
  - 测试表结构

    通过如下语句创建一张名为 `t0` 的表：

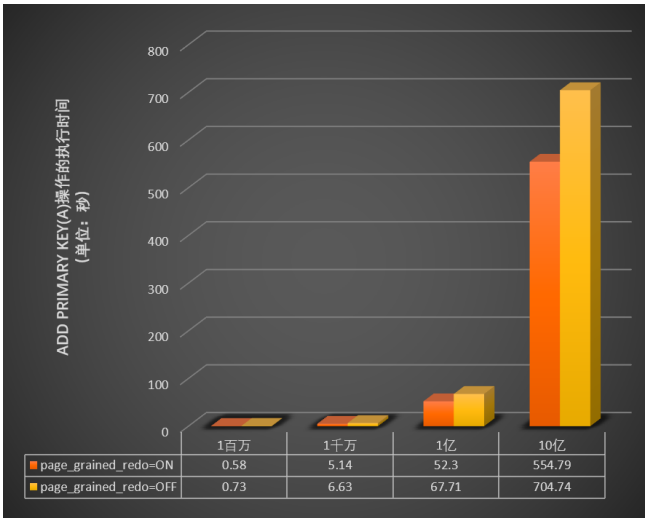
    ```
    CREATE TABLE t0(a INT PRIMARY KEY, b INT) ENGINE=InnoDB;
    ```
  - 测试表数据

    使用如下语句生成随机测试数据：

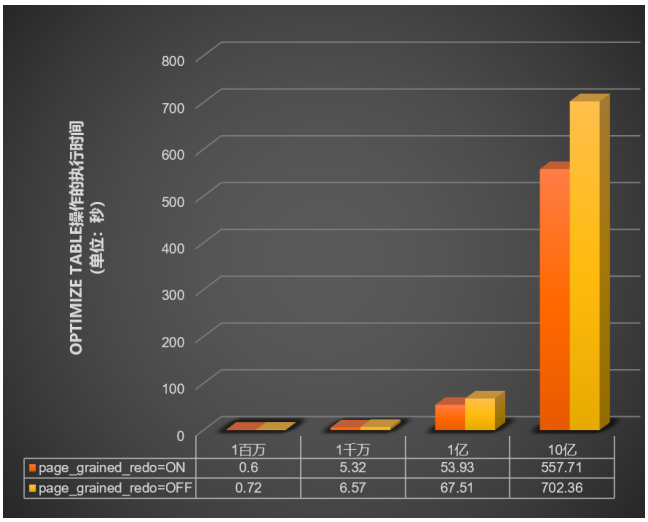
    ```
    DELIMITER //
    CREATE PROCEDURE populate_t0()
    BEGIN
         DECLARE i int DEFAULT 1;
         WHILE (i <= $table_size) DO
                 INSERT INTO t0 VALUES (i, 1000000 * RAND());
                 SET i = i + 1;
         END WHILE;
    END //
    DELIMITER ;
    CALL populate_t0();
    ```

    > **说明**
    > - 实际测试时请将 `$table_size` 替换成具体的表内记录数，如 `1000000` 。
    > - 本测试分别使用了包含1000000行、10000000行、100000000行、1000000000行记录数的表。

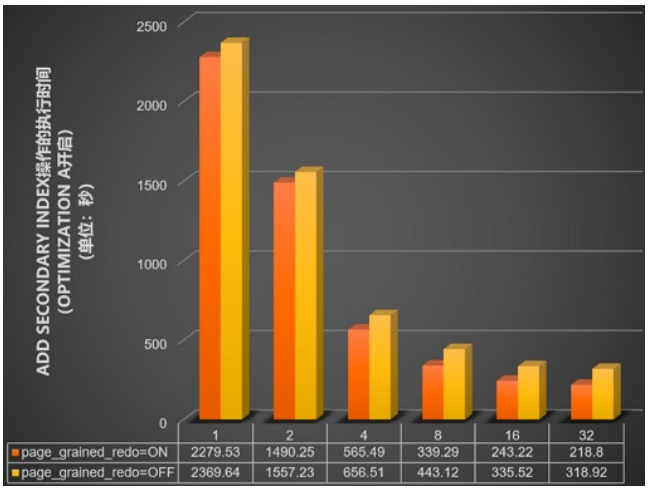
  - 测试使用的DDL操作
    - `add primary key`
    - `add secondary Index`
    - `optimize table`
- 测试方法
  - 测试一：测试当物理复制DDL优化功能开启或关闭时，在不同数据量的表中执行不同DDL操作所需的时间。
  - 测试二：测试当物理复制DDL优化功能与并行DDL配合使用时，在包含10亿数据量的表中执行`add secondary Index`操作所需的时间。
  - 测试三：测试当物理复制DDL优化功能开启或关闭时，集群（包含10亿数据量的表）中主节点的并发执行DDL操作数量不同的情况下（1、2、4、6和8），只读节点的性能（如节点状态是否正常、CPU使用率峰值和复制延迟时间等）。
- 测试结果
  - 测试一
    - 当innodb_bulk_load_page_grained_redo_enable参数开启或关闭时，测试在不同数据量（1百万、1千万、1亿和10亿）的表中执行`add primary key(a)`操作所需的时间（单位：秒），结果如下图所示。



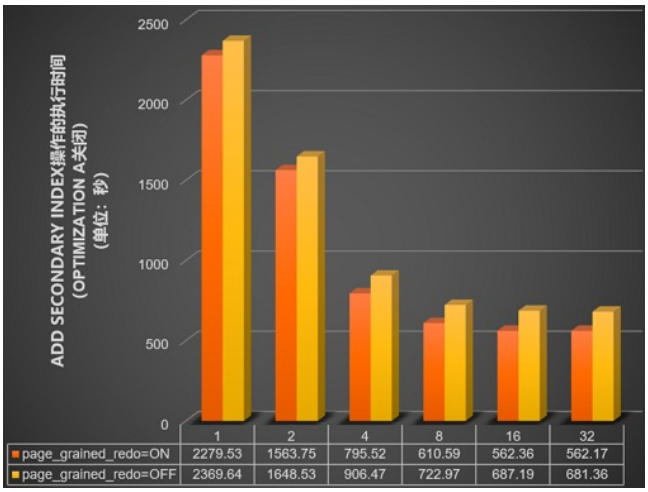
- 当innodb_bulk_load_page_grained_redo_enable参数开启或关闭时，测试在不同数据量（1百万、1千万、1亿和10亿）的表中执行 `optimize table` 操作所需的时间（单位：秒），结果如下图所示。



- 测试二
  - 并行DDL `Optimization A` 优化开启的情况下，测试innodb_bulk_load_page_grained_redo_enable参数开启或关闭后，在包含10亿数据量的表中使用不同并行线程数（即设置innodb_polar_parallel_ddl_threads参数为1、2、4、8、16和32），执行 `add secondary Index` 操作所需的时间（单位：秒），结果如下图所示。



  - 并行DDL `Optimization A` 优化关闭的情况下，测试innodb_bulk_load_page_grained_redo_enable参数开启或关闭后，在包含10亿数据量的表中使用不同并行线程数（1、2、4、8、16和32），执行 `add secondary Index` 操作所需的时间（单位：秒），结果如下图所示。



- 测试三

- innodb_bulk_load_page_grained_redo_enable参数开启的情况下，测试当集群（包含10亿数据量的表）中主节点的并发执行DDL操作数量不同（1、2、4、6和8）时只读节点的性能，结果如下表所示。

并发DDL数量	1	2	4	6	8
只读节点状态	正常	正常	正常	正常	正常
CPU使用率峰值（单位：%）	1.86	1.71	1.76	2.25	2.36
内存使用率峰值（单位：%）	10.37	10.80	10.88	11	11.1
读IOPS峰值（单位：次/每秒）	10965	10762	10305	10611	10751
复制延迟峰值（单位：秒）	0	0.73	0.87	0.93	0.03

- innodb_bulk_load_page_grained_redo_enable参数关闭的情况下，测试当集群（包含10亿数据量的表）中主节点的并发执行DDL操作数量不同（1、2、4、6和8）时只读节点的性能，结果如下表所示。

> **说明**
> - 并发DDL数为4时的数据为只读节点不可用前的测试数据。
> - 表中 `-` 符号表示在当前并发DDL数场景下未能执行相关DDL操作，因此无相应测试数据。

并发DDL数	1	2	4	6	8
只读节点状态	正常	正常	不可用	不可用	不可用
CPU使用率峰值（单位：%）	4.2	9.5	10.3	-	-
内存使用率峰值（单位：%）	22.15	23.55	68.61	-	-
读IOPS峰值（单位：次/每秒）	9243	7578	7669	-	-
复制延迟峰值（单位：秒）	0.8	14.67	211	-	-

4.2.4. 并行元数据锁同步

PolarDB支持并行元数据锁同步（Async Metadata Lock Replication）功能，用于提高DDL操作的执行效率。本文介绍如何使用并行元数据锁同步功能。

前提条件

- 对于PolarDB MySQL引擎5.6和5.7版本的集群，所有Revision version都支持该功能。

- 对于PolarDB MySQL引擎8.0版本的集群，Revision version需为8.0.1.1.10或以上，您可以通过查询版本号确认集群版本。

功能介绍

对数据库执行DDL操作时，需要在主节点和只读节点间同步元数据锁（Metadata Lock，简称MDL）信息来保证数据定义（Data Definition）的一致性。但主节点上的DDL操作常常会占有MDL，导致只读节点在长时间的等待后才能获取MDL信息进行同步，而在MDL同步成功前，只读节点不再解析Redo Log，这将严重影响DDL操作的整体效率。

通过并行MDL同步功能，PolarDB将MDL同步操作与Redo Log的解析操作解耦，实现了即使在等待MDL锁信息进行同步时，只读节点仍能继续解析并应用Redo Log。

使用方法

- 该功能默认开启，无需额外操作。
- 您可以在只读节点上使用如下SQL语句，获取目标只读节点上MDL同步相关的信息：

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOG_MDL_SLOT;
```

> **说明** 关于如何强制指定在某节点执行某查询命令，请参见Hint语法。

返回结果如下：

```
+---------+------------+-------------------+----------+-------------------+
| slot_id | slot_state | slot_name         | slot_lsn | thread_id         |
+---------+------------+-------------------+----------+-------------------+
|       0 | SLOT_NONE  | no targeted table |        0 | no running thread |
|       1 | SLOT_NONE  | no targeted table |        0 | no running thread |
|       2 | SLOT_NONE  | no targeted table |        0 | no running thread |
|       3 | SLOT_NONE  | no targeted table |        0 | no running thread |
|       4 | SLOT_NONE  | no targeted table |        0 | no running thread |
+---------+------------+-------------------+----------+-------------------+
```

该表展示了正在进行同步的MDL信息详情，其中 `slot_name` 展示了相关的数据表信息，`slot_state` 展示了当前的MDL同步状态信息，包括：

- SLOT_NONE：初始化状态。
- SLOT_RESERVED：只读节点已经接收到MDL获取请求，等待调度系统分配工作线程进行处理。
- SLOT_ACQUIRING：系统已经分配工作线程资源，只读节点正在发送MDL请求。

  > **说明** 若只读节点需要的MDL被其它连接所持有，MDL同步状态将保持在该状态。

- SLOT_LOCKED：只读节点上MDL锁已经获取并持有。
- SLOT_RELEASING：只读节点已经接收到MDL释放请求，等待调度系统分配工作线程进行处理。

- 您可在使用如下SQL语句，获取只读节点上用于同步MDL请求的工作线程状态信息：

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOG_MDL_THREAD;
```

返回结果如下：

```
+-----------+-----------+-----------------+-------------------+----------+
| thread_id | thr_state | slot_state      | slot_name         | slot_lsn |
+-----------+-----------+-----------------+-------------------+----------+
|         0 | free      | not in acquiring | no targeted table |        0 |
|         1 | free      | not in acquiring | no targeted table |        0 |
|         2 | free      | not in acquiring | no targeted table |        0 |
|         3 | free      | not in acquiring | no targeted table |        0 |
+-----------+-----------+-----------------+-------------------+----------+
```

当 `INNODB_LOG_MDL_SLOT` 表中存在状态为 `SLOT_ACQUIRING` 的MDL同步请求时， `INNODB_LOG_MDL_THREAD` 表中将会存在一个对应的工作线程用于处理该请求（即 `thr_state` 值不为 `free` ）。该情形说明只读节点上很可能存在MDL等待的情况。您可以通过 `thr_state` 的值是否为 `free` 帮助排查是否出现MDL阻塞的问题。

4.2.5. 防止只读节点上长事务阻塞DDL操作

PolarDB支持参数polar_slave_work_on_nonblock_mdl_mode。开启该参数后，可以防止PolarDB只读节点上的长事务阻塞主节点上的DDL操作。本文介绍如何为PolarDB开启该参数。

使用限制

集群版本需为PolarDB MySQL引擎8.0版本且修订版本为8.0.1.1.23或以上。

> **说明** 您可以通过查询版本号确认集群的内核版本。

仅当PolarDB只读节点事务隔离级别为 `Read Committed` 或 `Read Uncommitted` 时，polar_slave_work_on_nonblock_mdl_mode参数才有效。

polar_slave_work_on_nonblock_mdl_mode参数仅在PolarDB的只读节点生效。

背景信息

PolarDB只读节点上的长事务会一直持有访问过的数据表的元数据锁（MDL），它将导致PolarDB主节点上的DDL操作无法完成MDL同步，并最终超时，返回错误 `ERROR 8007 (HY000): Fail to get MDL on replica during DDL synchronize` 。

> **说明** MDL同步过程的超时时间由参数 `replica_lock_wait_timeout` 控制，该参数值默认为50s。

当发生该问题时，在PolarDB主节点上执行命令 `show processlist` ，查询结果中 `State` 字段若提示 `Wait for syncing with replicas` ，您可以在PolarDB只读节点上执行命令 `select * from information_schema.innodb_log_mdl_slot where slot_state = "SLOT_ACQUIRING"` 查询处于等待状态的MDL信息。若该命令返回如下结果，则表明PolarDB只读节点上存在MDL等待问题。

```
+---------+---------------+-----------+----------+-----------+
| slot_id | slot_state    | slot_name | slot_lsn | thread_id |
+---------+---------------+-----------+----------+-----------+
|       0 | SLOT_ACQUIRING | test/t    | 35025648 | thread-0  |
+---------+---------------+-----------+----------+-----------+
```

注意事项

polar_slave_work_on_nonblock_mdl_mode只能解决长事务导致的DDL阻塞问题。开启polar_slave_work_on_nonblock_mdl_mode后，您仍将可能遇到大查询（即耗时超过MDL同步过程超时时间的查询）导致的MDL同步失败。

由于lock table和flush table获取的MDL需要显式通过unlock tables来释放，因此polar_slave_work_on_nonblock_mdl_mode无法解决lock table和flush table导致的MDL阻塞问题。

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在左侧导航栏中选择**配置与管理 > 参数配置**。
5. 找到目标参数loose_polar_slave_work_on_nonblock_mdl_mode，单击**当前值**列的图标，在弹出的对话框中，输入新的参数值，单击**确定**。

6. 单击左上角**提交修改**，在**保存改动**对话框中，单击**确定**。

使用示例

开启polar_slave_work_on_nonblock_mdl_mode后，在只读节点上，同一事务将可能看到不一样的表结构，如下：

1. 首先，在只读节点上查询 `test.t` ：

```
mysql> begin;
mysql> select * from test.t;
+------+
| a    |
+------+
|    1 |
+------+
1 row in set (0.00 sec)
```

2. 然后，在主节点上对 `test.t` 进行DDL操作：

```
mysql> alter table test.t add column b int;
Query OK, 0 rows affected (0.32 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3. 最后，再次在只读节点上查询 `test.t` ：

```
mysql> select * from test.t;
+------+------+
| a    | b    |
+------+------+
|    1 | NULL |
+------+------+
1 row in set (0.00 sec)
```

通过上述示例，可以看出在开启polar_slave_work_on_nonblock_mdl_mode的情况下，由于在主节点上进行了DDL操作，从而在只读节点上的同一个事务看到了不同的列数目。如果关闭polar_slave_work_on_nonblock_mdl_mode，在主节点上的DDL操作将持续等待只读节点上的事务，直至该事务被提交或等待超时（报错： `ERROR 8007 (HY000): Fail to get MDL on replica during DDL synchronize` ）。

4.2.6. 查看DDL执行状态和MDL锁状态

PolarDB支持Polar Performance Schema功能，它可以监测数据库中DDL语句的执行状态及MDL锁状态。Polar Performance Schema属于轻量化的状态监测功能，与MySQL的Performance Schema功能相比，该功能内存占用更小，性能开销更低。本文主要介绍如何借助Polar Performance Schema功能查看DDL语句的执行状态及MDL锁状态。

前提条件

集群版本需为PolarDB MySQL引擎8.0.1版本且Revision version为8.0.1.1.21及以上，您可以通过查询版本号确认集群版本。

注意事项

请连接PolarDB的主地址查看DDL语句的执行状态。关于如何查看主地址，请参见查看连接地址和端口。

操作步骤

> **说明**
> - 本文介绍的使用方法仅支持使用innodb存储引擎的表。
> - 开启Performance Schema功能后，Polar Performance Schema功能会自动关闭。

1. 开启Polar Performance Schema功能。

   您需要在控制台中将loose_polar_performance_schema参数设为ON，该参数需重启集群后才能生效。具体操作请参见设置集群参数。

   Polar Performance Schema功能的相关参数说明如下：

参数	说明
loose_polar_performance_schema	控制是否启用Polar Performance Schema功能。取值： ○ ON：开启Polar Performance Schema ○ OFF：关闭Polar Performance Schema

参数	说明
performance_schema_max_thread_instances	配置Polar Performance Schema监控的最大线程数。取值范围：-1~65536。取值为-1时，表示自适应。 **说明** 该参数已进行调优，不建议用户自行修改。
performance_schema_max_metadata_locks	配置Polar Performance Schema监控的最大MDL锁数。取值范围：-1~1048576。取值为-1时，表示自适应。 **说明** 该参数已进行调优，不建议用户自行修改。

集群启动后，可通过如下指令，确认是否成功开启Polar Performance Schema功能。

```
SHOW VARIABLES LIKE 'polar_performance_schema';
```

显示结果如下：

```
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| polar_performance_schema | ON    |
+--------------------------+-------+
1 row in set (0.00 sec)
```

2. 查看DDL语句执行状态和MDL锁状态。
   - 在DDL语句执行过程中，通过执行以下命令查看 `performance_schema.events_stages_current` 表，可以获取当前DDL语句的执行状态：

     ```
     SELECT THREAD_ID,EVENT_ID, EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED,(WORK_COMPLETED
     /WORK_ESTIMATED)*100 as PROGRESS FROM performance_schema.events_stages_current;
     ```

     查询结果如下：

     ```
     +-----------+----------+--------------------------------------------------------------+------
     ----------+----------------+----------+
     | THREAD_ID | EVENT_ID | EVENT_NAME                                                   | WORK_
     COMPLETED | WORK_ESTIMATED | PROGRESS |
     +-----------+----------+--------------------------------------------------------------+------
     ----------+----------------+----------+
     |   3057989 |       13 | stage/innodb/alter table (read PK and internal sort) |
     56634 |         330135 |  17.1548 |
     +-----------+----------+--------------------------------------------------------------+------
     ----------+----------------+----------+
     1 row in set (0.00 sec)
     ```

     执行以下命令，您可以查看当前事件对应的SQL语句。

```
SELECT esc.THREAD_ID, esc.EVENT_NAME, esc.WORK_COMPLETED, esc.WORK_ESTIMATED, pl.INFO
FROM performance_schema.events_stages_current esc LEFT JOIN performance_schema.thread
s th ON esc.thread_id = th.thread_id LEFT JOIN information_schema.PROCESSLIST pl ON t
h.PROCESSLIST_ID = pl.ID;
```

查询结果如下：

```
+-----------+-----------------------------------------------------+----------------+
----------------+-------------------------------------------------------------------
--------------------+
| THREAD_ID | EVENT_NAME                                          | WORK_COMPLETED |
WORK_ESTIMATED | INFO
|
+-----------+-----------------------------------------------------+----------------+
----------------+-------------------------------------------------------------------
--------------------+
|   3057989 | stage/innodb/alter table (read PK and internal sort) |          77034 |
330519 | ALTER TABLE test.test ALGORITHM=INPLACE, ADD testA VARCHAR(20) NOT NULL DEFA
ULT 'testA' |
+-----------+-----------------------------------------------------+----------------+
----------------+-------------------------------------------------------------------
--------------------+
1 row in set (0.00 sec)
```

- 通过执行以下命令查看表 `performance_schema.metadata_locks` ，可以获取当前集群中MDL锁的使用情况：

```
SELECT * FROM performance_schema.metadata_locks;
```

查询结果如下：

```
+-------------+-------------------+-----------------+-------------+----------------
-------+--------------------+--------------+-------------+------------------------+
-----------------+----------------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME      | COLUMN_NAME | OBJECT_INSTANCE
_BEGIN | LOCK_TYPE           | LOCK_DURATION | LOCK_STATUS | SOURCE                 |
OWNER_THREAD_ID | OWNER_EVENT_ID |
+-------------+-------------------+-----------------+-------------+----------------
-------+--------------------+--------------+-------------+------------------------+
-----------------+----------------+
| GLOBAL      | NULL               | NULL             | NULL        |       139949462
878336 | INTENTION_EXCLUSIVE | STATEMENT     | GRANTED     | sql_base.cc:3103       |
3055785 |              1 |
| TABLE       | test               | test             | NULL        |       139931318
980224 | SHARED_WRITE        | TRANSACTION   | GRANTED     | sql_parse.cc:6479      |
3055785 |              1 |
| COMMIT      | NULL               | NULL             | NULL        |       139931318
980480 | INTENTION_EXCLUSIVE | EXPLICIT      | GRANTED     | handler.cc:1669        |
3055785 |              1 |
| TABLE       | performance_schema | metadata_locks   | NULL        |       139934227
366144 | SHARED_READ         | TRANSACTION   | GRANTED     | sql_parse.cc:6479      |
3057612 |              1 |
| GLOBAL      | NULL               | NULL             | NULL        |       139934216
849664 | INTENTION_EXCLUSIVE | STATEMENT     | GRANTED     | sql_base.cc:5519       |
3057989 |             13 |
| SCHEMA      | test               | NULL             | NULL        |       139934216
849408 | INTENTION_EXCLUSIVE | TRANSACTION   | GRANTED     | sql_base.cc:5506       |
3057989 |             13 |
| TABLE       | test               | test             | NULL        |       139934216
848640 | SHARED_UPGRADABLE   | TRANSACTION   | GRANTED     | sql_parse.cc:6479      |
3057989 |             13 |
| BACKUP LOCK | NULL               | NULL             | NULL        |       139934216
849280 | INTENTION_EXCLUSIVE | TRANSACTION   | GRANTED     | sql_base.cc:5526       |
3057989 |             13 |
| TABLESPACE  | NULL               | test/test        | NULL        |       139934216
848384 | INTENTION_EXCLUSIVE | TRANSACTION   | GRANTED     | lock.cc:815            |
3057989 |             13 |
| TABLE       | test               | #sql-17d9_2ea89a | NULL        |       139934216
848896 | EXCLUSIVE           | STATEMENT     | GRANTED     | sql_table.cc:15054     |
3057989 |             13 |
| GLOBAL      | NULL               | NULL             | NULL        |       139934216
850176 | INTENTION_EXCLUSIVE | TRANSACTION   | GRANTED     | dictionary_impl.cc:416 |
3057989 |             13 |
| TABLESPACE  | NULL               | test/test        | NULL        |       139934216
849920 | EXCLUSIVE           | TRANSACTION   | GRANTED     | dictionary_impl.cc:397 |
3057989 |             13 |
+-------------+-------------------+-----------------+-------------+----------------
-------+--------------------+--------------+-------------+------------------------+
-----------------+----------------+
12 rows in set (0.00 sec)
```

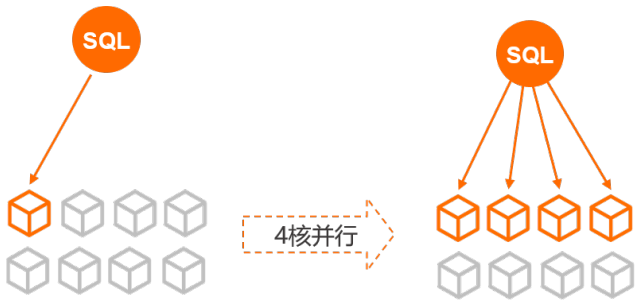
4.3. 大查询优化（Parallel Query）

4.3.1. 并行查询（Parallel Query）

PolarDB MySQL引擎8.0版本重磅推出并行查询框架，当您的查询数据量到达一定阈值，就会自动启动并行查询框架，从而使查询耗时指数级下降。

在存储层将数据分片到不同的线程上，多个线程并行计算，将结果流水线汇总到总线程，最后总线程做些简单归并返回给用户，提高查询效率。

并行查询（Parallel Query）利用多核CPU的并行处理能力，以8核32 GB（独享规格）的PolarDB MySQL引擎集群版为例，示意图如下所示。



前提条件

PolarDB集群版本需为PolarDB MySQL引擎8.0版本且修订版本需满足如下条件：

- 8.0.1.0.5或以上。
- 8.0.2.1.4.1或以上。

如何查看集群版本，请参见查询版本号。

应用场景

并行查询适用于大部分SELECT语句，例如大表查询、多表连接查询、计算量较大的查询。对于非常短的查询，效果不太显著。

- 轻分析类业务

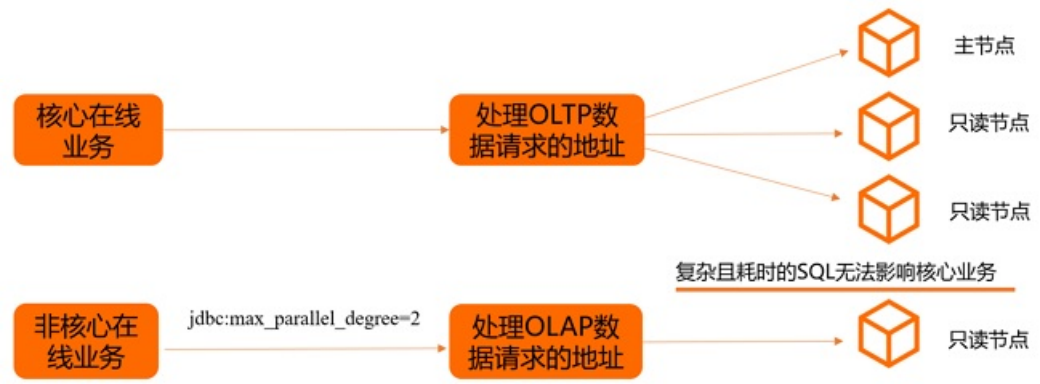
  报表查询通常SQL复杂而且比较耗费时间，通过并行查询可以加速单次查询效率。

- 系统资源相对空闲

  并行查询会使用更多的系统资源，只有当系统的CPU较多、IO负载不高、内存够大的时候，才可以充分使用并行查询来提高资源利用率和查询效率。

- 在离线混合场景

  不同的业务用不同的连接地址，使用不同的数据库节点，避免互相影响。如果您为单节点的集群地址开启了并行查询，那么就会通过处理OLAP数据请求的地址对该单节点地址下的对应只读节点进行访问。



使用说明

> **说明** 只读节点和主节点都支持并行查询功能。主节点上并行查询默认关闭，只读节点上默认开启。

- 通过链接地址或者系统参数开启和关闭并行查询，并设置并行度
  - 在控制台的编辑链接地址页面，可开启或关闭并行查询，并设置并行度参数。具体操作请参见配置数据库代理。

    > **说明**
    > - 当集群地址的读写模式为**只读**时支持该配置。
    > - 此配置的优先级高于集群参数max_parallel_degree。

  - 在控制台的参数配置页面，通过全局参数max_parallel_degree来设置并行度（即控制每一条SQL最多使用多少个线程并行执行）。若设置为0，则表示关闭并行查询。具体操作请参见设置集群参数。

    > **说明** 您可以在使用过程中随时修改该参数，无需重启数据库。

  并行查询推荐设置以及相关说明如下：

  - 并行度参数从低到高逐渐增加，建议不要超过CPU核数的四分之一 。集群的CPU内核数大于等于8才支持开启并行查询，小规格的集群不建议开启该参数。例如，刚开始使用并行查询时，设置并行度参数为*2*，试运行一天后，如果CPU压力不大，可以往上增加；如果CPU压力较大，停止增加。
  - 当集群的CPU内核数大于等于8时，并行度参数默认为2；当集群的CPU内核数大于等于16时，并行度参数默认为4；当集群的CPU内核数大于等于88时，并行度参数默认为8。
  - 为保证max_parallel_degree参数也能兼容其它程序版本以及考虑到MySQL配置文件中该参数的默认配置，PolarDB在控制台上增加了前缀 `loose` ，因此参数名称是loose_max_parallel_degree，这样保证其他版本接受该参数时也不会报错， 详情请参见Program Option Modifiers。
  - 打开并行查询功能时， 需要设置innodb_adaptive_hash_index参数为*OFF*，innodb_adaptive_hash_index参数开启会影响并行查询的性能。

  除了Global集群级别，您也可以单独调整某Session内SQL查询的并行度。例如，通过Session级环境变量，加到JDBC的连接串配置中，则可以对某个应用程序单独设置并行度。

  ```
  set max_parallel_degree = n
  ```

- 通过Hint来控制并行查询

使用Hint语法可以对单个语句进行控制，例如系统默认关闭并行查询情况下，但需要对某个高频的慢SQL查询进行加速，此时就可以使用Hint对特定SQL进行加速。更多详情，请参见通过Hint控制。

- 通过设置阈值来控制优化器是否选择并行执行

  PolarDB提供了两个阈值来控制优化器是否选择并行执行，SQL语句只要满足其中任意一个条件，优化器就会考虑并行执行。

  - records_threshold_for_parallelism

    若优化器估算出语句中存在扫描记录数超过该阈值的表，优化器会考虑选择并行执行计划。默认值为10000。若您的业务量较小或复杂查询业务并发较低，您可以选择将该阈值设置为2000或以上。

    > **说明** 上文提到的扫描记录数是根据对应表的统计信息进行估算得出的值，可能存在一定的误差。

  - cost_threshold_for_parallelism

    若优化器估算查询的串行执行代价超过该阈值，优化器会考虑选择并行执行计划。默认值为50000。

相关参数和变量

系统参数

参数名	级别	描述
max_parallel_degree	Global、Session	单个查询的最大并行度，即最多使用多少个Worker进行并行执行。 • 取值范围：[0-1024] • 默认值：0，表示关闭并行查询。 **说明** PolarDB优化器可能会对主查询和子查询分别并行执行，如果同时并行执行，它们的最大Worker数不能超过max_parallel_degree的值，整个查询使用的Worker数为主查询和子查询使用的Worker数之和。
parallel_degree_policy	Global	设置单个查询的并行度配置策略，取值范围如下： • `TYPICAL`：PolarDB选择查询并行度时不会考虑数据库负载（如CPU使用率等），而尽可能与`max_parallel_degree`设置的并行度保持一致。 • `AUTO`：PolarDB会根据数据库负载（如CPU使用率等）来决定是否禁止并行查询计划，并会根据查询代价选择并行度。 • `REPLICA_AUTO`（默认）：仅只读节点会根据数据库负载（如CPU使用率等）决定是否禁止并行查询计划，并会根据查询代价选择并行度，而主节点不会开启并行查询。 **说明** 更多关于并行度配置策略的详细介绍，请参见并行度控制策略。

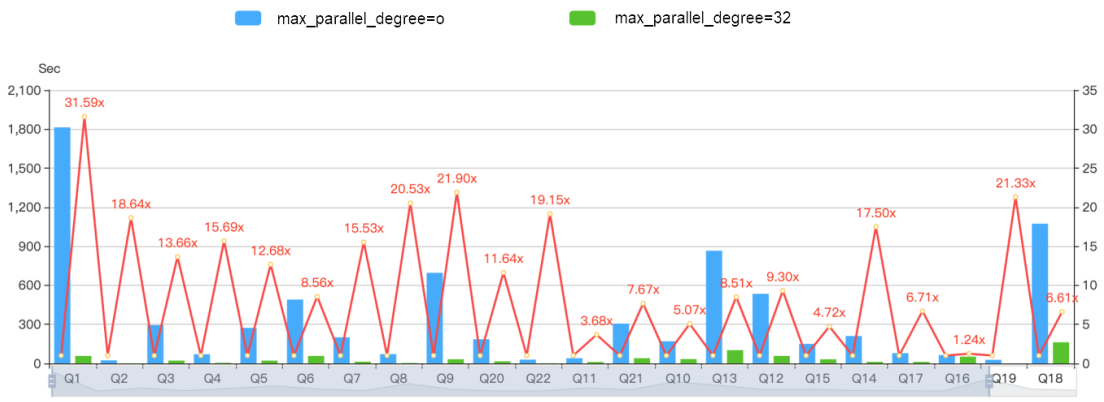
参数名	级别	描述
records_threshold_for_parallelism	Session	若优化器估算出语句中存在扫描记录数超过该阈值的表，优化器会考虑选择并行执行计划。 • 取值范围：[0-18446744073709551615] • 默认值：10000。 **说明** 若您的业务量较小或复杂查询业务并发较低，您可以选择将该阈值设置为2000或以上。
cost_threshold_for_parallelism	Session	若优化器估算查询的串行执行代价超过该阈值，优化器会考虑选择并行执行计划。 • 取值范围：[0-18446744073709551615] • 默认值：50000。

状态变量

变量名	级别	描述
Parallel_workers_created	Session、Global	从Session启动开始，生成Parallel Worker的个数。
Gather_records	Session、Global	Gather记录总数。
PQ_refused_over_memory_soft_limit	Session、Global	由于内存限制没有启用并行的查询数。
PQ_refused_over_total_workers	Session、Global	由于总Worker数限制没有启用并行的查询数。
Total_used_query_memory	Global	当前总的已使用的查询内存（Virtual Memory）。
Total_running_parallel_workers	Global	当前正在运行的Parallel Worker的数目。

性能指标

本次测试将使用TPC-H生成100 GB数据来测试PolarDB MySQL引擎8.0版本集群的性能指标。测试用的PolarDB集群规格为32核256 GB（独享规格），并行度max_parallel_degree分别设置为32和0，具体测试步骤请参见并行查询性能（OLAP）。



通过以上测试结果图得出结论，TPC-H中95%的SQL可以被加速，且在被加速的SQL语句中，有70%的SQL语句的执行速度是未使用并行查询的8倍以上。

> **说明** 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。

并行执行EXPLAIN

更多关于EXPLAIN执行计划输出中与并行查询相关的内容，请参见并行执行EXPLAIN。

相关概念

- 并行扫描

  在并行扫描中，每个Worker并行独立扫描数据表中的数据。Worker扫描产生的中间结果集将会返回给Leader线程，Leader线程通过Gather操作收集产生的中间结果，并将所有结果汇总返回到客户端。

- 多表并行连接

  并行查询会将多表连接操作完整的下推到Worker上去执行。PolarDB优化器只会选择一个自认为最优的表进行并行扫描，而除了该表外，其他表都是一般扫描。每个Worker会将连接结果集返回给Leader线程，Leader线程通过Gather操作进行汇总，最后将结果返回给客户端。

- 并行排序

  PolarDB优化器会根据查询情况，将ORDER BY下推到每个Worker里执行，每个Worker将排序后的结果返回给Leader，Leader通过Gather Merge Sort操作进行归并排序，最后将排序结果返回到客户端。

- 并行分组

  PolarDB优化器会根据查询情况，将GROUP BY下推到Worker上去并行执行。每个Worker负责部分数据的GROUP BY。Worker会将GROUP BY的中间结果返回给Leader，Leader通过Gather操作汇总所有数据。这里PolarDB优化器会根据查询计划情况来自动识别是否需要再次在Leader上进行GROUP BY。例如，如果GROUP BY使用了Loose Index Scan，Leader上将不会进行再次GROUP BY；否则Leader会再次进行GROUP BY操作，然后把最终结果返回到客户端。

- 并行聚集

  并行查询执行聚集函数下推到Worker上并行执行。并行聚集是通过两次聚集来完成的。第一次，参与并行查询部分的每个Worker执行聚集步骤。第二次，Gather或Gather Merge节点将每个Worker产生的结果汇总到Leader。最后，Leader会将所有Worker的结果再次进行聚集得到最终结果。

- 子查询支持

  在并行查询下子查询有四种执行策略：

- 在Leader线程中串行执行

  当子查询不可并行执行时，例如在2个表JOIN，在JOIN条件上引用了用户的函数，此时子查询会在Leader线程上进行串行查询。

- 在Leader上并行执行（Leader会启动另一组Worker）

  生成并行计划后，在Leader上执行的计划包含有支持并行执行的子查询，但这些子查询不能提前并行执行（即不能采用Shared access)。例如，当前如果子查询中包括window function，子查询就不能采用Shared access策略。

- Shared access

  生成并行计划后，Worker的执行计划引用了可并行执行的子查询，PolarDB优化器会选择先提前并行执行这些子查询，让Worker可以直接访问这些子查询的结果。

- Pushed down

  生成并行计划后，Worker执行计划引用了相关子查询，这些子查询会被整体推送到Worker上执行。

相关视频

4.3.2. 并行执行EXPLAIN

本文介绍如何使用EXPLAIN语句查看执行计划输出中与并行查询相关的内容。

查询用表

本文示例中使用 `pq_test` 表进行并行查询测试，其中：

- 表结构如下：

```
mysql> SHOW CREATE TABLE pq_test\G
*************************** 1. row ***************************
       Table: pq_test
Create Table: CREATE TABLE `pq_test` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `help_topic_id` INT(10) UNSIGNED NOT NULL,
  `name` CHAR(64) NOT NULL,
  `help_category_id` SMALLINT(5) UNSIGNED NOT NULL,
  `description` TEXT NOT NULL,
  `example` TEXT NOT NULL,
  `url` TEXT NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=21495809 DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

- 表大小如下：

```
mysql> SHOW TABLE STATUS\G
*************************** 1. row ***************************
           Name: pq_test
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 20064988
 Avg_row_length: 1898
    Data_length: 38085328896
Max_data_length: 0
   Index_length: 0
      Data_free: 4194304
 Auto_increment: 21495809
    Create_time: 2019-07-30 01:35:27
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.02 sec)
```

- 查询SQL如下：

```
SELECT COUNT(*) FROM pq_test;
```

EXPLAIN查询语句

- 通过EXPLAIN语句查看不使用并行查询的情况。查询语句如下：

```
SET max_parallel_degree=0; EXPLAIN SELECT COUNT(*) FROM pq_test\G
```

查询结果如下：

```
*************************** 1. row ***************************
           Id: 1
  Select_type: SIMPLE
        Table: pq_test
  Partitions: NULL
         Type: index
Possible_keys: NULL
          Key: PRIMARY
      Key_len: 8
          Ref: NULL
         Rows: 20064988
     Filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.03 sec)
```

- 通过EXPLAIN语句查看使用并行查询的情况，查询语句如下：

```
EXPLAIN SELECT COUNT(*) FROM pq_test\G
```

查询结果如下：

```
*************************** 1. row ***************************
           Id: 1
  Select_type: SIMPLE
        Table: &lt;gather2&gt;
   Partitions: NULL
         Type: ALL
Possible_keys: NULL
          Key: NULL
      Key_len: NULL
          Ref: NULL
         Rows: 20064988
     Filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
           Id: 2
  Select_type: SIMPLE
        Table: pq_test
   Partitions: NULL
         Type: index
Possible_keys: NULL
          Key: PRIMARY
      Key_len: 8
          Ref: NULL
         Rows: 10032494
     Filtered: 100.00
        Extra: Parallel scan (2 workers); Using index
2 rows in set, 1 warning (0.00 sec)
```

从上述结果可以看出：

- 上述并行计划中包含了Gather操作，该操作负责汇总所有Worker返回的中间结果。
- 从执行计划输出的Extra信息中可以看到 `pq_test` 表使用了Parallel scan（并行扫描）策略，期望用2个Workers来并行执行。

- 通过带有子查询的EXPLAIN语句查看使用并行查询的情况，查询语句如下：

```
EXPLAIN SELECT a, (select sum(t2.b) from t2 where t2.a = t1.b) FROM t1 WHERE (a, b) IN (S
ELECT b, MAX(a) FROM t2 GROUP BY b)\G
```

查询结果如下：

```
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: <gather1>
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2
     filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
```

```
           id: 1
  select_type: PRIMARY
        table: t1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2
     filtered: 100.00
        Extra: Parallel scan (1 workers); Using where
*************************** 3. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2
     filtered: 50.00
        Extra: Parallel pushdown; Using where
*************************** 4. row ***************************
           id: 3
  select_type: SUBQUERY
        table: <gather3>
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Shared access; Using temporary
*************************** 5. row ***************************
           id: 3
  select_type: SIMPLE
        table: t2
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2
     filtered: 100.00
        Extra: Parallel scan (1 workers); Using temporary
5 rows in set, 2 warnings (0.02 sec)
```

从上述结果可以看出：

- `select_type` 为 `SBUQUERY` 的子查询中，`Extra` 显示 `Parallel pushdown`，表示该子查询使用了 `Parallel pushdown` 策略，即整个子查询被整个下推到Worker去执行。
- `select_type` 为 `DEPENDENT SUBQUERY` 的子查询中，`Extra` 显示 `Shared access`，表示该子查询使用了 `Shared access` 策略，即PolarDB优化器选择提前并行执行该子查询并将执行结果Share给所有Worker。

4.3.3. 并行度控制策略

PolarDB支持通过parallel_degree_policy参数来设置并行查询中并行度的配置策略。本文将介绍相关参数说明。

前提条件

集群版本需为PolarDB MySQL引擎且修订版本为8.0.1.1.11或以上，详情请参见查询版本号。

并行度控制参数

参数	级别	说明
parallel_degree_policy	Global	设置单个查询的并行度配置策略，取值范围如下： • `TYPICAL` • `AUTO` • `REPLICA_AUTO`（默认） **说明** 更多关于并行查询的使用方式，请参见并行查询（Parallel Query）。

TYPICAL策略

TYPICAL策略模式下，PolarDB选择查询并行度时不会考虑数据库负载（如CPU使用率等），而尽可能与 `max_parallel_degree` 设置的并行度保持一致。

AUTO策略

AUTO策略下，PolarDB会根据数据库的CPU、内存或IOPS资源的使用率来决定是否禁止并行查询计划，并支持在需要并行执行的前提下，自定义并行查询的并行度选择策略。

参数	级别	取值	说明
loose_auto_dop_cpu_pct_hwm		• 取值范围：0~100 • 默认值：70	CPU使用率阈值。若CPU使用率超过阈值，PolarDB会禁止并行查询计划。
loose_auto_dop_mem_pct_hwm		• 取值范围：0~100 • 默认值：90	内存使用率阈值。若内存使用率超过阈值，PolarDB会禁止并行查询计划。

参数	级别	取值	说明
loose_auto_dop_iops_pct_hwm	Global	• 取值范围：0-100 • 默认值：80	IOPS使用率阈值。若IOPS使用率超过阈值，PolarDB会禁止并行查询计划。
loose_auto_dop_low_degree_cost		• 取值范围：0~18446744073709551615 • 默认值：500000	自动并行度选择策略。在确定需要并行执行后，PolarDB会根据如下标准选择查询并行度： • 当优化器估算查询的串行执行代价低于该值时，查询并行度为2。 • 当优化器估算查询的串行执行代价大于或等于该值时，查询并行度将尽可能与max_parallel_degree设置的并行度保持一致。 **说明** 该参数仅用于在PolarDB确定需要并行执行后，选择查询的并行度，而不用于选择是否需要并行执行。

REPLICA_AUTO策略

REPLICA_AUTO策略下，仅只读节点会根据数据库的CPU、内存或IOPS使用率决定是否禁止并行查询计划，而主节点不会选择并行执行计划。支持的配置参数与AUTO策略的一致。

4.3.4. Hash Join的并行执行

本文介绍如何在PolarDB的并行查询中使用Hash Join功能。

前提条件

集群版本需为PolarDB MySQL引擎 8.0集群版，且Revision version为8.0.2.1.0或以上，您可以参见查询版本号确认集群版本。

使用方法

- 语句

  目前在PolarDB中Hash Join只能通过 `EXPLAIN FORMAT=TREE` 语句来显示。

- 示例

  如下示例中创建了2个表，且在表里插入了一些数据：

```
CREATE TABLE t1 (c1 INT, c2 INT);
CREATE TABLE t2 (c1 INT, c2 INT);
INSERT INTO t1 VALUES (1,1),(2,2),(3,3),(5,5);
INSERT INTO t2 VALUES (1,1),(2,2),(3,3),(5,5);
```

  如下是上述2个表的JOIN查询计划：

```
EXPLAIN FORMAT=TREE
 EXPLAIN
  -> Gather (slice: 1; workers: 4)  (cost=10.82 rows=4)
    -> Parallel inner hash join (t2.c2 = t1.c1)  (cost=0.57 rows=1)
        -> Parallel table scan on t2, with parallel partitions: 1  (cost=0.03 rows=1)
        -> Parallel hash
            -> Parallel table scan on t1, with parallel partitions: 1  (cost=0.16 rows=1)
```

上述是并行度为4的并行查询计划（即PolarDB会启用4个Worker来执行查询）。其中t1表会执行Parallel Scan，即由4个Worker分扫这个表，每个Worker使用t1的一部分数据建立各自的Hash表，再和整个t2表执行JOIN操作，最后收集（Gather）在Leader，得到整个查询的结果。

相关视频

4.3.5. Semi-Join的并行执行

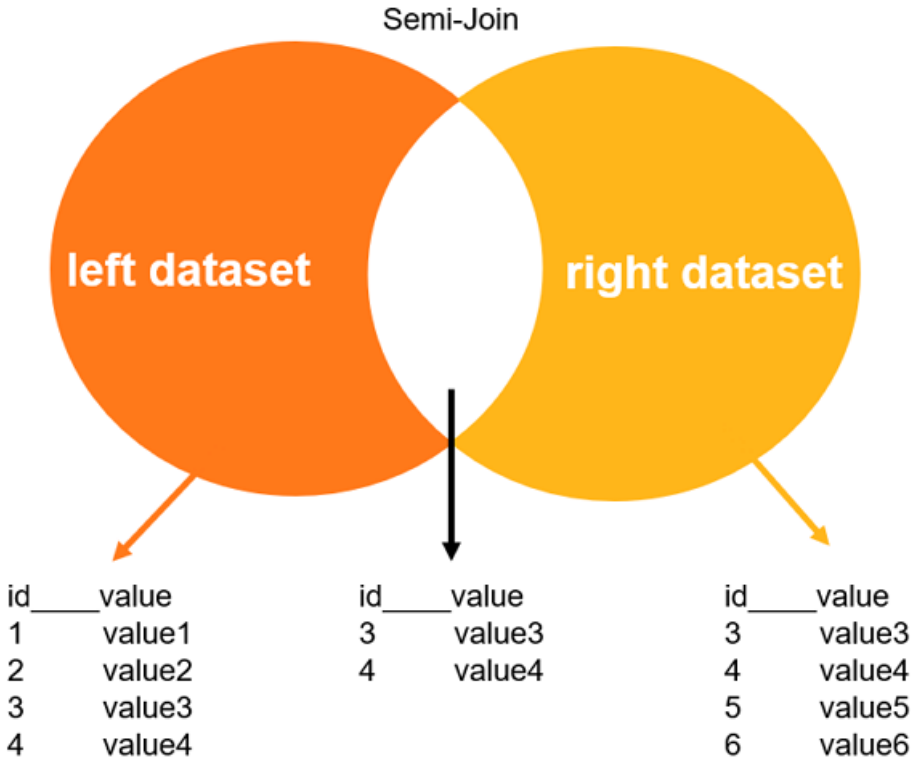
您可以使用Semi-Join半连接优化子查询，减少查询次数，提高查询性能，本文将介绍Semi-Join半连接的基本信息和操作方法。

前提条件

PolarDB集群版本需为PolarDB MySQL引擎8.0且Revision version为8.0.1.1.2或以上，您可以参见查询版本号确认集群版本。

背景信息

MySQL 5.6.5引入了Semi-Join半连接，当外表在内表中找到匹配的记录之后，Semi-Join会返回外表中的记录。但即使在内表中找到多条匹配的记录，外表也只会返回已经存在于外表中的记录。而对于子查询，外表的每个符合条件的元组都要执行一轮子查询，效率比较低下。此时使用半连接操作优化子查询，会减少查询次数，提高查询性能，其主要思路是将子查询上拉到父查询中，这样内表和外表是并列关系，外表的每个符合条件的元组，只需要在内表中找符合条件的元组即可，所以效率会大大提高。

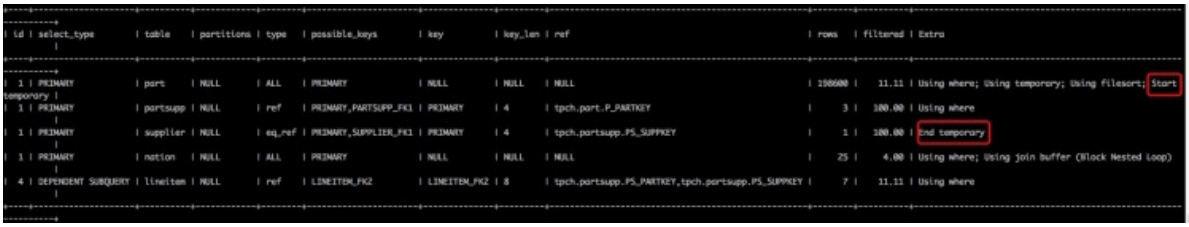


策略

Semi-Join主要使用了如下策略：

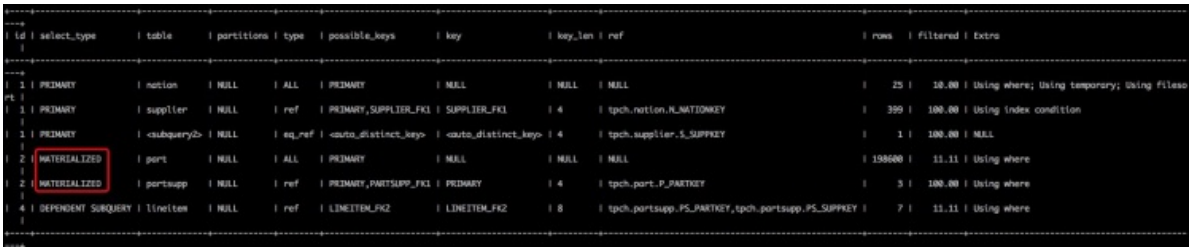
- DuplicateWeedout Strategy

  该策略创建由 `row id` 组成唯一ID的临时表，再通过该唯一ID达到去重目的。



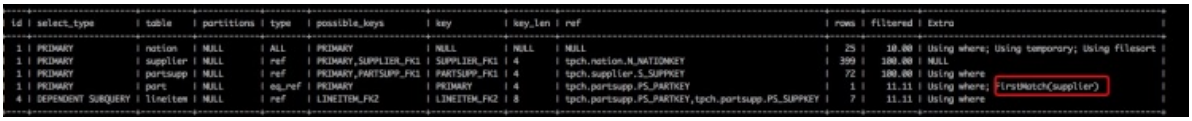
- Materialization Strategy

  该策略将 `nested tables` 物化到临时表中，再通过查找物化表或者遍历物化表查找外表的方法达到去重目的。



- Firstmatch Strategy

  该策略采用顺序查找表的方式，找到第一个匹配的记录后立即跳转到最后一个外表，并对外表的下一条记录执行JOIN操作，从而达到去重的目的



- LooseScan Strategy

  该策略对内表基于索引（Index）进行分组，分组后与外表执行JOIN（进行Condition的匹配）操作，如果存在匹配的记录，则提取外表的记录，内表选取下一个分组继续进行计算，从而达到去重目的。

语法

Semi-Join通常使用IN或EXISTS作为连接条件。

```
IN
SELECT *
FROM Employee
WHERE DeptName IN (
  SELECT DeptName
  FROM Dept
)
```

```
EXISTS
SELECT *
FROM Employee
WHERE EXISTS (
  SELECT 1
  FROM Dept
  WHERE Employee.DeptName = Dept.DeptName
)
```

并行Semi-Join性能提升

对于选择Semi-Join策略的查询，PolarDB对Semi-Join所有策略实现了并行加速，通过拆分Semi-Join的任务，多线程模型并行运行任务集，强化去重能力，使查询性能得到了显著的提升，以Q20为例。

```
SELECT
   s_name,
   s_address
FROM
   supplier,
   nation
WHERE
   s_suppkey IN
   (
      SELECT
         ps_suppkey
      FROM
         partsupp
      WHERE
         ps_partkey IN
         (
            SELECT
               p_partkey
            FROM
               part
            WHERE
               p_name LIKE '[COLOR]%'
         )
         AND ps_availqty > (
         SELECT
            0.5 * SUM(l_quantity)
         FROM
            lineitem
         WHERE
            l_partkey = ps_partkey
            AND l_suppkey = ps_suppkey
            AND l_shipdate >= date('[DATE]')
            AND l_shipdate < date('[DATE]') + interval '1' year )
   )
   AND s_nationkey = n_nationkey
   AND n_name = '[NATION]'
ORDER BY
   s_name;
```

本文例子中将物化处理提前，并且以并行度（DOP）为32执行，后续的处理通过共享之前的物化表，同样充分发挥CPU的处理能力，将主查询的并行能力最大化，在如下的执行计划中，在标准TPCH SCALE为1 G的数据量场景下，开启并行后，双层并行处理能力：

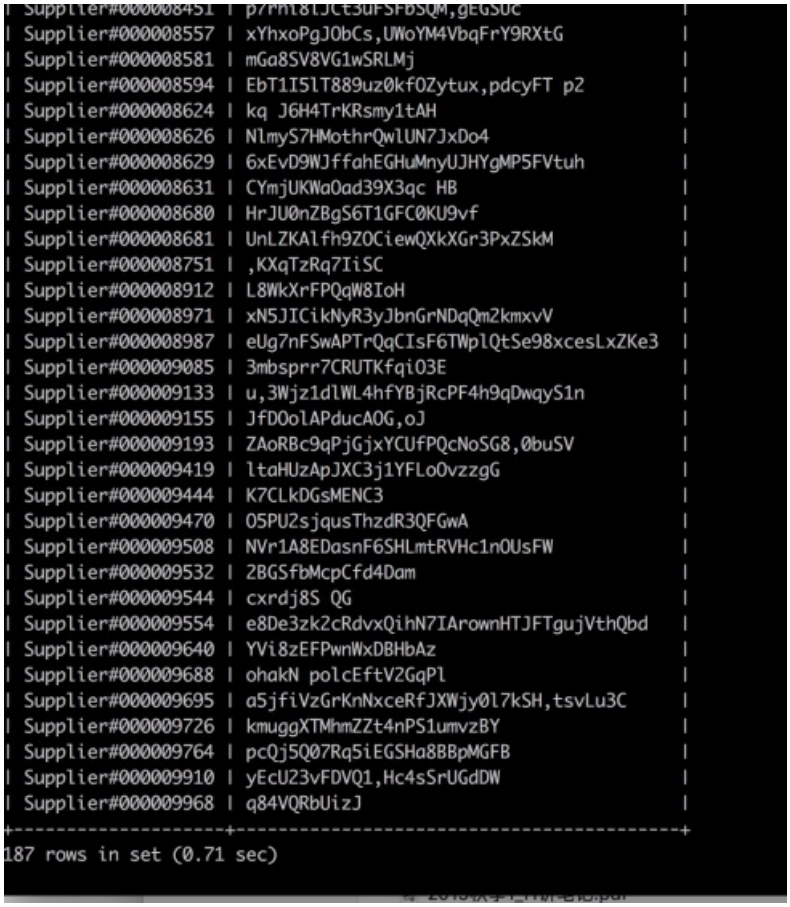
> **说明** 本文的TPC-H的实现基于TPC-H的基准测试，并不能与已发布的TPC-H基准测试结果相比较，本文中的测试并不符合TPC-H基准测试的所有要求。

```
root:dbt3> explain select  s_name,     s_address from supplier, nation where s_suppkey in ( select  ps_suppkey from partsupp where ps_partkey in ( select p_partkey from part where p_name like 'peru%' ) and ps_availqty > ( select 0.5
* sum(l_quantity) from lineitem where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_shipdate >= '1993-01-01' and l_shipdate < date_add( '1993-01-01' ,interval '1' year) ) ) and s_nationkey = n_nationkey and n_name = 'PERU'
order by s_name;
+----+--------------------+-------------+------------+--------+---------------------------------------+---------------------+---------+-------------------------------------+---------+----------+-----
-----------------------------------------+
| id | select_type        | table       | partitions | type   | possible_keys                         | key                 | key_len | ref                                 | rows    | filtered | Extr
a                                        |
+----+--------------------+-------------+------------+--------+---------------------------------------+---------------------+---------+-------------------------------------+---------+----------+-----
-----------------------------------------+
|  1 | PRIMARY            | <gather1>   | NULL       | ALL    | NULL                                  | NULL                | NULL    | NULL                                |    3215 |   100.00 | Merg
e sort                                   |
|  1 | PRIMARY            | nation      | NULL       | ALL    | PRIMARY                               | NULL                | NULL    | NULL                                |      25 |    10.00 | Usin
g where; Using temporary; Using filesort |
|  1 | PRIMARY            | supplier    | NULL       | ref    | PRIMARY,i_s_nationkey                 | i_s_nationkey       | 5       | dbt3.nation.n_nationkey             |    1286 |   100.00 | Para
llel scan (32 workers); Using index condition |
|  1 | PRIMARY            | <subquery2> | NULL       | eq_ref | <auto_distinct_key>                   | <auto_distinct_key> | 4       | dbt3.supplier.s_suppkey             |       1 |   100.00 | Para
llel pushdown; Shared access             |
|  2 | MATERIALIZED       | part        | NULL       | ALL    | PRIMARY                               | NULL                | NULL    | NULL                                | 19681079 |    11.11 | Para
llel scan (32 workers); Using where      |
|  2 | MATERIALIZED       | partsupp    | NULL       | ref    | PRIMARY,i_ps_partkey,i_ps_suppkey     | PRIMARY             | 4       | dbt3.part.p_partkey                 |       4 |   100.00 | Usin
g where                                  |
|  4 | DEPENDENT SUBQUERY | lineitem    | NULL       | ref    | i_l_partkey,i_l_suppkey,i_l_partkey_suppkey,i_l_shipdate | i_l_partkey_suppkey | 10      | dbt3.partsupp.ps_partkey,dbt3.partsupp.ps_suppkey |       6 |    31.53 | Para
llel pushdown; Using where               |
+----+--------------------+-------------+------------+--------+---------------------------------------+---------------------+---------+-------------------------------------+---------+----------+-----
-----------------------------------------+
7 rows in set, 3 warnings (0.05 sec)
```

在标准TPCH SCALE为1 G的数据量场景下，串行的执行时间：

```
| Supplier#000008629 | 6xEvD9WJffahEGHuMnyUJHYgMP5FVtuh          |
| Supplier#000008631 | CYmjUKWaOad39X3qc HB                      |
| Supplier#000008680 | HrJU0nZBgS6T1GFC0KU9vf                    |
| Supplier#000008681 | UnLZKAlfh9ZOCiewQXkXGr3PxZSkM             |
| Supplier#000008751 | ,KXqTzRq7IiSC                             |
| Supplier#000008912 | L8WkXrFPQqW8IoH                           |
| Supplier#000008971 | xN5JICikNyR3yJbnGrNDqQm2kmxvV             |
| Supplier#000008987 | eUg7nFSwAPTrQqCIsF6TWplQtSe98xcesLxZKe3   |
| Supplier#000009085 | 3mbsprr7CRUTKfqiO3E                       |
| Supplier#000009133 | u,3Wjz1dlWL4hfYBjRcPF4h9qDwqyS1n          |
| Supplier#000009155 | JfDOolAPducAOG,oJ                         |
| Supplier#000009193 | ZAoRBc9qPjGjxYCUfPQcNoSG8,0buSV           |
| Supplier#000009419 | ltaHUzApJXC3j1YFLoOvzzgG                  |
| Supplier#000009444 | K7CLkDGsMENC3                             |
| Supplier#000009470 | O5PU2sjqusThzdR3QFGwA                     |
| Supplier#000009508 | NVr1A8EDasnF6SHLmtRVHc1nOUsFW             |
| Supplier#000009532 | 2BGSfbMcpCfd4Dam                          |
| Supplier#000009544 | cxrdj8S QG                                |
| Supplier#000009554 | e8De3zk2cRdvxQihN7IArownHTJFTgujVthQbd    |
| Supplier#000009640 | YVi8zEFPwnWxDBHbAz                        |
| Supplier#000009688 | ohakN polcEftV2GqPl                       |
| Supplier#000009695 | a5jfiVzGrKnNxceRfJXWjy0l7kSH,tsvLu3C      |
| Supplier#000009726 | kmuggXTMhmZZt4nPS1umvzBY                  |
| Supplier#000009764 | pcQj5Q07Rq5iEGSHa8BBpMGFB                 |
| Supplier#000009910 | yEcU23vFDVQ1,Hc4sSrUGdDW                  |
| Supplier#000009968 | q84VQRbUizJ                               |
+--------------------+-------------------------------------------+
187 rows in set (1.75 sec)
```

并行开启情况下的执行时间：



```
| Supplier#000008451 | p7Rnt8lJCt3uFSFbSQM,gEGSUC                |
| Supplier#000008557 | xYhxoPgJObCs,UWoYM4VbqFrY9RXtG            |
| Supplier#000008581 | mGa8SV8VG1wSRLMj                          |
| Supplier#000008594 | EbT1I5lT889uz0kfOZytux,pdcyFT p2          |
| Supplier#000008624 | kq J6H4TrKRsmy1tAH                        |
| Supplier#000008626 | NlmyS7HMothrQwlUN7JxDo4                   |
| Supplier#000008629 | 6xEvD9WJffahEGHuMnyUJHYgMP5FVtuh          |
| Supplier#000008631 | CYmjUKWaOad39X3qc HB                      |
| Supplier#000008680 | HrJU0nZBgS6T1GFC0KU9vf                    |
| Supplier#000008681 | UnLZKAlfh9ZOCiewQXkXGr3PxZSkM             |
| Supplier#000008751 | ,KXqTzRq7IiSC                             |
| Supplier#000008912 | L8WkXrFPQqW8IoH                           |
| Supplier#000008971 | xN5JICikNyR3yJbnGrNDqQm2kmxvV             |
| Supplier#000008987 | eUg7nFSwAPTrQqCIsF6TWplQtSe98xcesLxZKe3   |
| Supplier#000009085 | 3mbsprr7CRUTKfqiO3E                       |
| Supplier#000009133 | u,3Wjz1dlWL4hfYBjRcPF4h9qDwqyS1n          |
| Supplier#000009155 | JfDOolAPducAOG,oJ                         |
| Supplier#000009193 | ZAoRBc9qPjGjxYCUfPQcNoSG8,0buSV           |
| Supplier#000009419 | ltaHUzApJXC3j1YFLoOvzzgG                  |
| Supplier#000009444 | K7CLkDGsMENC3                             |
| Supplier#000009470 | O5PU2sjqusThzdR3QFGwA                     |
| Supplier#000009508 | NVr1A8EDasnF6SHLmtRVHc1nOUsFW             |
| Supplier#000009532 | 2BGSfbMcpCfd4Dam                          |
| Supplier#000009544 | cxrdj8S QG                                |
| Supplier#000009554 | e8De3zk2cRdvxQihN7IArownHTJFTgujVthQbd    |
| Supplier#000009640 | YVi8zEFPwnWxDBHbAz                        |
| Supplier#000009688 | ohakN polcEftV2GqPl                       |
| Supplier#000009695 | a5jfiVzGrKnNxceRfJXWjy0l7kSH,tsvLu3C      |
| Supplier#000009726 | kmuggXTMhmZZt4nPS1umvzBY                  |
| Supplier#000009764 | pcQj5Q07Rq5iEGSHa8BBpMGFB                 |
| Supplier#000009910 | yEcU23vFDVQ1,Hc4sSrUGdDW                  |
| Supplier#000009968 | q84VQRbUizJ                               |
+--------------------+-------------------------------------------+
187 rows in set (0.71 sec)
```

在如下自定义SQL并行中，使用了Semi-Join下推的并行方式，在 `max_parallel_degree=32` 的情况下，执行时间从2.59s减少到0.34s：

```
mysql> SELECT c1,d1 FROM t1 WHERE c1 IN ( SELECT t2.c1 FROM t2 WHERE t2.c1 = 'f'      OR t2
.c2 < 'y' ) AND t1.c1 AND d1 > '1900-1-1' LIKE "R1%" ORDER BY t1.c1 DESC, t1.d1 DESC;
Empty set, 1024 warnings (0.34 sec)
mysql> SET max_parallel_degree=0;
Query OK, 0 rows affected (0.00 sec)
mysql>  SELECT c1,d1 FROM t1 WHERE c1 IN ( SELECT t2.c1 FROM t2 WHERE t2.c1 = 'f'      OR t
2.c2 < 'y' ) AND t1.c1 AND d1 > '1900-1-1' LIKE "R1%" ORDER BY t1.c1 DESC, t1.d1 DESC;
Empty set, 65535 warnings (2.69 sec)
mysql> EXPLAIN SELECT c1,d1 FROM t1 WHERE c1 IN ( SELECT t2.c1 FROM t2 WHERE t2.c1 = 'f'
OR t2.c2 < 'y' ) AND t1.c1 AND d1 > '1900-1-1' LIKE "R1%" ORDER BY t1.c1 DESC, t1.d1 DESC;
+----+--------------+-------------+------------+--------+---------------+------------+-----
----+----------+--------+----------+-----------------------------------------------------------
--+
| id | select_type  | table       | partitions | type   | possible_keys | key        | key_
len | ref      | rows   | filtered | Extra
|
+----+--------------+-------------+------------+--------+---------------+------------+-----
----+----------+--------+----------+-----------------------------------------------------------
--+
|  1 | SIMPLE       | <gather1>   | NULL       | ALL    | NULL          | NULL       | NULL
| NULL     |  33464 |   100.00 | Merge sort                                                |
|  1 | SIMPLE       | t1          | NULL       | ALL    | NULL          | NULL       | NULL
| NULL     |  62802 |    30.00 | Parallel scan (32 workers); Using where; Using filesort |
|  1 | SIMPLE       | <subquery2> | NULL       | eq_ref | <auto_key>    | <auto_key> | 103
| sj.t1.c1 |      1 |   100.00 | NULL                                                      |
|  2 | MATERIALIZED | t2          | p0,p1      | ALL    | c1,c2         | NULL       | NULL
| NULL     | 100401 |    33.33 | Using where                                               |
+----+--------------+-------------+------------+--------+---------------+------------+-----
-
```

4.3.6. ROLLUP性能增强

您可以使用 `ROLLUP` 语法实现只借助一条查询语句，就计算出数据在不同维度上划分组之后得到的统计结果以及所有数据的总体值。本文将介绍如何使用 `ROLLUP` 语法。

前提条件

PolarDB集群版本需为PolarDB MySQL引擎 8.0且Revision version为8.0.1.1.0或以上，您可以参见查询版本号确认集群版本。

语法

`ROLLUP` 语法可以看作是 `GROUP BY` 语法的拓展，您只需要在原本的 `GROUP BY` 列之后加上 `WITH ROLLUP` 即可，例如：

```
SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
```

除了产生按照GROUP BY所指定的列聚合产生的结果外，`ROLLUP`还会从右至左依次计算更高层次的聚合结果，直至所有数据的聚合。例如上述语句就会先按照`GROUP BY year, country, product`计算总收益，再按照`GROUP BY year, country`以及`GROUP BY year`计算总收益，最后按照不带任何`GROUP BY`条件计算整个sales表的总收益。

使用`ROLLUP`语法主要有如下优势：

- 方便对数据进行多维度的统计分析，简化原本对不同维度各自进行SQL查询的编程复杂度。
- 实现更快更高效的查询处理。
- `ROLLUP`可以将所有的聚合工作转移到服务端，只用读一次数据就能够完成原本多次查询才能完成的统计工作，减轻了客户端的处理负载和网络流量。

并行查询ROLLUP增强性能测试

PolarDB还实现了针对`ROLLUP`语法的并行查询（Parallel Query）增强，支持多个线程并发计算聚合结果并输出汇总后的结果，大大提升语句执行效率。

使用TPCH进行测试，以Q1为例，在原始的语句上加入`ROLLUP`语法：

> **说明** 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。

```
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date_sub('1998-12-01', interval ':1' day)
group by
    l_returnflag,
    l_linestatus
with rollup
order by
    l_returnflag,
    l_linestatus;
```

- 不开启并行查询时，耗时318.73秒。

```
root@localhost:dbt3 8.0.13-rds-dev> select
    ->     l_returnflag,
    ->     l_linestatus,
    ->     sum(l_quantity) as sum_qty,
    ->     sum(l_extendedprice) as sum_base_price,
    ->     sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    ->     sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    ->     avg(l_quantity) as avg_qty,
    ->     avg(l_extendedprice) as avg_price,
    ->     avg(l_discount) as avg_disc,
    ->     count(*) as count_order
    -> from
    ->     lineitem
    -> where
    ->     l_shipdate <= date_sub('1998-12-01', interval ':1' day)
    -> group by
    ->     l_returnflag,
    ->     l_linestatus
    -> with rollup
    -> order by
    ->     l_returnflag,
    ->     l_linestatus;
+--------------+--------------+---------------+-------------------+-------------------+--------------------+----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price    | sum_disc_price    | sum_charge         | avg_qty  | avg_price   | avg_disc | count_order |
+--------------+--------------+---------------+-------------------+-------------------+--------------------+----------+-------------+----------+-------------+
| NULL         | NULL         | 1526902118.00 | 2289567239008.24  | 2175081814581.5635 | 2262103960452.919838 | 25.501098 | 38238.521048 | 0.050000 |    59875936 |
| A            | NULL         |  376833718.00 |  565037383437.68  |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| A            | F            |  376833718.00 |  565037383437.68  |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| N            | NULL         |  773043634.00 | 1159154050514.15  | 1101191254365.0758 | 1145250897418.143233 | 25.497847 | 38233.200251 | 0.050000 |    30317997 |
| N            | F            |    9830917.00 |   14735826224.79  |   13998714160.5500 |   14559295139.030004 | 25.516888 | 38247.950728 | 0.049977 |      385271 |
| N            | O            |  763212717.00 | 1144418224289.36  | 1087192540204.5258 | 1130691602279.113229 | 25.497601 | 38233.010394 | 0.050000 |    29932726 |
| R            | NULL         |  377024766.00 |  565375805056.41  |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
| R            | F            |  377024766.00 |  565375805056.41  |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
+--------------+--------------+---------------+-------------------+-------------------+--------------------+----------+-------------+----------+-------------+
8 rows in set, 3 warnings (5 min 18.73 sec)
```

- 开启并行查询后，耗时22.30秒，语句执行效率提升14倍以上。

```
root@localhost:dbt3 8.0.13-rds-dev> select /*+ SET_VAR(max_parallel_degree=32) */
    ->     l_returnflag,
    ->     l_linestatus,
    ->     sum(l_quantity) as sum_qty,
    ->     sum(l_extendedprice) as sum_base_price,
    ->     sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    ->     sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    ->     avg(l_quantity) as avg_qty,
    ->     avg(l_extendedprice) as avg_price,
    ->     avg(l_discount) as avg_disc,
    ->     count(*) as count_order
    -> from
    ->     lineitem
    -> where
    ->     l_shipdate <= date_sub('1998-12-01', interval ':1' day)
    -> group by
    ->     l_returnflag,
    ->     l_linestatus
    -> with rollup
    -> order by
    ->     l_returnflag,
    ->     l_linestatus;
+--------------+--------------+---------------+-------------------+-------------------+--------------------+----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price    | sum_disc_price    | sum_charge         | avg_qty  | avg_price   | avg_disc | count_order |
+--------------+--------------+---------------+-------------------+-------------------+--------------------+----------+-------------+----------+-------------+
| NULL         | NULL         | 1526902118.00 | 2289567239008.24  | 2175081814581.5635 | 2262103960452.919838 | 25.501098 | 38238.521048 | 0.050000 |    59875936 |
| A            | NULL         |  376833718.00 |  565037383437.68  |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| A            | F            |  376833718.00 |  565037383437.68  |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| N            | NULL         |  773043634.00 | 1159154050514.15  | 1101191254365.0758 | 1145250897418.143233 | 25.497847 | 38233.200251 | 0.050000 |    30317997 |
| N            | F            |    9830917.00 |   14735826224.79  |   13998714160.5500 |   14559295139.030004 | 25.516888 | 38247.950728 | 0.049977 |      385271 |
| N            | O            |  763212717.00 | 1144418224289.36  | 1087192540204.5258 | 1130691602279.113229 | 25.497601 | 38233.010394 | 0.050000 |    29932726 |
| R            | NULL         |  377024766.00 |  565375805056.41  |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
| R            | F            |  377024766.00 |  565375805056.41  |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
+--------------+--------------+---------------+-------------------+-------------------+--------------------+----------+-------------+----------+-------------+
8 rows in set, 3 warnings (22.30 sec)
```

4.3.7. 通过Hint控制

Parallel Hints可以指定优化器是否选择并行执行，还支持指定并行度以及需要并行的表。PolarDB目前支持在并行查询中使用 `PARALLEL` 和 `NO_PARALLEL` 两种Hints。

开启或关闭并行查询

- 开启并行查询

  您可以使用如下任意一种方式开启并行查询：

```
SELECT /*+PARALLEL(x)*/ ... FROM ...;   -- x >0
```

```
SELECT /*+ SET_VAR(max_parallel_degree=n) */  *  FROM ...   // n > 0
```

- 关闭并行查询

  您可以使用如下任意一种方式关闭并行查询：

```
SELECT /*+NO_PARALLEL()*/ ... FROM ...;
```

```
SELECT /*+ SET_VAR(max_parallel_degree=0) */  *  FROM ...
```

Hint高级用法

并行查询提供了PARALLEL和NO_PARALLEL两种Hint，其中：

- 通过PARALLEL Hint可以强制查询并行执行，同时可以指定并行度和并行扫描的表。语法如下所示：

```
/*+ PARALLEL [( [query_block] [table_name]  [degree] )] */
```

- 通过NO_PARALLEL Hint可以强制查询串行执行，或者指定不选择某些表作为并行扫描的表。

```
/*+ NO_PARALLEL [( [query_block] [table_name][, table_name] )] */
```

其中参数说明如下所示。

参数	说明
query_block	应用Hint的query block名称。
table_name	应用Hint的表名称。
degree	并行度。

示例：

```
SELECT /*+PARALLEL()*/ * FROM t1, t2;
-- 当表记录数小于records_threshold_for_parallelism设置的行数 （ 默认10000行）时，会强制并行，
-- 并行度用系统默认max_parallel_degree， 如果max_parallel_degree > 0，
-- 则打开并行，如果max_parallel_degree等于0时，依旧时关闭并行。
SELECT /*+PARALLEL(8)*/ * FROM t1, t2;
-- 强制并行度8并行执行，
-- 当表记录数小于records_threshold_for_parallelism设置的行数 （ 默认10000行）时，会强制并行，
-- 并行度设置max_parallel_degree为8。
SELECT /*+ SET_VAR(max_parallel_degree=8) */  *  FROM ...
-- 设置并行度max_parallel_degree为8，
-- 当表记录数小于records_threshold_for_parallelism设置的行数（如20000行）时，会自动关闭并行。
SELECT /*+PARALLEL(t1)*/ * FROM t1, t2;
-- 选择t1表并行，对t1表执行/*+PARALLEL()*/ 语法
SELECT /*+PARALLEL(t1 8)*/ * FROM t1, t2;
-- 强制并行度8且选择t1表并行执行， 对t1表执行/*+PARALLEL(8)*/语法
SELECT /*+PARALLEL(@subq1)*/ SUM(t.a) FROM t WHERE t.a =
  (SELECT /*+QB_NAME(subq1)*/ SUM(t1.a) FROM t1);
--强制subquery并行执行， 并行度用系统默认max_parallel_degree，
-- 如果max_parallel_degree > 0，则打开并行，max_parallel_degree等于0时，依旧时关闭并行
SELECT /*+PARALLEL(@subq1 8)*/ SUM(t.a) FROM t WHERE t.a =
  (SELECT /*+QB_NAME(subq1)*/ SUM(t1.a) FROM t1);
--强制subquery并行执行， 并行度设置max_parallel_degree为8
SELECT SUM(t.a) FROM t WHERE t.a =
  (SELECT /*+PARALLEL()*/ SUM(t1.a) FROM t1);
--强制subquery并行执行，
-- 并行度用系统默认max_parallel_degree，
-- 如果max_parallel_degree > 0，则打开并行，max_parallel_degree等于0时，依旧时关闭并行
SELECT SUM(t.a) FROM t WHERE t.a =
  (SELECT /*+PARALLEL(8)*/ SUM(t1.a) FROM t1);
--强制subquery并行执行， 设置并行度max_parallel_degree为8
SELECT /*+NO_PARALLEL()*/ * FROM t1, t2;
-- 禁止并行执行
SELECT /*+NO_PARALLEL(t1)*/ * FROM t1, t2;
-- 只对t1表禁止并行， 当系统打开并行时， 有可能对t2进行并行扫描，并行执行
SELECT /*+NO_PARALLEL(t1, t2)*/ * FROM t1, t2;
-- 同时对t1和t2表禁止并行
SELECT /*+NO_PARALLEL(@subq1)*/ SUM(t.a) FROM t WHERE t.a =
  (SELECT /*+QB_NAME(subq1)*/ SUM(t1.a) FROM t1);
--禁止subquery 并行执行
SELECT SUM(t.a) FROM t WHERE t.a =
  (SELECT /*+NO_PARALLEL()*/ SUM(t1.a) FROM t1);
 --禁止subquery 并行执行
```

> **说明** 对于不支持并行的查询或者并行扫描的表，PARALLEL Hint不生效。

并行子查询

并行子查询的选择方式（并行子查询详细信息请参见子查询支持）也可以通过Hint来进行控制，语法及说明如下：

```
/*+ PQ_PUSHDOWN [( [query_block])] */ 对应的子查询会选择push down的并行子查询执行策略
/*+ NO_PQ_PUSHDOWN [( [query_block])] */ 对应的子查询会选择shared access的并行子查询执行策略
```

示例：

```
#子查询选择push down并行策略
EXPLAIN SELECT /*+ PQ_PUSHDOWN(@qb1) */ * FROM t2 WHERE t2.a =
                  (SELECT /*+ qb_name(qb1) */ a FROM t1);
#子查询选择shared access并行策略
EXPLAIN SELECT /*+ NO_PQ_PUSHDOWN(@qb1) */ * FROM t2 WHERE t2.a =
                  (SELECT /*+ qb_name(qb1) */ a FROM t1);
#不加query block进行控制
EXPLAIN SELECT * FROM t2 WHERE t2.a =
                  (SELECT /*+ NO_PQ_PUSHDOWN() */ a FROM t1);
```

4.3.8. 并行查询使用限制与串行执行结果兼容问题

本文为您介绍并行查询的使用限制以及与串行执行结果可能不兼容的地方，帮助您正确使用并行查询功能。

并行查询的使用限制

PolarDB会持续迭代并行查询的能力，目前以下情况暂时无法享受并行查询带来的性能提升：

- 查询系统表或非Innodb表。
- 使用全文索引的查询。
- 存储过程Procedures。
- Recursive CTE。
- Windows Functions（函数本身不能并行计算，但整个查询可以并行）。
- GIS （函数本身不能并行计算，但整个查询可以并行）。
- Index Merge。
- 串行化隔离级别事务内的查询语句。

与串行执行结果可能不兼容的地方

- 错误提示次数可能会增多

  串行执行中出现错误提示的查询，在并行执行的情况下，可能每个工作线程都会提示错误，导致总体错误提示数增多。

- 精度问题

  并行查询的执行过程中，可能会出现比串行执行多出中间结果的存储，如果中间结果是浮点型，可能会导致浮点部分精度差别，导致最终结果有细微的差别。

- 网络包或者中间结果长度超出max_allowed_packet允许的最大长度

  并行查询的执行过程中，相比串行执行可能会多出中间结果。如果中间结果的长度超出了max_allowed_packet定义的最大长度，可能出现错误提示，可以通过增加max_allowed_packet参数的值来解决。如何修改参数请参见设置集群参数。

- 结果集顺序差别

当并行执行未加ORDER BY关键字的 `SELECT ... LIMIT n` 语句时，返回的结果集可能与执行顺序不一致。由于有多个Worker同时执行，每次执行时Worker的执行速度是不确定的，当Leader得到足够的数据后，就会返回结果，因此返回的结果集可能与执行顺序不一致。

- 加了行锁的数据记录数增多

  当并行执行 `SELECT ... FROM ... FOR SHARE` 语句时，InnoDB会将访问到的每一行数据都加锁，因此加了行锁的记录数可能会比非并行执行的情况下要多，这属于正常现象。

4.3.9. 并行查询使用示例

本文以TPC-H为例，为您介绍并行查询使用示例。

> **说明** 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。

- GROUP BY和ORDER BY支持
- AGGREGATE函数支持（SUM/AVG/COUNT）
- JOIN支持
- BETWEEN函数和IN函数支持
- LIMIT支持
- INTERVAL函数支持
- CASE WHEN支持
- LIKE支持

测试设计

- 测试用数据量：100 GB（Scalar Factor = 100）
- 测试用PolarDB MySQL引擎8.0版本的集群：节点规格为88核710 GB，在主节点上进行测试。

GROUP BY和ORDER BY支持

原始SQL语句如下所示：

```
SELECT   l_returnflag,
         l_linestatus,
         Sum(l_quantity)                                       AS sum_qty,
         Sum(l_extendedprice)                                  AS sum_base_price,
         Sum(l_extendedprice * (1 - l_discount))               AS sum_disc_price,
         Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
         Avg(l_quantity)                                       AS avg_qty,
         Avg(l_extendedprice)                                  AS avg_price,
         Avg(l_discount)                                       AS avg_disc,
         Count(*)                                              AS count_order
FROM     lineitem
WHERE    l_shipdate <= date '1998-12-01' - INTERVAL '93' day
GROUP BY l_returnflag,
         l_linestatus
ORDER BY l_returnflag,
         l_linestatus ;
```

- 未开启并行查询，耗时1563.32秒。

```
mysql> SELECT  /*+ SET_VAR(max_parallel_degree=0) */ l_returnflag,
    ->          l_linestatus,
    ->          Sum(l_quantity)                                       AS sum_qty,
    ->          Sum(l_extendedprice)                                  AS sum_base_price,
    ->          Sum(l_extendedprice * (1 - l_discount))               AS sum_disc_price,
    ->          Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    ->          Avg(l_quantity)                                       AS avg_qty,
    ->          Avg(l_extendedprice)                                  AS avg_price,
    ->          Avg(l_discount)                                       AS avg_disc,
    ->          Count(*)                                              AS count_order
    -> FROM     lineitem
    -> WHERE    l_shipdate <= date '1998-12-01' - INTERVAL '93' day
    -> GROUP BY l_returnflag,
    ->          l_linestatus
    -> ORDER BY l_returnflag,
    ->          l_linestatus ;
^@+--------------+-------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price   | sum_disc_price     | sum_charge           | avg_qty  | avg_price   | avg_disc | count_order |
+--------------+-------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| A            | F           | 3775127758.00 |  5660776097194.45 |  5377736398183.9374 |  5592847429515.927026 | 25.499370 | 38236.116984 | 0.050002 |   148047881 |
| N            | F           |   98553062.00 |   147771098385.98 |   140384965965.0348 |   145999793032.775829 | 25.501557 | 38237.199389 | 0.049985 |     3864590 |
| N            | O           | 7421794698.00 | 11128967412861.24 | 10572526740167.5066 | 10995437028892.436612 | 25.500030 | 38237.248190 | 0.049998 |   291050427 |
| R            | F           | 3775724970.00 |  5661603032745.34 |  5378513563915.4097 |  5593662252666.916161 | 25.500066 | 38236.697258 | 0.050001 |   148067261 |
+--------------+-------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
4 rows in set (26 min 3.32 sec)
```

- 开启并行查询后，耗时只用49.65秒，提升31.48 倍。

```
mysql> SELECT  /*+ SET_VAR(max_parallel_degree=32) */ l_returnflag,
    ->          l_linestatus,
    ->          Sum(l_quantity)                                       AS sum_qty,
    ->          Sum(l_extendedprice)                                  AS sum_base_price,
    ->          Sum(l_extendedprice * (1 - l_discount))               AS sum_disc_price,
    ->          Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    ->          Avg(l_quantity)                                       AS avg_qty,
    ->          Avg(l_extendedprice)                                  AS avg_price,
    ->          Avg(l_discount)                                       AS avg_disc,
    ->          Count(*)                                              AS count_order
    -> FROM     lineitem
    -> WHERE    l_shipdate <= date '1998-12-01' - INTERVAL '93' day
    -> GROUP BY l_returnflag,
    ->          l_linestatus
    -> ORDER BY l_returnflag,
    ->          l_linestatus ;
+--------------+-------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price   | sum_disc_price     | sum_charge           | avg_qty  | avg_price   | avg_disc | count_order |
+--------------+-------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| A            | F           | 3775127758.00 |  5660776097194.45 |  5377736398183.9374 |  5592847429515.927026 | 25.499370 | 38236.116984 | 0.050002 |   148047881 |
| N            | F           |   98553062.00 |   147771098385.98 |   140384965965.0348 |   145999793032.775829 | 25.501557 | 38237.199389 | 0.049985 |     3864590 |
| N            | O           | 7421794698.00 | 11128967412861.24 | 10572526740167.5066 | 10995437028892.436612 | 25.500030 | 38237.248190 | 0.049998 |   291050427 |
| R            | F           | 3775724970.00 |  5661603032745.34 |  5378513563915.4097 |  5593662252666.916161 | 25.500066 | 38236.697258 | 0.050001 |   148067261 |
+--------------+-------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
4 rows in set (49.65 sec)
```

AGGREGATE函数支持（SUM/AVG/COUNT）

原始SQL语句，如下所示：

```
SELECT   l_returnflag,
         l_linestatus,
         Sum(l_quantity)                                       AS sum_qty,
         Sum(l_extendedprice)                                  AS sum_base_price,
         Sum(l_extendedprice * (1 - l_discount))               AS sum_disc_price,
         Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
         Avg(l_quantity)                                       AS avg_qty,
         Avg(l_extendedprice)                                  AS avg_price,
         Avg(l_discount)                                       AS avg_disc,
         Count(*)                                              AS count_order
FROM     lineitem
WHERE    l_shipdate <= date '1998-12-01' - INTERVAL '93' day
GROUP BY l_returnflag,
         l_linestatus
ORDER BY l_returnflag,
         l_linestatus ;
```

- 未开启并行查询，耗时1563.32秒。

```
mysql> SELECT  /*+ SET_VAR(max_parallel_degree=0) */ l_returnflag,
    ->         l_linestatus,
    ->         Sum(l_quantity)                                    AS sum_qty,
    ->         Sum(l_extendedprice)                               AS sum_base_price,
    ->         Sum(l_extendedprice * (1 - l_discount))            AS sum_disc_price,
    ->         Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    ->         Avg(l_quantity)                                    AS avg_qty,
    ->         Avg(l_extendedprice)                               AS avg_price,
    ->         Avg(l_discount)                                    AS avg_disc,
    ->         Count(*)                                           AS count_order
    -> FROM    lineitem
    -> WHERE   l_shipdate <= date '1998-12-01' - INTERVAL '93' day
    -> GROUP BY l_returnflag,
    ->         l_linestatus
    -> ORDER BY l_returnflag,
    ->         l_linestatus ;
^@+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price   | sum_disc_price     | sum_charge           | avg_qty  | avg_price   | avg_disc | count_order |
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| A            | F            | 3775127758.00 |  5660776097194.45 |  5377736398183.9374 |  5592847429515.927026 | 25.499370 | 38236.116984 | 0.050002 |   148047881 |
| N            | F            |   98553062.00 |   147771098385.98 |   140384965965.0348 |   145999793032.775829 | 25.501557 | 38237.199389 | 0.049985 |     3864590 |
| N            | O            | 7421794698.00 | 11128967412861.24 | 10572526740167.5066 | 10995437028892.436612 | 25.500030 | 38237.248190 | 0.049998 |   291050427 |
| R            | F            | 3775724970.00 |  5661603032745.34 |  5378513563915.4097 |  5593662252666.916161 | 25.500066 | 38236.697258 | 0.050001 |   148067261 |
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
4 rows in set (26 min 3.32 sec)
```

- 开启并行查询后，耗时只用49.65秒，提升31.48 倍。

```
mysql> SELECT  /*+ SET_VAR(max_parallel_degree=32) */ l_returnflag,
    ->         l_linestatus,
    ->         Sum(l_quantity)                                    AS sum_qty,
    ->         Sum(l_extendedprice)                               AS sum_base_price,
    ->         Sum(l_extendedprice * (1 - l_discount))            AS sum_disc_price,
    ->         Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    ->         Avg(l_quantity)                                    AS avg_qty,
    ->         Avg(l_extendedprice)                               AS avg_price,
    ->         Avg(l_discount)                                    AS avg_disc,
    ->         Count(*)                                           AS count_order
    -> FROM    lineitem
    -> WHERE   l_shipdate <= date '1998-12-01' - INTERVAL '93' day
    -> GROUP BY l_returnflag,
    ->         l_linestatus
    -> ORDER BY l_returnflag,
    ->         l_linestatus ;
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price   | sum_disc_price     | sum_charge           | avg_qty  | avg_price   | avg_disc | count_order |
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| A            | F            | 3775127758.00 |  5660776097194.45 |  5377736398183.9374 |  5592847429515.927026 | 25.499370 | 38236.116984 | 0.050002 |   148047881 |
| N            | F            |   98553062.00 |   147771098385.98 |   140384965965.0348 |   145999793032.775829 | 25.501557 | 38237.199389 | 0.049985 |     3864590 |
| N            | O            | 7421794698.00 | 11128967412861.24 | 10572526740167.5066 | 10995437028892.436612 | 25.500030 | 38237.248190 | 0.049998 |   291050427 |
| R            | F            | 3775724970.00 |  5661603032745.34 |  5378513563915.4097 |  5593662252666.916161 | 25.500066 | 38236.697258 | 0.050001 |   148067261 |
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
4 rows in set (49.65 sec)
```

JOIN支持

原始SQL语句，如下所示：

```
select sum(l_extendedprice* (1 - l_discount)) as revenue
from   lineitem,   part
where ( p_partkey = l_partkey and p_brand = 'Brand#12'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 6 and l_quantity <= 6 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON' )
    or ( p_partkey = l_partkey and p_brand = 'Brand#13'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 10 and l_quantity <= 10 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON' )
    or ( p_partkey = l_partkey and p_brand = 'Brand#24'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 21 and l_quantity <= 21 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON' );
```

- 未开启并行查询，耗时21.73秒。

```
mysql> select /*+ SET_VAR(max_parallel_degree=0) */ sum(l_extendedprice* (1 - l_discount)) as revenue
    -> from   lineitem,   part
    -> where ( p_partkey = l_partkey and p_brand = 'Brand#12'
    ->         and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    ->         and l_quantity >= 6 and l_quantity <= 6 + 10
    ->         and p_size between 1 and 5
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#13'
    ->         and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    ->         and l_quantity >= 10 and l_quantity <= 10 + 10
    ->         and p_size between 1 and 10
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#24'
    ->         and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    ->         and l_quantity >= 21 and l_quantity <= 21 + 10
    ->         and p_size between 1 and 15
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' );

+----------------+
| revenue        |
+----------------+
| 317693789.3851 |
+----------------+
1 row in set (21.73 sec)
```

- 开启并行查询后，耗时1.37秒，提升15.86倍。

```
mysql> select /*+ SET_VAR(max_parallel_degree=32) */ sum(l_extendedprice* (1 - l_discount)) as revenue
    -> from   lineitem,   part
    -> where ( p_partkey = l_partkey and p_brand = 'Brand#12'
    ->         and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    ->         and l_quantity >= 6 and l_quantity <= 6 + 10
    ->         and p_size between 1 and 5
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#13'
    ->         and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    ->         and l_quantity >= 10 and l_quantity <= 10 + 10
    ->         and p_size between 1 and 10
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#24'
    ->         and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    ->         and l_quantity >= 21 and l_quantity <= 21 + 10
    ->         and p_size between 1 and 15
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' );
+----------------+
| revenue        |
+----------------+
| 317693789.3851 |
+----------------+
1 row in set (1.37 sec)
```

BETWEEN函数和IN函数支持

原始SQL语句，如下所示：

```
select sum(l_extendedprice* (1 - l_discount)) as revenue
from   lineitem,   part
where ( p_partkey = l_partkey and p_brand = 'Brand#12'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 6 and l_quantity <= 6 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON' )
    or ( p_partkey = l_partkey and p_brand = 'Brand#13'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 10 and l_quantity <= 10 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON' )
    or ( p_partkey = l_partkey and p_brand = 'Brand#24'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 21 and l_quantity <= 21 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON' );
```

- 未开启并行查询，耗时21.73秒。

```
mysql> select /*+ SET_VAR(max_parallel_degree=0) */ sum(l_extendedprice* (1 - l_discount)) as revenue
    -> from   lineitem,   part
    -> where ( p_partkey = l_partkey and p_brand = 'Brand#12'
    ->         and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    ->         and l_quantity >= 6 and l_quantity <= 6 + 10
    ->         and p_size between 1 and 5
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#13'
    ->         and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    ->         and l_quantity >= 10 and l_quantity <= 10 + 10
    ->         and p_size between 1 and 10
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#24'
    ->         and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    ->         and l_quantity >= 21 and l_quantity <= 21 + 10
    ->         and p_size between 1 and 15
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' );

+----------------+
| revenue        |
+----------------+
| 317693789.3851 |
+----------------+
1 row in set (21.73 sec)
```

- 开启并行查询后，耗时1.37秒，提升15.86倍。

```
mysql> select /*+ SET_VAR(max_parallel_degree=32) */ sum(l_extendedprice* (1 - l_discount)) as revenue
    -> from   lineitem,   part
    -> where ( p_partkey = l_partkey and p_brand = 'Brand#12'
    ->         and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    ->         and l_quantity >= 6 and l_quantity <= 6 + 10
    ->         and p_size between 1 and 5
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#13'
    ->         and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    ->         and l_quantity >= 10 and l_quantity <= 10 + 10
    ->         and p_size between 1 and 10
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' )
    ->     or ( p_partkey = l_partkey and p_brand = 'Brand#24'
    ->         and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    ->         and l_quantity >= 21 and l_quantity <= 21 + 10
    ->         and p_size between 1 and 15
    ->         and l_shipmode in ('AIR', 'AIR REG')
    ->         and l_shipinstruct = 'DELIVER IN PERSON' );
+---------------+
| revenue       |
+---------------+
| 317693789.3851 |
+---------------+
1 row in set (1.37 sec)
```

LIMIT支持

原始SQL语句，如下所示：

```
select l_shipmode, sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = '2-HIGH'
then 1
    else 0
end) as high_line_count, sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <>
'2-HIGH' then 1
else 0
end) as low_line_count
from   orders,   lineitem
where o_orderkey = l_orderkey
and l_shipmode in ('MAIL', 'TRUCK')
and l_commitdate < l_receiptdate
and l_shipdate < l_commitdate
and l_receiptdate >= date '1996-01-01'
and l_receiptdate < date '1996-01-01' + interval '1' year
group by l_shipmode
order by l_shipmode limit 10;
```

- 未开启并行查询，耗时339.22 秒。

```
mysql> select /*+ SET_VAR(max_parallel_degree=0) */
    l_shipmode, sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = '2-HIGH' then 1
    else 0
end) as high_line_count, sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH' then 1
else 0
end) as low_line_count
from   orders,   lineitem
where o_orderkey = l_orderkey
and l_shipmode in ('MAIL', 'TRUCK')
and l_commitdate < l_receiptdate
and l_shipdate < l_commitdate
and l_receiptdate >= date '1996-01-01'
and l_receiptdate < date '1996-01-01' + interval '1' year
group by l_shipmode
order by l_shipmode limit 10;
+------------+-----------------+----------------+
| l_shipmode | high_line_count | low_line_count |
+------------+-----------------+----------------+
| MAIL       | 625339          | 937117         |
| TRUCK      | 625580          | 937993         |
+------------+-----------------+----------------+
2 rows in set (339.22 sec)
```

- 开启并行查询后，耗时29.31秒，提升11.57倍。

```
mysql> select /*+ SET_VAR(max_parallel_degree=32) */
    l_shipmode, sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = '2-HIGH' then 1
    else 0
end) as high_line_count, sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH' then 1
else 0
end) as low_line_count
from   orders,   lineitem
where o_orderkey = l_orderkey
and l_shipmode in ('MAIL', 'TRUCK')
and l_commitdate < l_receiptdate
and l_shipdate < l_commitdate
and l_receiptdate >= date '1996-01-01'
and l_receiptdate < date '1996-01-01' + interval '1' year
group by l_shipmode
order by l_shipmode limit 10;
+------------+-----------------+----------------+
| l_shipmode | high_line_count | low_line_count |
+------------+-----------------+----------------+
| MAIL       | 625339          | 937117         |
| TRUCK      | 625580          | 937993         |
+------------+-----------------+----------------+
2 rows in set (29.31 sec)
```

INTERVAL函数支持

原始SQL语句，如下所示：

```
select
    100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount)
    else 0
end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from   lineitem,   part
where l_partkey = p_partkey
and l_shipdate >= date '1996-01-01'
and l_shipdate < date '1996-01-01' + interval '1' month limit 10;
```

- 未开启并行查询，耗时220.87秒。

```
mysql> select /*+ SET_VAR(max_parallel_degree=0) */
    ->     100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount)
    ->     else 0
    -> end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
    -> from   lineitem,   part
    -> where l_partkey = p_partkey
    -> and l_shipdate >= date '1996-01-01'
    -> and l_shipdate < date '1996-01-01' + interval '1' month limit 10;

^@+---------------+
| promo_revenue |
+---------------+
| 16.6415897388 |
+---------------+
1 row in set (3 min 40.87 sec)
```

- 开启并行查询后，耗时7.75秒，提升28.5倍。

```
mysql>
mysql> select /*+ SET_VAR(max_parallel_degree=32) */
    ->     100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount)
    ->     else 0
    -> end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
    -> from   lineitem,   part
    -> where l_partkey = p_partkey
    -> and l_shipdate >= date '1996-01-01'
    -> and l_shipdate < date '1996-01-01' + interval '1' month limit 10;
+---------------+
| promo_revenue |
+---------------+
| 16.6415897388 |
+---------------+
1 row in set (7.75 sec)
```

CASE WHEN支持

原始SQL语句，如下所示：

```
select
    100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount)
    else 0
end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from   lineitem,   part
where l_partkey = p_partkey
and l_shipdate >= date '1996-01-01'
and l_shipdate < date '1996-01-01' + interval '1' month limit 10;
```

- 未开启并行查询，耗时220.87秒。

```
mysql> select /*+ SET_VAR(max_parallel_degree=0) */
    ->     100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount)
    ->     else 0
    -> end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
    -> from   lineitem,   part
    -> where l_partkey = p_partkey
    -> and l_shipdate >= date '1996-01-01'
    -> and l_shipdate < date '1996-01-01' + interval '1' month limit 10;

^@+---------------+
| promo_revenue |
+---------------+
| 16.6415897388 |
+---------------+
1 row in set (3 min 40.87 sec)
```

- 开启并行查询后，耗时7.75秒，提升28.5倍。

```
mysql>
mysql> select /*+ SET_VAR(max_parallel_degree=32) */
    ->     100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount)
    ->     else 0
    -> end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
    -> from   lineitem,   part
    -> where l_partkey = p_partkey
    -> and l_shipdate >= date '1996-01-01'
    -> and l_shipdate < date '1996-01-01' + interval '1' month limit 10;
+---------------+
| promo_revenue |
+---------------+
| 16.6415897388 |
+---------------+
1 row in set (7.75 sec)
```

LIKE支持

原始SQL语句，如下所示：

```
select s_name, s_address from
 supplier,  nation where
s_suppkey in
    ( select ps_suppkey from  partsupp where
             ps_partkey in ( select p_partkey from  part where p_name like 'dark%')
            and ps_availqty>(select 0.0005 * sum(l_quantity) as col1
     from   lineitem,   partsupp
     where l_partkey = ps_partkey and l_suppkey = ps_suppkey
     and l_shipdate >= date '1993-01-01' and l_shipdate < date '1993-01-01' + interval '1'
year)
    )
and s_nationkey = n_nationkey and n_name = 'JORDAN'
order by s_name limit 10;
```

- 未开启并行查询，耗时427.46秒。

```
mysql>
mysql> select /*+ SET_VAR(max_parallel_degree=0) */ s_name, s_address from
    ->  supplier,  nation where
    -> s_suppkey in
    ->     ( select ps_suppkey from  partsupp where
    ->              ps_partkey in ( select p_partkey from  part where p_name like 'dark%')
    ->             and ps_availqty>(select 0.0005 * sum(l_quantity) as col1
    ->      from   lineitem,   partsupp
    ->      where l_partkey = ps_partkey and l_suppkey = ps_suppkey
    ->      and l_shipdate >= date '1993-01-01' and l_shipdate < date '1993-01-01' + interval '1' year)
    ->     )
    -> and s_nationkey = n_nationkey and n_name = 'JORDAN'
    -> order by s_name limit 10;
ct p_partkey from  part where p_name like 'dark%')
            and ps_availqty>(select 0.0005 * sum(l_quantity) as col1
     from   lineitem,   partsupp
     where l_partkey = ps_partkey and l_suppkey = ps_suppkey
     and l_shipdate >= date '1993-01-01' and l_shipdate < date '1993-01-01' + interval '1' year)
    )
and s_nationkey = n_nationkey and n_name = 'JORDAN'
order by s_name limit 10;
^@^@^@^@^@^@Empty set (7 min 7.46 sec)
```

- 开启并行查询后，耗时33.72秒，提升12.68倍。

```
mysql> select /*+ SET_VAR(max_parallel_degree=32) */ s_name, s_address from
    ->  supplier,  nation where
    -> s_suppkey in
    ->     ( select ps_suppkey from  partsupp where
    ->              ps_partkey in ( select p_partkey from  part where p_name like 'dark%')
    ->             and ps_availqty>(select 0.0005 * sum(l_quantity) as col1
    ->      from   lineitem,   partsupp
    ->      where l_partkey = ps_partkey and l_suppkey = ps_suppkey
    ->      and l_shipdate >= date '1993-01-01' and l_shipdate < date '1993-01-01' + interval '1' year)
    ->     )
    -> and s_nationkey = n_nationkey and n_name = 'JORDAN'
    -> order by s_name limit 10;
^@Empty set (33.72 sec)
```

SUBQUERY支持

原始SQL语句，如下所示：

```
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 35
    and p_type like '%STEEL'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp,
            supplier,
            nation,
            region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'AMERICA'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey;
limit 100;
```

- 未开启并行查询，耗时16.23秒。

```
mysql> select /*+ SET_VAR(max_parallel_degree=0) */
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 35
    and p_type like '%STEEL'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp,
            supplier,
            nation,
            region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'AMERICA'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey
limit 100;
+-----------+--------------------+---------------+------------+----------------+-----------------------------------------+-----------------+
+-----------------------------------------------------------------------------------------------------------------------------------+
| s_acctbal | s_name             | n_name        | p_partkey | p_mfgr         | s_address                               | s_phone         | s_comment
|
+-----------+--------------------+---------------+------------+----------------+-----------------------------------------+-----------------+
+-----------------------------------------------------------------------------------------------------------------------------------+
| 9999.99   | Supplier#000627583 | CANADA        |    3127576 | Manufacture#4 | DN0XHzR9 1XXcAZtG4fux1Z                 | 13-841-451-5653 | uffily about the thin asymptotes--regular,busy
instructions behind the Tiresias'
| 9999.57   | Supplier#000389448 | PERU          |    5139442 | Manufacture#4 | gfhQW53f5lRhkpLjOOIUgsfZOXpl0hN.4Qw     | 27-224-704-7021 | tealthy,sly asymptotes according to the idle depths
```

- 开启并行查询后，耗时1.4秒，提升11.59倍。

```
mysql> select /*+ SET_VAR(max_parallel_degree=32) */
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 35
    and p_type like '%STEEL'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp,
            supplier,
            nation,
            region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'AMERICA'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey
limit 100;
+-----------+--------------------+---------------+------------+----------------+-----------------------------------------+-----------------+
+-----------------------------------------------------------------------------------------------------------------------------------+
| s_acctbal | s_name             | n_name        | p_partkey | p_mfgr         | s_address                               | s_phone         | s_comment
|
+-----------+--------------------+---------------+------------+----------------+-----------------------------------------+-----------------+
+-----------------------------------------------------------------------------------------------------------------------------------+
| 9999.99   | Supplier#000627583 | CANADA        |    3127576 | Manufacture#4 | DN0XHzR9 1XXcAZtG4fux1Z                 | 13-841-451-5653 | uffily about the thin asymptotes--regular,busy
```

GROUP BY WITH ROLLUP支持

GROUP BY WITH ROLLUP详细介绍请参见MySQL-with rollup函数运用和MySQL ROLLUP。

原始SQL语句，如下所示：

```
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date_sub('1998-12-01', interval ':1' day)
group by
    l_returnflag,
    l_linestatus
with rollup
order by
    l_returnflag,
    l_linestatus;
```

- 未开启并行查询，耗时318.73秒。

```
root@localhost:dbt3 8.0.13-rds-dev> select
    ->     l_returnflag,
    ->     l_linestatus,
    ->     sum(l_quantity) as sum_qty,
    ->     sum(l_extendedprice) as sum_base_price,
    ->     sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    ->     sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    ->     avg(l_quantity) as avg_qty,
    ->     avg(l_extendedprice) as avg_price,
    ->     avg(l_discount) as avg_disc,
    ->     count(*) as count_order
    -> from
    ->     lineitem
    -> where
    ->     l_shipdate <= date_sub('1998-12-01', interval ':1' day)
    -> group by
    ->     l_returnflag,
    ->     l_linestatus
    -> with rollup
    -> order by
    ->     l_returnflag,
    ->     l_linestatus;
+--------------+--------------+---------------+------------------+--------------------+----------------------+-----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price   | sum_disc_price     | sum_charge           | avg_qty   | avg_price   | avg_disc | count_order |
+--------------+--------------+---------------+------------------+--------------------+----------------------+-----------+-------------+----------+-------------+
| NULL         | NULL         | 1526902118.00 | 2289567239008.24 | 2175081814581.5635 | 2262103960452.919838 | 25.501098 | 38238.521048 | 0.050000 |    59875936 |
| A            | NULL         |  376833718.00 |  565037383437.68 |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| A            | F            |  376833718.00 |  565037383437.68 |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| N            | NULL         |  773043634.00 | 1159154050514.15 | 1101191254365.0758 | 1145250897418.143233 | 25.497847 | 38233.200251 | 0.050000 |    30317997 |
| N            | F            |    9830917.00 |   14735826224.79 |   13998714160.5500 |   14559295139.030004 | 25.516888 | 38247.950728 | 0.049977 |      385271 |
| N            | O            |  763212717.00 | 1144418224289.36 | 1087192540204.5258 | 1130691602279.113229 | 25.497601 | 38233.010394 | 0.050000 |    29932726 |
| R            | NULL         |  377024766.00 |  565375805056.41 |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
| R            | F            |  377024766.00 |  565375805056.41 |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
+--------------+--------------+---------------+------------------+--------------------+----------------------+-----------+-------------+----------+-------------+
8 rows in set, 3 warnings (5 min 18.73 sec)
```

- 开启并行查询后，耗时22.30秒，提升14.29倍。

```
root@localhost:dbt3 8.0.13-rds-dev> select /*+ SET_VAR(max_parallel_degree=32) */
    ->     l_returnflag,
    ->     l_linestatus,
    ->     sum(l_quantity) as sum_qty,
    ->     sum(l_extendedprice) as sum_base_price,
    ->     sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    ->     sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    ->     avg(l_quantity) as avg_qty,
    ->     avg(l_extendedprice) as avg_price,
    ->     avg(l_discount) as avg_disc,
    ->     count(*) as count_order
    -> from
    ->     lineitem
    -> where
    ->     l_shipdate <= date_sub('1998-12-01', interval ':1' day)
    -> group by
    ->     l_returnflag,
    ->     l_linestatus
    -> with rollup
    -> order by
    ->     l_returnflag,
    ->     l_linestatus;
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| l_returnflag | l_linestatus | sum_qty       | sum_base_price   | sum_disc_price     | sum_charge           | avg_qty  | avg_price   | avg_disc | count_order |
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
| NULL         | NULL         | 1526902118.00 | 2289567239008.24 | 2175081814581.5635 | 2262103960452.919838 | 25.501098 | 38238.521048 | 0.050000 |    59875936 |
| A            | NULL         |  376833718.00 |  565037383437.68 |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| A            | F            |  376833718.00 |  565037383437.68 |  536782889857.2197 |  558261263942.606353 | 25.500331 | 38236.069808 | 0.050005 |    14777601 |
| N            | NULL         |  773043634.00 | 1159154050514.15 | 1101191254365.0758 | 1145250897418.143233 | 25.497847 | 38233.200251 | 0.050000 |    30317997 |
| N            | F            |    9830917.00 |   14735826224.79 |   13998714160.5500 |   14559295139.030004 | 25.516888 | 38247.950728 | 0.049977 |      385271 |
| N            | O            |  763212717.00 | 1144418224289.36 | 1087192540204.5258 | 1130691602279.113229 | 25.497601 | 38233.010394 | 0.050000 |    29932726 |
| R            | NULL         |  377024766.00 |  565375805056.41 |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
| R            | F            |  377024766.00 |  565375805056.41 |  537107670359.2680 |  558591799092.170252 | 25.508535 | 38251.886057 | 0.049997 |    14780338 |
+--------------+--------------+---------------+------------------+--------------------+----------------------+----------+-------------+----------+-------------+
8 rows in set, 3 warnings (22.30 sec)
```

INSERT ... SELECT和REPLACE ... SELECT支持

原始SQL语句，如下所示：

```
insert into line_item_ap
SELECT   l_returnflag,
         l_linestatus,
         Sum(l_quantity)                                       AS sum_qty,
         Sum(l_extendedprice)                                  AS sum_base_price,
         Sum(l_extendedprice * (1 - l_discount))               AS sum_disc_price,
         Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
         Avg(l_quantity)                                       AS avg_qty,
         Avg(l_extendedprice)                                  AS avg_price,
         Avg(l_discount)                                       AS avg_disc,
         Count(*)                                              AS count_order
FROM     lineitem
WHERE    l_shipdate <= date '1998-12-01' - INTERVAL '93' day
GROUP BY l_returnflag,
         l_linestatus
ORDER BY l_returnflag,
         l_linestatus ;
```

- 未开启并行查询，耗时182.82秒。

```
mysql> insert into line_item_ap
    -> SELECT   l_returnflag,
    ->          l_linestatus,
    ->          Sum(l_quantity)                                       AS sum_qty,
    ->          Sum(l_extendedprice)                                  AS sum_base_price,
    ->          Sum(l_extendedprice * (1 - l_discount))               AS sum_disc_price,
    ->          Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    ->          Avg(l_quantity)                                       AS avg_qty,
    ->          Avg(l_extendedprice)                                  AS avg_price,
    ->          Avg(l_discount)                                       AS avg_disc,
    ->          Count(*)                                              AS count_order
    -> FROM     lineitem
    -> WHERE    l_shipdate <= date '1998-12-01' - INTERVAL '93' day
    -> GROUP BY l_returnflag,
    ->          l_linestatus
    -> ORDER BY l_returnflag,
    ->          l_linestatus ;
Query OK, 4 rows affected (3 min 2.82 sec)
```

- 开启并行查询后，耗时23.25秒，提升7.86倍。

```
mysql> insert into line_item_ap
    -> SELECT   l_returnflag,
    ->          l_linestatus,
    ->          Sum(l_quantity)                                    AS sum_qty,
    ->          Sum(l_extendedprice)                               AS sum_base_price,
    ->          Sum(l_extendedprice * (1 - l_discount))            AS sum_disc_price,
    ->          Sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    ->          Avg(l_quantity)                                    AS avg_qty,
    ->          Avg(l_extendedprice)                               AS avg_price,
    ->          Avg(l_discount)                                    AS avg_disc,
    ->          Count(*)                                           AS count_order
    -> FROM     lineitem
    -> WHERE    l_shipdate <= date '1998-12-01' - INTERVAL '93' day
    -> GROUP BY l_returnflag,
    ->          l_linestatus
    -> ORDER BY l_returnflag,
    ->          l_linestatus ;
Query OK, 4 rows affected (23.25 sec)
```

4.4. 高并发优化

4.4.1. Statement Concurrency Control

为了应对突发的数据库请求流量、资源消耗过高的语句访问以及SQL访问模型的变化，保证MySQL实例持续稳定运行，阿里云提供了基于语句规则的并发控制CCL（Concurrency Control），并提供了工具包（DBMS_CCL）便于您快捷使用。

使用限制

- PolarDB集群版本需为以下版本之一：
  - PolarDB MySQL引擎8.0版本
  - PolarDB MySQL引擎5.7版本且内核小版本需为5.7.1.0.6及以上
  - PolarDB MySQL引擎5.6版本
- 当集群版本为PolarDB MySQL引擎5.7版本且内核小版本为5.7.1.0.6及以上时，CCL与Thread Pool互斥，不能同时使用。

注意事项

- CCL的操作不产生Binlog，所以CCL的操作只影响当前节点。例如主节点进行CCL操作，不会同步到只读节点。
- CCL提供超时机制以应对DML导致事务锁死锁，等待中的线程也会响应事务超时和线程KILL操作以应对死锁。

功能设计

CCL规则定义了如下三个维度特征：

维度	说明
SQL command	SQL命令类型（如SELECT、UPDATE、INSERT、DELETE等）。
Object	SQL命令操作的对象（如TABLE、VIEW等）。
keywords	SQL命令的关键字。

创建CCL规则表

PolarDB设计了一个系统表（concurrency_control）保存CCL规则，系统启动时会自动创建该表，无需您手动创建。该系统表的创建语句如下所示：

```
CREATE TABLE `concurrency_control` (
  `Id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Type` enum('SELECT','UPDATE','INSERT','DELETE') NOT NULL DEFAULT 'SELECT',
  `Schema_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Table_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Concurrency_count` bigint(20) DEFAULT NULL,
  `Keywords` text COLLATE utf8_bin,
  `State` enum('N','Y') NOT NULL DEFAULT 'Y',
  `Ordered` enum('N','Y') NOT NULL DEFAULT 'N',
  PRIMARY KEY (`Id`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_bin
STATS_PERSISTENT=0 COMMENT='Concurrency control'
```

参数	说明
Id	CCL规则ID。 说明 ID越大，规则的优先级越高，关于如何修改优先级，请参见管理CCL规则。
Type	SQL命令类型（如SELECT、UPDATE、INSERT、DELETE等）。
Schema_name	数据库名。
Table_name	数据库内的表名。
Concurrency_count	并发数。
Keywords	关键字，多个关键字用英文分号（;）分隔。
State	本规则是否启用，取值范围为N（否）和Y（是），默认为Y。
Ordered	Keywords中多个关键字是否按顺序匹配，取值范围为N（否）和Y（是），默认为N。

管理CCL规则

为了便捷地管理CCL规则，PolarDB在DBMS_CCL中定义了四个本地存储规则。详细说明如下：

- add_ccl_rule

  增加规则。命令如下：

  ```
  dbms_ccl.add_ccl_rule('<Type>','<Schema_name>','<Table_name>',<Concurrency_count>,'<Keywords>');
  ```

  示例：

- SELECT语句的并发数为10。

```
mysql> call dbms_ccl.add_ccl_rule('SELECT', '', '', 10, '');
```

- SELECT语句中出现关键字key1的并发数为20。

```
mysql> call dbms_ccl.add_ccl_rule('SELECT', '', '', 20, 'key1');
```

- test.t表的SELECT语句的并发数为10。

```
mysql> call dbms_ccl.add_ccl_rule('SELECT', 'test', 't', 20, '');
```

- del_ccl_rule

  删除规则。命令如下：

```
dbms_ccl.del_ccl_rule(<Id>);
```

  示例：

  删除规则ID为15的CCL规则。

```
mysql> call dbms_ccl.del_ccl_rule(15);
```

  说明 如果删除的规则不存在，系统会报相应的警告，您可以使用 `show warnings;` 查看警告内容。

```
mysql> call dbms_ccl.del_ccl_rule(100);
  Query OK, 0 rows affected, 2 warnings (0.00 sec)
mysql> show warnings;
+---------+------+-----------------------------------------------------+
| Level   | Code | Message                                             |
+---------+------+-----------------------------------------------------+
| Warning | 7514 | Concurrency control rule 100 is not found in table |
| Warning | 7514 | Concurrency control rule 100 is not found in cache |
+---------+------+-----------------------------------------------------+
```

- show_ccl_rule

  查看内存中已启用规则。命令如下：

```
dbms_ccl.show_ccl_rule();
```

  示例：

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNING
| WAITTING | KEYWORDS |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
|   17 | SELECT | test   | t     | Y     | N     |                30 |       0 |       0
|        0 |          |
|   16 | SELECT |        |       | Y     | N     |                20 |       0 |       0
|        0 | key1     |
|   18 | SELECT |        |       | Y     | N     |                10 |       0 |       0
|        0 |          |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
```

> **说明** 关于MATCHED、RUNNING和WAITTING参数的说明如下：
>
> - MATCHED：规则匹配成功次数。
> - RUNNING：该规则下正在并发执行的线程数。
> - WAITTING：该规则下正在等待执行的线程数。

- flush_ccl_rule

  如果通过修改表concurrency_control中的内容来修改规则，您还需要使用如下命令让该规则生效：

  ```
  dbms_ccl.flush_ccl_rule();
  ```

  示例：

  ```
  mysql> update mysql.concurrency_control set CONCURRENCY_COUNT = 15 where Id = 18;
  Query OK, 1 row affected (0.00 sec)
  Rows matched: 1  Changed: 1  Warnings: 0
  mysql> call dbms_ccl.flush_ccl_rule();
  Query OK, 0 rows affected (0.00 sec)
  ```

- 您可以通过使用UPDATE语句修改规则ID来调整目标规则的优先级。

  ```
  update mysql.concurrency_control set ID = xx where ID = xx;
  ```

  示例：

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNING
| WAITTING | KEYWORDS |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
|   17 | SELECT | test   | t     | Y     | N     |                30 |       0 |       0
|        0 |          |
|   16 | SELECT |        |       | Y     | N     |                20 |       0 |       0
|        0 | key1     |
|   18 | SELECT |        |       | Y     | N     |                10 |       0 |       0
|        0 |          |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
mysql> update mysql.concurrency_control set ID = 20 where ID = 17;
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNING
| WAITTING | KEYWORDS |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
|   16 | SELECT |        |       | Y     | N     |                20 |       0 |       0
|        0 | key1     |
|   18 | SELECT |        |       | Y     | N     |                10 |       0 |       0
|        0 |          |
|   20 | SELECT | test   | t     | Y     | N     |                30 |       0 |       0
|        0 |          |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
```

功能测试

- 设计如下三条规则分别对应三个维度：

```
call dbms_ccl.add_ccl_rule('SELECT', 'test', 'sbtest1', 3, '');  //SELECT命令操作表sbtest1并发数为3
call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, 'sbtest2');       //SELECT命令关键字sbtest2并发数为2
call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, '');            //SELECT命令并发数为2
```

- 使用Sysbench进行测试，具体场景如下：
  - 64 threads
  - 4 tables
  - select.lua
- 查看规则并发数情况如下：

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+---------+-------+-------+-------------------+---------+-------
--+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE   | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNIN
G | WAITTING | KEYWORDS |
+------+--------+--------+---------+-------+-------+-------------------+---------+-------
--+----------+----------+
|   20 | SELECT | test   | sbtest1 | Y     | N     |                 3 |     389 |
3 |        9 |          |
|   21 | SELECT |        |         | Y     | N     |                 2 |     375 |
2 |       14 | sbtest2  |
|   22 | SELECT |        |         | Y     | N     |                 2 |     519 |
2 |       34 |          |
+------+--------+--------+---------+-------+-------+-------------------+---------+-------
--+----------+----------+
3 rows in set (0.00 sec)
```

查看RUNNING列，符合预期的并行数量。

4.4.2. Inventory Hint

PolarDB提供Inventory Hint，帮助您快速提交、回滚事务。您还可以将Inventory Hint和Statement Queue一起配合使用，有效提高业务吞吐能力。

前提条件

PolarDB集群版本需为PolarDB MySQL引擎8.0版本且Revision version为8.0.1.1.1及以上，您可以通过查询版本号确认集群版本。

背景信息

在电商秒杀活动等业务场景中，减少库存是一个常见的需要高并发，同时也需要串行化的任务模型，PolarDB使用排队和事务性Hint来控制并发和快速提交或回滚事务，提高业务吞吐能力。

语法

PolarDB提供的Inventory Hint支持SELECT、UPDATE、INSERT、DELETE语句。

Inventory Hint包括如下三个事务语法：

- `COMMIT_ON_SUCCESS`：当前语句执行成功就提交事务。

  示例：

  - PolarDB MySQL引擎5.6版本

    ```
    UPDATE COMMIT_ON_SUCCESS T
    SET c = c - 1
    WHERE id = 1;
    ```

  - PolarDB MySQL引擎8.0版本

    ```
    UPDATE /*+ COMMIT_ON_SUCCESS */ T
    SET c = c - 1
    WHERE id = 1;
    ```

- `ROLLBACK_ON_FAIL`：当前语句执行失败就回滚事务。

  示例：

  - PolarDB MySQL引擎5.6版本

    ```
    UPDATE ROLLBACK_ON_FAIL T
    SET c = c - 1
    WHERE id = 1;
    ```

  - PolarDB MySQL引擎8.0版本

    ```
    UPDATE /*+ ROLLBACK_ON_FAIL */ T
    SET c = c - 1
    WHERE id = 1;
    ```

- `TARGET_AFFECT_ROW number`：如果当前语句影响的行数符合设定的行数就成功，否则语句失败。

  假设Target Affect Row的值设为1，如果更新语句实际更新到了一条数据则认为成功，如果更新没有命中任何记录则认为失败。

  示例：

  - PolarDB MySQL引擎5.6版本

    ```
    UPDATE TARGET_AFFECT_ROW 1  T
    SET c = c - 1
    WHERE id = 1;
    ERROR HY000: The affected row number does not match that of user specified.
    ```

  - PolarDB MySQL引擎8.0版本

    ```
    UPDATE /*+ TARGET_AFFECT_ROW(1) */ T
    SET c = c - 1
    WHERE id = 1;
    ERROR HY000: The affected row number does not match that of user specified.
    ```

配合Statement Queue使用

UPDATE、INSERT、DELETE语句中的`COMMIT_ON_SUCCESS`、`ROLLBACK_ON_FAIL`、`TARGET_AFFECT_ROW number`这三个事务语法可以配合Statement Queue进行排队。

示例（以PolarDB MySQL引擎5.6版本语法为例）：

```
UPDATE COMMIT_ON_SUCCESS POLARDB_STATEMENT_CONCURRENT_QUEUE id ROLLBACK_ON_FAIL TARGET_AFFE
CT_ROW 1 t
SET col1 = col1 + 1
WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
UPDATE COMMIT_ON_SUCCESS POLARDB_STATEMENT_CONCURRENT_QUEUE 1 ROLLBACK_ON_FAIL TARGET_AFFEC
T_ROW 1 t
SET col1 = col1 + 1
WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

4.4.3. Statement Queue

PolarDB设计了针对语句的排队（Statement Queue）机制，将语句进行分桶排队，尽量把可能具有相同冲突（例如操作相同行）的语句放在一个桶内排队，减少冲突的开销。

背景信息

MySQL的服务层和引擎层在语句并发执行过程中，有很多串行的点容易导致冲突。例如在DML语句中，比较常见的是事务锁冲突。InnoDB中事务锁的最细粒度是行级锁，如果语句针对相同行进行并发操作，会导致比较严重的冲突，系统吞吐量会随着并发的增加而递减。PolarDB提供Statement Queue机制，能够减少冲突开销、有效提高数据库性能。

使用限制

- PolarDB集群版本需为以下版本之一：
  - PolarDB MySQL引擎8.0版本且内核小版本需为8.0.1.1.10及以上
  - PolarDB MySQL引擎5.7版本且内核小版本需为5.7.1.0.6及以上。
  - PolarDB MySQL引擎5.6版本且内核小版本需为20200601及以上。
- PolarDB MySQL引擎5.6版本与PolarDB MySQL引擎5.7和8.0版本使用了不同的语法实现Statement Queue机制，因此仅PolarDB MySQL引擎5.7和8.0版本的集群支持与Statement Outline配合使用完成在线修改SQL statement的并发控制。
- 热点行性能优化总开关开启后（hotspot参数设置为ON后），将不能使用Statement Queue机制。

效果

在针对单行进行并发UPDATE的场景测试中，相比原生的MySQL，PolarDB有接近4倍的性能提升。

语法

- 针对PolarDB MySQL引擎5.6版本，您可以通过以下方式进行hash分桶。

  > **说明** `POLARDB_STATEMENT_CONCURRENT_QUEUE` 语法支持SELECT、UPDATE、INSERT和DELETE等语句。

  - 根据int或string值进行hash分桶。
    - 语法：

      ```
      POLARDB_STATEMENT_CONCURRENT_QUEUE [int | string]
      ```

    - 示例：

      ```
      insert POLARDB_STATEMENT_CONCURRENT_QUEUE 1 into t values(1, 1, 'xpchild');
      update POLARDB_STATEMENT_CONCURRENT_QUEUE 1 t set c=c+1 where id = 1;
      update POLARDB_STATEMENT_CONCURRENT_QUEUE "xpchild" t set col1 = col1+1 where id =1;
      ```

- 根据WHERE条件中的field值进行hash分桶。
  - 语法：

```
POLARDB_STATEMENT_CONCURRENT_QUEUE [field]
```

  - 示例：

```
 select POLARDB_STATEMENT_CONCURRENT_QUEUE id * from t where 3 = id;
 update POLARDB_STATEMENT_CONCURRENT_QUEUE id t set c=c+1 where id = 1 and name = 'xp
child';
```

> **说明** 在 `POLARDB_STATEMENT_CONCURRENT_QUEUE [field]` 语法中，WHERE条件只支持原始字段（没有在字段上使用任何函数、计算等）的二元运算，并且二元运算的右侧值必须是数字或者字符串。

- 针对PolarDB MySQL引擎5.7和8.0，您可以通过以下HINT语法进行hash分桶。

> **说明** `ccl_queue_value` 语法支持SELECT、UPDATE、INSERT和DELETE等语句。

  - 根据int或string值进行hash分桶。
    - 语法：

```
/*+ CCL_QUEUE_VALUE([INT | STRING)] */
```

    - 示例：

```
update /*+ ccl_queue_value(1) */ t set c=c+1 where id = 1;
update /*+ ccl_queue_value('xpchild') */ t set c=c+1
                                          where name = 'xpchild';
```

  - 根据WHERE条件中的field值进行hash分桶。
    - 语法：

```
/*+ CCL_QUEUE_FIELD(STRING) */
```

    - 示例：

```
  update /*+ ccl_queue_field("id") */ t set c=c+1
                                      where id = 1 and name = 'xpchild';
```

> **说明** 在 `/*+ CCL_QUEUE_FIELD(STRING) */` 语法中，WHERE条件只支持原始字段（没有在字段上使用任何函数、计算等）的二元运算，并且二元运算的右侧值必须是数字或者字符串。

变量

PolarDB提供了如下两个变量来定义语句队列的桶数量和大小：

变量	说明
ccl_queue_bucket_count	表示桶的数量，取值范围为1~64，默认值为4。

变量	说明
ccl_queue_bucket_size	表示一个桶允许的并发数，取值范围为1~4096，默认值为64。

接口

PolarDB提供以下两个接口便于您查询Statement Queue状态：

- dbms_ccl.show_ccl_queue()

  查询当前Statement Queue状态。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+------------------+---------+---------+----------+
|    1 | QUEUE |                64 |       1 |       0 |        0 |
|    2 | QUEUE |                64 |   40744 |      65 |        6 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+------------------+---------+---------+----------+
4 rows in set (0.01 sec)
```

  说明 对其中的CONCURRENCY_COUNT、MATCHED、RUNNING和WAITTING说明如下：
  - CONCURRENCY_COUNT：最大并发数。
  - MATCHED：命中规则的总数。
  - RUNNING：当前并发的数量。
  - WAITTING：当前等待的数量。

- dbms_ccl.flush_ccl_queue()

  清理内存中的数据。

```
mysql> call dbms_ccl.flush_ccl_queue();
Query OK, 0 rows affected (0.00 sec)
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+------------------+---------+---------+----------+
|    1 | QUEUE |                64 |       0 |       0 |        0 |
|    2 | QUEUE |                64 |       0 |       0 |        0 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+------------------+---------+---------+----------+
4 rows in set (0.00 sec)
```

配合Statement Outline在线修改

针对PolarDB MySQL引擎5.7和8.0版本，Statement Queue还可以配合Statement Outline使用实现快速在线修改SQL statement的并发控制，避免介入冗长的应用业务代码的修改。 下文使用SysBench的update_non_index为例进行演示。

> **说明** 由于语法不同，PolarDB MySQL引擎5.6版本不支持与Statement Outline配合使用。

- 测试环境
  - 测试表结构

```
CREATE TABLE `sbtest1` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `k` int(10) unsigned NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 MAX_ROWS=1000000;
```

  - 测试语句

```
UPDATE sbtest1 SET c='xpchild' WHERE id=0;
```

  - 测试脚本

```
./sysbench
--mysql-host= {$ip}
--mysql-port= {$port}
--mysql-db=test
--test=./sysbench/share/sysbench/update_non_index.lua
--oltp-tables-count=1
--oltp_table_size=1
--num-threads=128
--mysql-user=u0
```

- 测试过程

i. 在线增加Statement Outline。

```
mysql> CALL DBMS_OUTLN.add_optimizer_outline('test', '', 1,
                                             ' /*+ ccl_queue_field("id") */ ',
                         "UPDATE sbtest1 SET c='xpchild' WHERE id=0");
Query OK, 0 rows affected (0.01 sec)
```

ii. 查看Statement Outline。

```
mysql> call dbms_outln.show_outline();
+------+--------+-----------------------------------------------------------------+-
----------+-------+------+---------------------------------+------+----------+--------
-------------------------------------+
| ID   | SCHEMA | DIGEST                                                          |
TYPE      | SCOPE | POS  | HINT                            | HIT  | OVERFLOW | DIGEST_
TEXT                                  |
+------+--------+-----------------------------------------------------------------+-
----------+-------+------+---------------------------------+------+----------+--------
-------------------------------------+
|    1 | test   | 7b945614749e541e0600753367884acff5df7e7ee2f5fb0af5ea58897910f023 |
OPTIMIZER |       |    1 |  /*+ ccl_queue_field("id") */  |    0 |        0 | UPDATE
`sbtest1` SET `c` = ? WHERE `id` = ? |
+------+--------+-----------------------------------------------------------------+-
----------+-------+------+---------------------------------+------+----------+--------
-------------------------------------+
1 row in set (0.00 sec)
```

iii. 验证Statement Outline生效。

```
mysql> explain UPDATE sbtest1 SET c='xpchild' WHERE id=0;
+----+-------------+---------+------------+-------+---------------+---------+--------
-+--------+------+----------+-------------+
| id | select_type | table   | partitions | type  | possible_keys | key     | key_len
| ref   | rows | filtered | Extra       |
+----+-------------+---------+------------+-------+---------------+---------+--------
-+--------+------+----------+-------------+
|  1 | UPDATE      | sbtest1 | NULL       | range | PRIMARY       | PRIMARY | 4
| const |    1 |   100.00 | Using where |
+----+-------------+---------+------------+-------+---------------+---------+--------
-+--------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
mysql> show warnings;
+-------+------+-----------------------------------------------------------------------
-------------------------------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------
-------------------------------------------------------+
| Note  | 1003 | update /*+ CCL_QUEUE_FIELD('id') */ `test`.`sbtest1` set `test`.`sbt
est1`.`c` = 'xpchild' where (`test`.`sbtest1`.`id` = 0) |
+-------+------+-----------------------------------------------------------------------
-------------------------------------------------------+
1 row in set (0.00 sec)
```

iv. 查询Statement Queue状态。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+------------------+---------+---------+----------+
|    1 | QUEUE |                64 |       0 |       0 |        0 |
|    2 | QUEUE |                64 |       0 |       0 |        0 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+------------------+---------+---------+----------+
4 rows in set (0.00 sec)
```

v. 开启测试。

```
sysbench
--mysql-host= {$ip}
--mysql-port= {$port}
--mysql-db=test
--test=./sysbench/share/sysbench/update_non_index.lua
--oltp-tables-count=1
--oltp_table_size=1
--num-threads=128
--mysql-user=u0
```

vi. 验证测试效果。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+------------------+---------+---------+----------+
|    1 | QUEUE |                64 |   10996 |      63 |        4 |
|    2 | QUEUE |                64 |       0 |       0 |        0 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+------------------+---------+---------+----------+
4 rows in set (0.03 sec)
mysql> call dbms_outln.show_outline();
+------+--------+-----------+-----------+-------+------+-----------------------------
---+--------+----------+-----------------------------------------------+
| ID   | SCHEMA | DIGEST    | TYPE      | SCOPE | POS  | HINT
| HIT    | OVERFLOW | DIGEST_TEXT                                    |
+------+--------+-----------+-----------+-------+------+-----------------------------
---+--------+----------+-----------------------------------------------+
|    1 | test   | xxxxxxxxx | OPTIMIZER |       |    1 |  /*+ ccl_queue_field("id") *
/  | 115795 |        0 | UPDATE `sbtest1` SET `c` = ? WHERE `id` = ? |
+------+--------+-----------+-----------+-------+------+-----------------------------
---+--------+----------+-----------------------------------------------+
1 row in set (0.00 sec)
```

说明 查询结果显示Statement Outline命中了115,795次规则，Statement Queue状态显示命中了10,996次排队，当前运行并发63个，排队等待4个。

4.4.4. 热点行优化

本文将介绍热点行性能优化功能。

前提条件

PolarDB集群版本需为以下版本之一：

- PolarDB MySQL引擎5.6版本且内核小版本需为20200601及以上。
- PolarDB MySQL引擎5.7版本且内核小版本需为5.7.1.0.17及以上。
- PolarDB MySQL引擎8.0版本且内核小版本需为8.0.1.1.10及以上

> **说明** 关于如何升级内核小版本请参见版本管理。

背景信息

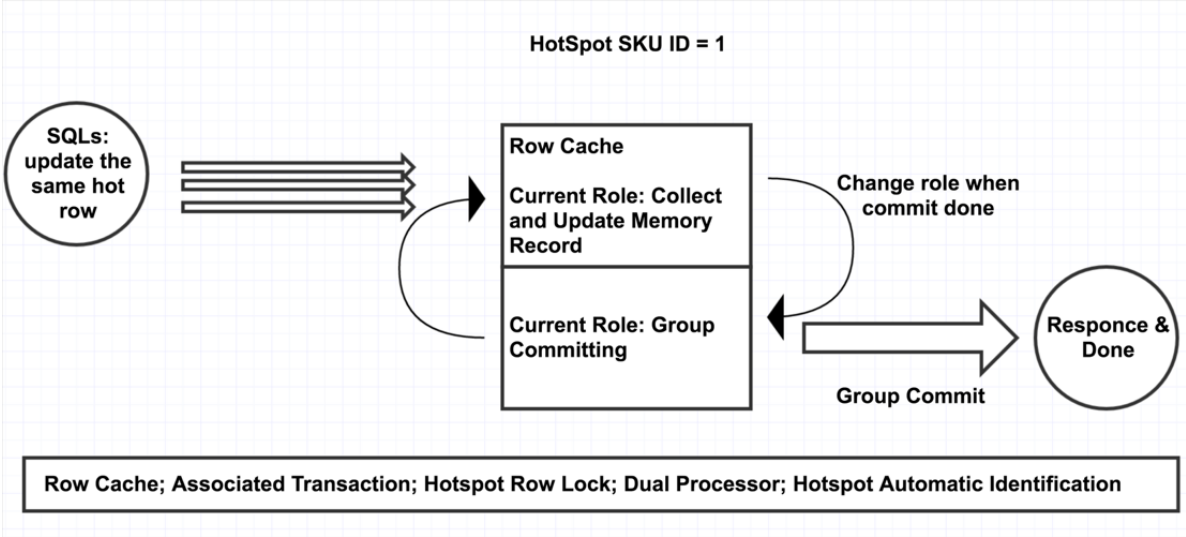
热点行面临以下问题：

- 数据库中那些会被频繁执行增删改查操作的数据行称为热点行。当一个事务对一行数据进行更新时，会对目标数据行加锁，直到事务提交或回滚时才释放。同一时段内，对于同一个数据行，只有一个事务能够进行更新，其它事务需要等待。由此可见，对单一热点行的更新请求其实是串行执行的，传统的分库分表策略在性能提升方面并不会有太大帮助。
- 在电商平台业务中，限购、秒杀是常用的促销手段。在这些场景下，大量对热点行的更新请求在极短时间间隔内到达后台数据库系统，必然造成严重的行锁竞争和等待，影响系统性能。如果一个更新请求等待执行的时间变长，将会对业务层面产生显著负面影响。

针对上述问题，单纯提高计算机硬件配置已经无法满足这样的低延迟需求。因此PolarDB在数据库内核层进行了创新性的优化，不但能够自动识别热点行更新请求，而且将一定时间间隔内对同一数据行的更新操作进行分组，不同分组采用流水线的方式并行处理，通过这些优化，极大地提升了系统的性能。具体方案如下：

- 串行处理变流水线处理

  为了提升数据库系统的性能，最直接的方法是使用并行处理，但是对同一热点行的更新操作很难做到完全并行，PolarDB创新性地使用了流水线处理方式，最大限度地将热点行更新操作并行化。



  热点行更新操作所使用的SQL语句会用 `autocommit` 或者 `commit_on_success` 进行标记，优化后的MySQL内核层会自动识别带此类标记的更新操作，在一定的时间间隔内，将收集到的更新操作按照主键或者唯一键进行Hash，对于Hash到同一个桶中的更新操作，会将它们按照到达的先后顺序分组分批地进行处理和提交。

为了使用流水线方式处理这些更新操作，需要使用两个执行单元对它们进行分组。当第一个分组收集完毕准备提交时，第二个分组立即开始收集更新操作；当第二个分组收集完毕准备提交时，第一个分组已经提交完毕并开始收集新一批的更新操作，两个分组不断切换，并行执行。

现如今多核CPU的使用已经非常普遍。这样的流水线式的处理方式能够充分利用硬件资源，提升CPU的使用率，提高数据库系统的并行处理能力，从而最大限度地提升数据库系统的吞吐量。

- 消除申请行锁时的等待

  为了保证数据的逻辑一致性，对一个数据行进行更新时，首先会对该数据行加锁。如果加锁请求无法立刻满足，则进入等待状态，这样一来，不但增加了处理延迟，而且还会触发死锁检测，导致额外的资源消耗。

  前面提到，我们会按照时间顺序将对同一数据行的更新操作进行分组，组内第一个更新操作为Leader，其读取目标数据行并且加锁，后续更新操作为Follower，对目标数据行加锁时，如果发现Leader已经持有行锁，不需要等待，直接获得行锁。

  通过这个优化，能够减少行锁的加锁次数和时间开销，整个数据库系统的性能有了显著的提升。

- 减少B-tree索引的遍历

  MySQL是以B-tree索引的方式管理数据的，每次执行查询时，都需要遍历索引才能定位到目标数据行，数据表越大，索引层级越多，遍历时间就越长。

  在前面提到的对更新操作进行分组的机制中，只有每组的Leader遍历索引定位数据行，之后把更新后的数据行缓存（row cache）在内存中，同组的Follower加锁成功后，直接从内存中读取目标数据行，而不需要再次遍历索引。

  这样一来，从整体上减少了索引遍历的次数和时间开销。

使用限制

在以下场景中，热点行性能优化将不会被使用：

- 热点行所属的数据表是分区表。
- 热点行所属的数据表上定义了触发器（Trigger）。
- 热点行使用了Statement Queue机制。

使用方法

- 使用热点行性能优化功能前，您需要对如下参数进行配置。如何配置参数，详细操作请参见设置集群参数。

参数	说明
hotspot	热点行性能优化总开关，取值为ON或OFF，默认值为OFF。
hotspot_for_autocommit	`Auto-commit` 模式下的UPDATE语句能否使用热点行性能优化，取值为ON或OFF，默认值为OFF。 **说明** 只有当hotspot为ON时，才支持配置该参数。
hotspot_update_max_wait_time	行数据分组批量更新（即Group update）过程中Leader等待Follower加入该分组的等待时间，单位为微秒（us），默认值为100us。

参数	说明
hotspot_lock_type	行数据分组批量更新过程中是否使用新类型的行锁，取值为ON或OFF，默认值为OFF。 **说明** 该参数打开时，对相同热点行的更新操作申请行锁时不需要等待，从而提升性能。

说明

- hotspot_for_autocommit、hotspot_update_max_wait_time和hotspot_lock_type参数暂未开放，如需使用请提交工单联系技术支持。
- 热点行性能优化需要依赖Binlog，所以，当Binlog功能处于关闭状态时，试图打开参数hotspot时会失败并报错。

  当Binlog处于关闭状态时，试图使用 `MySQL [(none)]> set global hotspot=ON;` 命令打开参数hotspot时会失败，并出现如下报错：

  ```
  ERROR 3042 (HY000):
  Variable 'hotspot' cannot be enabled because 'bin logging' is enabled/disabled
  ```

  如何开启Binlog，详细操作步骤请参见开启Binlog。
- 当全局Binlog开启，但是会话级别的Binlog关闭时，可以打开参数hotspot，但是执行UPDATE语句时并不会使用热点行性能优化。
- 当参数rds_ic_reduce_hint_enable处于打开状态时，试图使用 `MySQL [(none)]> set global hotspot=ON;` 命令打开参数hotspot时会失败，并出现如下报错：

  ```
  ERROR 3042 (HY000): Variable 'hotspot' cannot be enabled because 'rds_ic_reduce_hint_enable' is enabled/disabled
  ```

- 您可以使用如下命令查看参数配置：

```
show variables like "hotspot%";
```

返回结果示例

```
+------------------------------+-------+
|
Variable_name                | Value |
+------------------------------+-------+
|
hotspot                      | OFF   |
|
hotspot_for_autocommit       | OFF   |
|
hotspot_lock_type            | OFF   |
|
hotspot_update_max_wait_time | 100   |
+------------------------------+-------+
```

- 热点行优化功能涉及到三种新的优化器hint：

hint	是否必选	说明
COMMIT_ON_SUCCESS	必选	更新成功时提交。
ROLLBACK_ON_FAIL	可选	更新失败时回滚。
TARGET_AFFECT_ROW(1)	可选	显式指定该请求只会更新一行，若不符合，更新失败。

- 您可以使用如下命令查看热点行性能优化使用情况：

```
show global status like 'Group_update%';
```

性能测试

- 测试所需如下数据表定义和测试语句：
  - 数据表定义

    ```
    CREATE TABLE sbtest(id INT UNSIGNED NOT NULL, c BIGINT UNSIGNED NOT NULL) PRIMARY KEY (id)
    ```

  - 测试语句

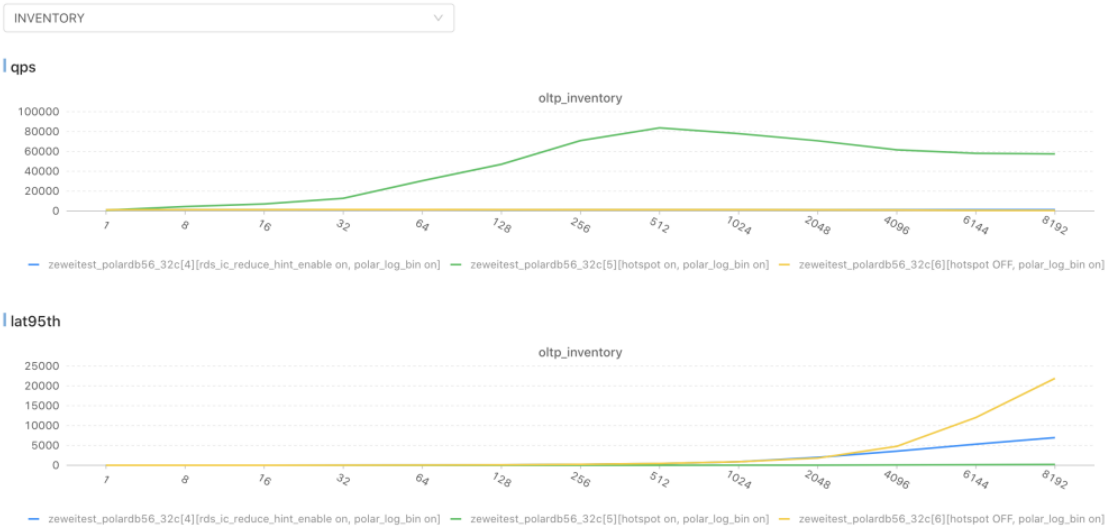
    ```
    UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL TARGET_AFFECT_ROW 1 sbtest SET c=c+1 WHERE id = 1
    ```

- 测试工具：Sysbench
- 测试场景和测试结果
  - 测试场景1：单个热点行加8内核CPU。

    测试结果：在单热点行加8内核CPU的场景下，引入热点行优化后，库存热点性能在高并发时提升近64倍。

- 测试场景2：单个热点行加32内核CPU

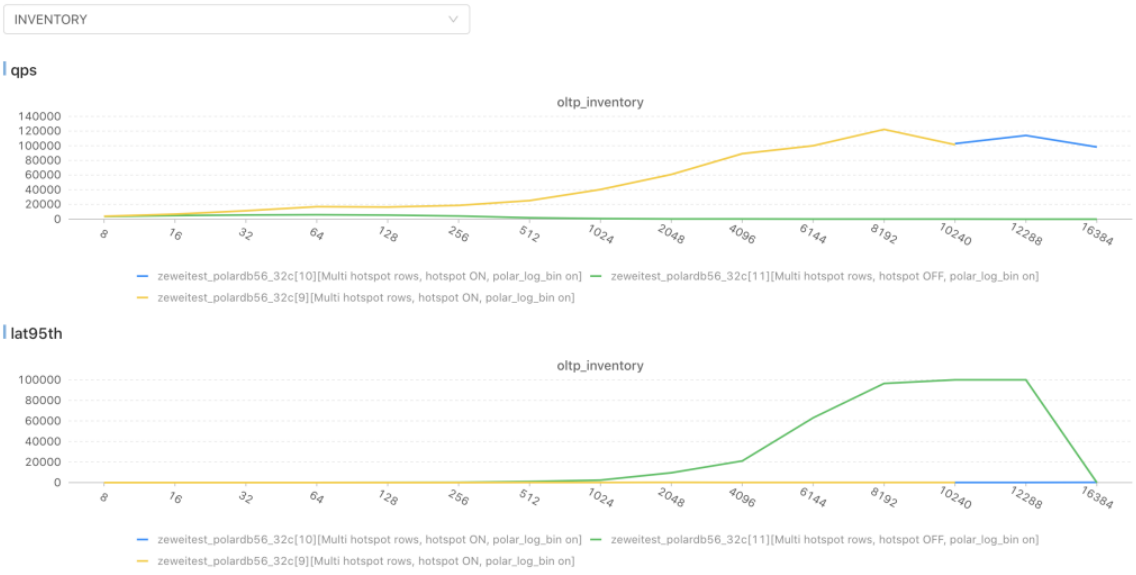
  测试结果：在单热点行加32核CPU的场景下，引入热点行优化后，峰值QPS提升了63倍；高并发为8000时，性能提升了46倍。



- 测试场景3：8个热点行加32核CPU

  测试结果：在多热点行（8个）加32核CPU的场景下，引入热点行优化后，峰值QPS提升了20倍。

  且当高并发达到16000时，在未使用热点行优化功能的情况下，更新操作会导致数据库出现故障无法返回更新操作结果；但使用热点行优化后，更新操作可以正常返回，且QPS维持稳定。



4.4.5. Thread Pool

为了发挥出PolarDB MySQL引擎的最佳性能，PolarDB提供线程池（Thread Pool）功能，将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。

优势

MySQL默认的线程使用模式是会话独占模式，每个会话都会创建一个独占的线程。当有大量的会话存在时，会出现大量的资源竞争导致性能下降。同时大量的系统线程调度和缓存失效也会导致性能急剧下降。

PolarDB的线程池实现了不同类型SQL操作的优先级及并发控制机制，将连接数始终控制在最佳连接数附近，使PolarDB数据库在高连接大并发情况下始终保持高性能。线程池的优势如下：

- 当大量线程并发工作时，线程池会自动调节并发的线程数量在合理的范围内，从而避免线程调度工作过多和大量缓存失效。
- 大量的事务并发执行时，线程池会将语句和事务分为不同的优先级，分别控制语句和事务的并发数量，从而减少资源竞争。
- 线程池给予管理类的SQL语句更高的优先级，保证这些语句优先执行。这样在系统负载很高时，新建连接、管理、监控等操作也能够稳定执行。
- 线程池给予复杂查询SQL语句相对较低的优先级，并且有最大并发数的限制。这样可以避免过多的复杂SQL语句将系统资源耗尽，导致整个数据库服务不可用。

使用Thread Pool

Thread Pool设计了以下参数，您可以在控制台进行修改。具体请参见设置集群参数。

参数名称	说明
loose_thread_pool_enabled	是否开启线程池功能。取值： • ON • OFF 默认值：OFF。 说明 开启或关闭线程池功能无需重启实例。
loose_thread_pool_high_prio_mode	线程池高优先级队列模式。取值： • transactions：已开启事务的SQL将被加入高优先级队列，并被赋予thread_pool_high_prio_tickets这个门票，接下来执行的SQL都将被放入高优先级队列，直到门票被耗尽。 • statements：所有SQL都会加入高优先级队列。 • none：所有SQL都不加入高优先级队列。 默认值：transactions。 说明 仅PolarDB MySQL引擎5.6和5.7版本支持该参数。
loose_thread_pool_high_prio_tickets	高优先级队列的单次门票数。 取值：0~4294967295。 默认值：4294967295。 说明 仅PolarDB MySQL引擎5.6和5.7版本支持该参数。

参数名称	说明
loose_thread_pool_idle_timeout	释放线程池空闲线程的时间阈值，超过此时间，没有服务任何请求的空闲线程将被释放。 取值：0~31536000。 单位：秒，默认值：60。 **说明** 仅PolarDB MySQL引擎5.6和5.7版本支持该参数。
loose_thread_pool_oversubscribe	每个线程组中允许的活跃线程的数量。 活跃线程是指正在执行SQL语句的线程，但是不包括以下两种情形： • SQL语句在等待磁盘I/O。 • SQL语句在等待事务提交。 取值：1~1000。 默认值：10。
loose_thread_pool_stall_limit	判断线程池进入拥塞状态的时间阈值。 当线程池进入拥塞状态时，系统会创建新的线程来服务SQL。 取值：1~18446744073709551615。 单位：毫秒。默认值：30。 **说明** 仅PolarDB MySQL引擎5.6和5.7版本支持该参数。
loose_bypass_thread_pool_ips	配置不受线程池限制的客户端ip地址。即便线程池被占满时，也可以执行SQL来进行一些管理操作。 配置示例： `10.69.96.16,10.69.96.17` **说明** 仅PolarDB MySQL引擎8.0.1.1.19及之后版本支持该参数。

参数名称	说明
loose_bypass_thread_pool_check_ignore_proxy	通过 `loose_bypass_thread_pool_ips` 检测客户端ip地址时，是否忽略通过Proxy连接的客户端ip地址。取值： • ON：表示对于通过Proxy连接的客户端ip地址，即便 `loose_bypass_thread_pool_ips` 配置了该客户端ip地址，也要受线程池功能的约束。 • OFF：表示对于通过Proxy连接的客户端ip地址，如果 `loose_bypass_thread_pool_ips` 配置了该客户端ip地址，则不受线程池功能的约束。 默认值：ON。 **说明** • 仅PolarDB MySQL引擎8.0.1.1.19及之后版本支持该参数。 • 该参数暂不支持修改，如需修改请提交工单联系技术支持。
loose_thread_pool_high_priority_users	配置高优先级别的数据库账号。配置后这些账号的请求将会被放到线程池的高优先级队列中，优先进行处理。 配置示例： `root1, root2` **说明** • 仅PolarDB MySQL引擎8.0.1.1.19及之后版本支持该参数。 • 配置该参数后，仅对新建的数据库连接生效。 • 不建议您配置过多的高优先级账号。
loose_thread_pool_mark_ddl_thread_timeout_sec	配置DDL的线程池超时时间阈值。超过此时间后，将DDL标记为超时状态，系统会创建新的线程来处理请求。 取值：0~864000。 单位：秒。默认值：600。 **说明** 仅PolarDB MySQL引擎8.0.1.1.19及之后版本支持该参数。

参数名称	说明
loose_thread_pool_mark_ddl_thread_timeout_immediately	在线程池处于高负载状态，造成低优先级队列发生堆积时，是否立刻将DDL标记为超时状态，此时系统会创建新的线程来处理请求。该参数适用于需要经常批量DDL的业务场景中。取值： • ON • OFF 默认值：OFF。 说明 • 仅PolarDB MySQL引擎8.0.1.1.19及之后版本支持该参数。 • 该参数暂不支持修改，如需修改请提交工单联系技术支持。

查询Thread Pool状态

您可以通过如下命令查询Thread Pool状态：

```
select * from information_schema.THREAD_POOL_STATUS;
```

返回示例如下所示。

```
mysql> select * from information_schema.THREAD_POOL_STATUS;
+----+--------------+--------------------+----------------------+-------------------+-----
----------------------+-----------------+-----------------+------------------+
| ID | THREAD_COUNT | ACTIVE_THREAD_COUNT | WAITING_THREAD_COUNT | DUMP_THREAD_COUNT | SLOW
_THREAD_TIMEOUT_COUNT | CONNECTION_COUNT | LOW_QUEUE_COUNT | HIGH_QUEUE_COUNT |
+----+--------------+--------------------+----------------------+-------------------+-----
----------------------+-----------------+-----------------+------------------+
|  0 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  1 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  2 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  3 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  4 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  5 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  6 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  7 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  8 |            1 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
|  9 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
| 10 |            2 |                  0 |                    0 |                 0 |
0 |                0 |               0 |                0 |
```

```
| 11 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 12 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 13 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 14 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 15 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 16 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 17 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 18 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 19 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 20 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 21 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 22 |            2 |                1 |                  0 |                0 |
0 |             1 |            0 |             0 |
| 23 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 24 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 25 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 26 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 27 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 28 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 29 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 30 |            2 |                0 |                  0 |                0 |
0 |             1 |            0 |             0 |
| 31 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 32 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 33 |            2 |                0 |                  0 |                0 |
0 |             1 |            0 |             0 |
| 34 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 35 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 36 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
| 37 |            2 |                0 |                  0 |                0 |
0 |             0 |            0 |             0 |
```

```
| 38 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 39 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 40 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 41 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 42 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 43 |             1 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 44 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 45 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 46 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 47 |             3 |                   1 |                     0 |                  0 |                   0 |
0 |                 1 |                 0 |                   0 |
| 48 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 49 |             3 |                   1 |                     0 |                  0 |                   0 |
0 |                 1 |                 0 |                   0 |
| 50 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 51 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 52 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 53 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 54 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 55 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 56 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 57 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 58 |             1 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 59 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 60 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 61 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 62 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
| 63 |             2 |                   0 |                     0 |                  0 |                   0 |
0 |                 0 |                 0 |                   0 |
+----+---------------+---------------------+-----------------------+--------------------+-----
----------------------+------------------+-----------------+------------------+
64 rows in set (0.00 sec)
```

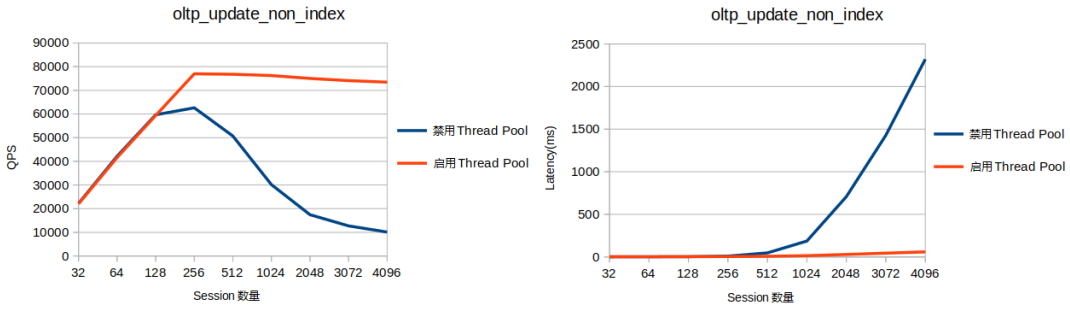
参数说明如下。

参数	说明
ID	线程池ID。
THREAD_COUNT	线程池中的线程数。
ACTIVE_THREAD_COUNT	线程池中的活跃线程数。
WAITING_THREAD_COUNT	线程池中正在等待磁盘I/O、事务提交的线程数。
DUMP_THREAD_COUNT	线程池中DUMP类长连接数量。
SLOW_THREAD_TIMEOUT_COUNT	线程池中被标记超时的线程数。
CONNECTION_COUNT	线程池中已建立的用户连接数。
LOW_QUEUE_COUNT	线程池中低优先级队列中等待的请求数。
HIGH_QUEUE_COUNT	线程池中高优先级队列中等待的请求数。

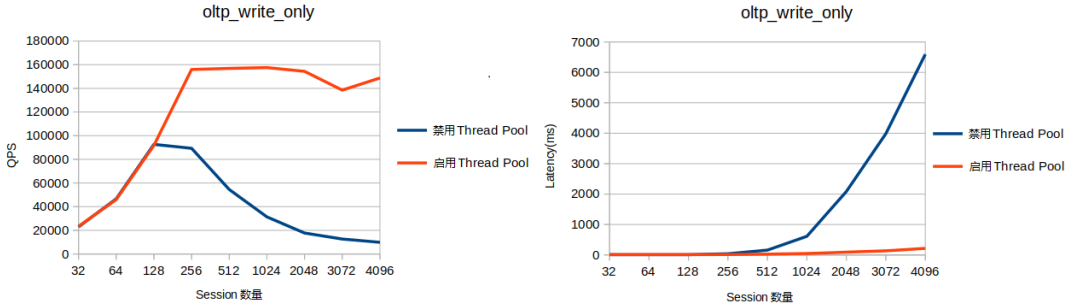
Sysbench测试

如下是开启线程池和不开启线程池的性能对比。从测试结果可以看出线程池在高并发的情况下有着明显的性能优势。

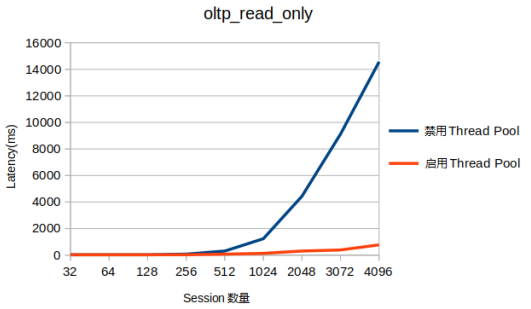
OLTP无索引更新测试



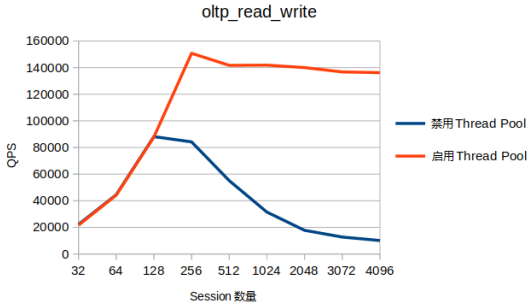
OLTP只写测试

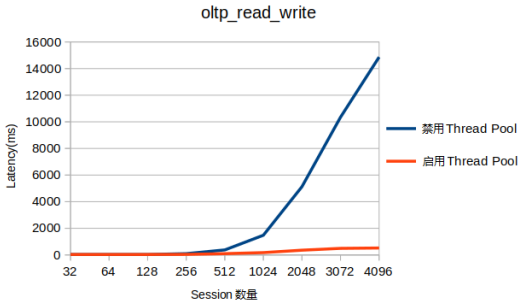


OLTP只读测试



OLTP读写测试





4.5. 性能监控

4.5.1. Performance Agent

Performance Agent是PolarDB提供的一种更加便捷的性能数据统计方案。通过PolarDB MySQL引擎插件的方式，实现PolarDB MySQL引擎集群中的节点内部各项性能数据的采集与统计。

前提条件

PolarDB集群版本需为PolarDB MySQL引擎8.0版本且Revision version为8.0.2.1.0或以上，您可以参见查询版本号确认集群版本。

背景信息

Performance Agent在information_schema系统库下新增了一张内存表PERF_STATISTICS，用于统计最近一段时间的性能数据，您可以直接查询该表获取相关指标的性能数据。

参数说明

Performance Agent功能相关参数说明如下：

参数	说明
performance_agent_enabled	是否开启Performance Agent功能。取值为ON或OFF。默认值为ON。
performance_agent_interval	性能数据采集间隔。单位为秒，取值范围为1~60，默认值为1。

参数	说明
performance_agent_perfstat_volume_size	PERF_STATISTICS表的最大数据条数。默认值为3600（即当采样间隔为1秒时，保存最近1小时的性能数据）。

表结构说明

PERF_STATISTICS内存表的结构如下：

```
CREATE TEMPORARY TABLE `PERF_STATISTICS` (
  `TIME` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `PROCS_MEM_USAGE` double NOT NULL DEFAULT '0',
  `PROCS_CPU_RATIO` double NOT NULL DEFAULT '0',
  `PROCS_IOPS` double NOT NULL DEFAULT '0',
  `PROCS_IO_READ_BYTES` bigint(21) NOT NULL DEFAULT '0',
  `PROCS_IO_WRITE_BYTES` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_CONN_ABORT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_CONN_CREATED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_USER_CONN_COUNT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_CONN_RUNNING` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LOCK_IMMEDIATE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LOCK_WAITED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_INSERT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_UPDATE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_DELETE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_SELECT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_COMMIT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_ROLLBACK` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_PREPARE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LONG_QUERY` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TCACHE_GET` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_TCACHE_MISS` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_TMPFILE_CREATED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TMP_TABLES` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TMP_DISKTABLES` int(11) NOT NULL DEFAULT '0',
  `MYSQL_SORT_MERGE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_SORT_ROWS` int(11) NOT NULL DEFAULT '0',
  `MYSQL_BYTES_RECEIVED` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_BYTES_SENT` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_BINLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `MYSQL_IOLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `MYSQL_RELAYLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `EXTRA` json NOT NULL DEFAULT 'null'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

列名	说明
TIME	时间（UTC），格式为yyyy-MM-dd HH:mm:ss。
PROCS_MEM_USAGE	物理内存使用量，单位为Byte。

列名	说明
PROCS_CPU_RATIO	CPU使用率。
PROCS_IOPS	系统IO调用次数。
PROCS_IO_READ_BYTES	IO读取数据量，单位为Byte。
PROCS_IO_WRITE_BYTES	IO写入数据量，单位为Byte。
MYSQL_CONN_ABORT	断开连接数。
MYSQL_CONN_CREATED	新建连接数。
MYSQL_USER_CONN_COUNT	当前总的用户连接数。
MYSQL_CONN_RUNNING	当前活跃连接数。
MYSQL_LOCK_IMMEDIATE	当前锁占用数。
MYSQL_LOCK_WAITED	当前锁等待数。
MYSQL_COM_INSERT	插入语句数。
MYSQL_COM_UPDATE	更新语句数。
MYSQL_COM_DELETE	删除语句数。
MYSQL_COM_SELECT	查询语句数。
MYSQL_COM_COMMIT	事务提交数（显式提交）。
MYSQL_COM_ROLLBACK	事务回滚数。
MYSQL_COM_PREPARE	预处理语句数。
MYSQL_LONG_QUERY	慢查询数。
MYSQL_TCACHE_GET	缓存表命中数。
MYSQL_TCACHE_MISS	缓存表未命中数。
MYSQL_TMPFILE_CREATED	临时文件创建数。
MYSQL_TMP_TABLES	临时表创建数。
MYSQL_TMP_DISKTABLES	临时磁盘表创建数。
MYSQL_SORT_MERGE	合并排序次数。
MYSQL_SORT_ROWS	排序行数。
MYSQL_BYTES_RECEIVED	接收数据量，单位为Byte。

列名	说明
MYSQL_BYTES_SENT	发送数据量，单位为Byte。
MYSQL_BINLOG_OFFSET	产生的Binlog文件大小，单位为Byte。
MYSQL_IOLOG_OFFSET	主库发送的Binlog文件大小，单位为Byte。
MYSQL_RELAYLOG_OFFSET	从库应用的Binlog文件大小，单位为Byte。
EXTRA	InnoDB统计信息。EXTRA包含多个字段，为JSON格式。详细字段介绍请参见下方EXTRA字段说明。 **说明** InnoDB统计信息的指标项与 `SHOW STATUS` 命令显示的值相同。

EXTRA字段说明

字段	说明
INNODB_TRX_CNT	事务数。
INNODB_DATA_READ	读取数据量，单位为Byte。
INNODB_IBUF_SIZE	合并记录页数。
INNODB_LOG_WAITS	Log写入等待次数。
INNODB_MAX_PURGE	清除事务数。
INNODB_N_WAITING	锁等待数。
INNODB_ROWS_READ	读取数据行数。
INNODB_LOG_WRITES	日志写次数。
INNODB_IBUF_MERGES	合并次数。
INNODB_DATA_WRITTEN	写入数据量，单位为Byte。
INNODB_DBLWR_WRITES	双写操作写入次数。
INNODB_IBUF_SEGSIZE	当前插入缓冲大小。
INNODB_ROWS_DELETED	删除数据行数。
INNODB_ROWS_UPDATED	更新数据行数。
INNODB_COMMIT_TRXCNT	提交事务数。
INNODB_IBUF_FREELIST	空闲列表长度。

字段	说明
INNODB_MYSQL_TRX_CNT	MySQL事务数。
INNODB_ROWS_INSERTED	插入数据行数。
INNODB_ACTIVE_TRX_CNT	活跃事务数。
INNODB_OS_LOG_WRITTEN	日志文件写入量，单位为Byte。
INNODB_ACTIVE_VIEW_CNT	活跃视图数。
INNODB_RSEG_HISTORY_LEN	TRX_RSEG_HISTORY表长度。
INNODB_AVG_COMMIT_TRXTIME	平均事务提交时间。
INNODB_MAX_COMMIT_TRXTIME	最长事务提交时间。
INNODB_DBLWR_PAGES_WRITTEN	双写操作完成写入次数。

使用方法

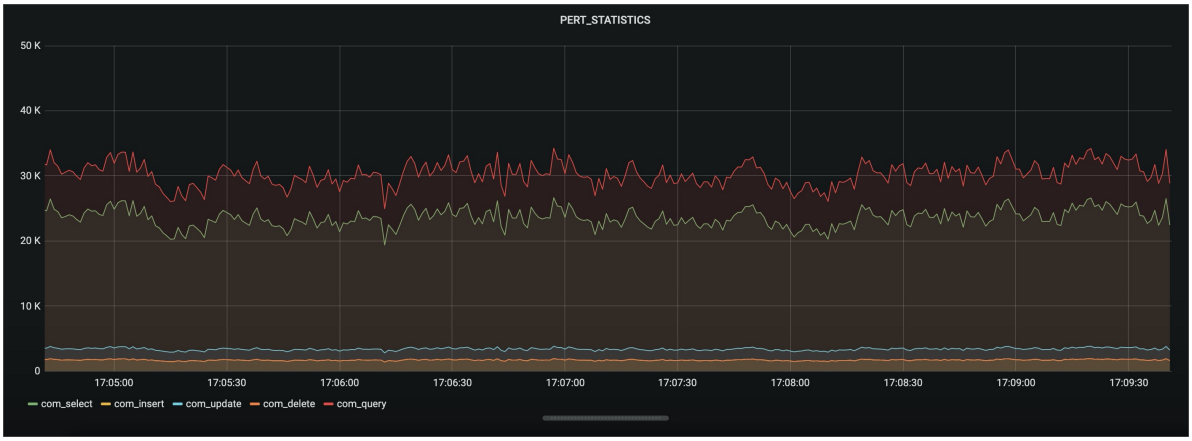
- 您可以参见如下示例直接查询系统表，获取性能数据。
  - 查询最近30秒的CPU和内存使用情况，示例如下：

```
MySQL> select TIME, PROCS_MEM_USAGE, PROCS_CPU_RATIO from information_schema.PERF_STATI
STICS order by time DESC limit 30;
+---------------------+-----------------+-----------------+
| TIME                | PROCS_MEM_USAGE | PROCS_CPU_RATIO |
+---------------------+-----------------+-----------------+
| 2020-02-27 11:15:36 |       857812992 |           18.55 |
| 2020-02-27 11:15:35 |       857808896 |           18.54 |
| 2020-02-27 11:15:34 |       857268224 |           19.64 |
| 2020-02-27 11:15:33 |       857268224 |           21.06 |
| 2020-02-27 11:15:32 |       857264128 |           20.39 |
| 2020-02-27 11:15:31 |       857272320 |           20.32 |
| 2020-02-27 11:15:30 |       857272320 |           21.35 |
| 2020-02-27 11:15:29 |       857272320 |            28.8 |
| 2020-02-27 11:15:28 |       857268224 |           29.08 |
| 2020-02-27 11:15:27 |       857268224 |           26.92 |
| 2020-02-27 11:15:26 |       857268224 |           23.84 |
| 2020-02-27 11:15:25 |       857264128 |           13.76 |
| 2020-02-27 11:15:24 |       857264128 |           15.12 |
| 2020-02-27 11:15:23 |       857264128 |           14.76 |
| 2020-02-27 11:15:22 |       857264128 |           15.38 |
| 2020-02-27 11:15:21 |       857260032 |           13.23 |
| 2020-02-27 11:15:20 |       857260032 |           12.75 |
| 2020-02-27 11:15:19 |       857260032 |           12.17 |
| 2020-02-27 11:15:18 |       857255936 |           13.22 |
| 2020-02-27 11:15:17 |       857255936 |           20.51 |
| 2020-02-27 11:15:16 |       857255936 |           28.74 |
| 2020-02-27 11:15:15 |       857251840 |           29.85 |
| 2020-02-27 11:15:14 |       857251840 |           29.31 |
| 2020-02-27 11:15:13 |       856981504 |           28.85 |
| 2020-02-27 11:15:12 |       856981504 |           29.19 |
| 2020-02-27 11:15:11 |       856977408 |           29.12 |
| 2020-02-27 11:15:10 |       856977408 |           29.32 |
| 2020-02-27 11:15:09 |       856977408 |            29.2 |
| 2020-02-27 11:15:08 |       856973312 |           29.36 |
| 2020-02-27 11:15:07 |       856973312 |           28.79 |
+---------------------+-----------------+-----------------+
30 rows in set (0.08 sec)
```

- 查询最近30秒的InnoDB读写的数据行数，示例如下：

```
MySQL> select TIME, EXTRA->'$.INNODB_ROWS_READ', EXTRA->'$.INNODB_ROWS_INSERTED' from i
nformation_schema.PERF_STATISTICS order by time DESC limit 30;
+---------------------+-----------------------------+--------------------------------+
| TIME                | EXTRA->'$.INNODB_ROWS_READ' | EXTRA->'$.INNODB_ROWS_INSERTED' |
+---------------------+-----------------------------+--------------------------------+
| 2020-02-27 11:22:17 | 39209                       | 0                              |
| 2020-02-27 11:22:16 | 36098                       | 0                              |
| 2020-02-27 11:22:15 | 38035                       | 0                              |
| 2020-02-27 11:22:14 | 37384                       | 0                              |
| 2020-02-27 11:22:13 | 38336                       | 0                              |
| 2020-02-27 11:22:12 | 33946                       | 0                              |
| 2020-02-27 11:22:11 | 36301                       | 0                              |
| 2020-02-27 11:22:10 | 36835                       | 0                              |
| 2020-02-27 11:22:09 | 36900                       | 0                              |
| 2020-02-27 11:22:08 | 36402                       | 0                              |
| 2020-02-27 11:22:07 | 39672                       | 0                              |
| 2020-02-27 11:22:06 | 39316                       | 0                              |
| 2020-02-27 11:22:05 | 37830                       | 0                              |
| 2020-02-27 11:22:04 | 36396                       | 0                              |
| 2020-02-27 11:22:03 | 34820                       | 0                              |
| 2020-02-27 11:22:02 | 37350                       | 0                              |
| 2020-02-27 11:22:01 | 39463                       | 0                              |
| 2020-02-27 11:22:00 | 38419                       | 0                              |
| 2020-02-27 11:21:59 | 37673                       | 0                              |
| 2020-02-27 11:21:58 | 35117                       | 0                              |
| 2020-02-27 11:21:57 | 36140                       | 0                              |
| 2020-02-27 11:21:56 | 37592                       | 0                              |
| 2020-02-27 11:21:55 | 39765                       | 0                              |
| 2020-02-27 11:21:54 | 35553                       | 0                              |
| 2020-02-27 11:21:53 | 35882                       | 0                              |
| 2020-02-27 11:21:52 | 37061                       | 0                              |
| 2020-02-27 11:21:51 | 40699                       | 0                              |
| 2020-02-27 11:21:50 | 39608                       | 0                              |
| 2020-02-27 11:21:49 | 39317                       | 0                              |
| 2020-02-27 11:21:48 | 37413                       | 0                              |
+---------------------+-----------------------------+--------------------------------+
30 rows in set (0.08 sec)
```

- 对接性能监控平台（如使用Grafana可视化工具），实现实时监控。



4.6. 其它

4.6.1. Fast Query Cache

针对MySQL原生Query Cache的不足，PolarDB进行重新设计和全新实现，推出了Fast Query Cache，能够有效提高数据库查询性能。

使用限制

PolarDB集群版本需为以下版本之一：

- PolarDB MySQL引擎8.0版本且Revision version为8.0.1.1.5或以上；
- PolarDB MySQL引擎5.7版本且Revision version为5.7.1.0.15或以上；
- PolarDB MySQL引擎5.6版本且Revision version为5.6.1.0.29或以上。

> **说明** 您可以参见查询版本号确认集群版本。

问题与优化

查询缓存（Query Cache）是为了提高查询性能而实现的一种缓存策略，它通过节约CPU资源来达到查询加速的目标，是一项非常实用的技术。其基本思想是：对于每个符合条件的查询语句，直接对结果集进行缓存。当该结果集被再次查询命中时，直接从缓存中读取对应的结果集并返回，不需要经历SQL的分析、优化、执行等复杂的过程。

MySQL原生Query Cache在设计和实现上存在较多的严重问题，具体如下：

- 并发处理较差，在多核情况下，并发度越高性能降低可能越严重。
- 内存管理较差，内存利用率低且回收不及时，造成内存浪费。
- 当缓存命中率较低时，性能无提升甚至严重降低。

基于以上问题，MySQL原生Query Cache没有得到广泛应用，在最新版的MySQL 8.0中，取消了此功能。PolarDB对Query Cache进行重新设计和全新实现，进行了如下优化：

- 优化并发控制

  取消全局锁同步机制，采用无锁机制，重新设计并发场景下的同步问题，能够充分利用多核的处理能力，保证高并发场景下的性能。

- 优化内存管理

取消内存预分配机制，采用更加灵活的动态内存分配机制，及时回收无效的内存，保证内存的真实利用率。

- 优化缓存机制

  动态检测缓存利用率，实时调整缓存策略，解决命中率偏低或读写混合等场景下的性能降低问题。

相比MySQL原生Query Cache，您可以在不同的业务场景中开启Fast Query Cache以提高查询性能。

开启Fast Query Cache

PolarDB已经为不同的集群规格配置了不同的Fast Query Cache内存空间。您仅需要调整loose_query_cache_type参数即可开启Fast Query Cache功能，关于如何修改参数值，请参见设置集群参数。

性能比较

在相同场景下，分别测试QC-OFF（关闭Query Cache）和PolarDB-QC（开启Fast Query Cache）的QPS。

- 测试环境
  - 一个规格为8核64 GB的PolarDB MySQL引擎8.0集群版。
  - Query Cache内存为4 GB。
- 测试工具

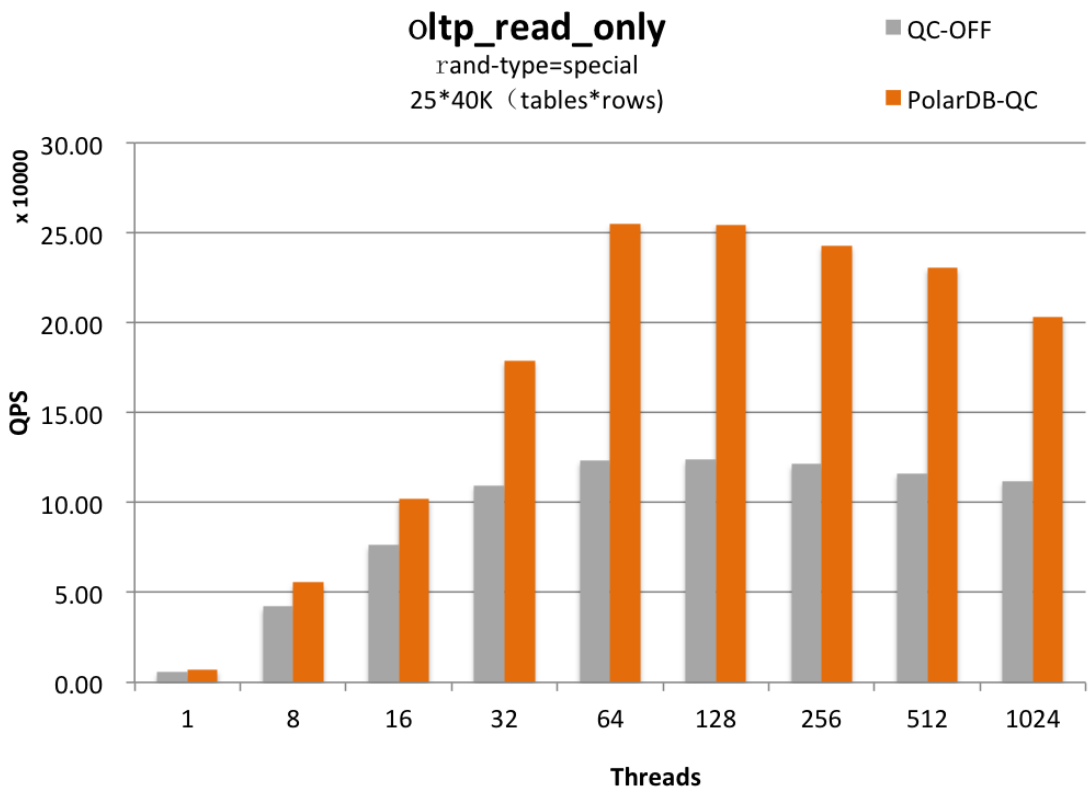
  Sysbench
- 测试数据量
  - 25张表，每张表4万条记录。
  - 25张表，每张表40万条记录。
- 测试用例

  使用以下Sysbench内置用例，分别在rand-type=special和uniform条件下进行测试。
  - oltp_read_only
  - oltp_point_select
  - oltp_read_write
- 测试结果及说明

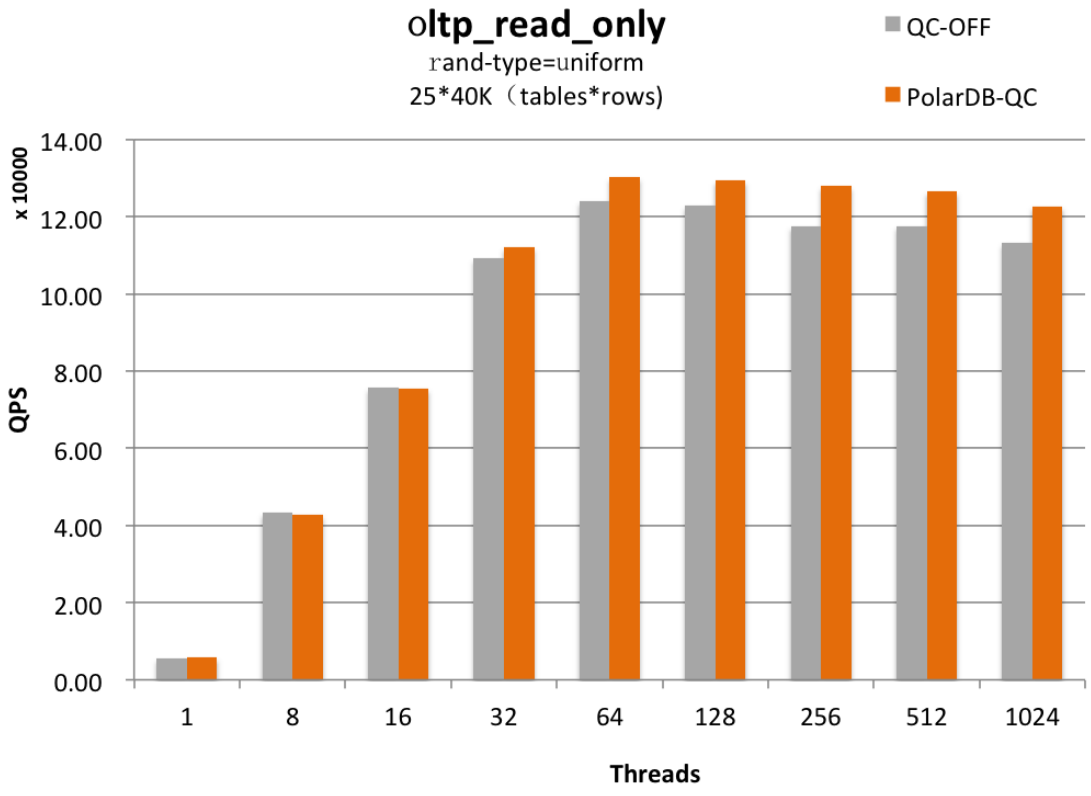
  从以下测试中可以看到，Fast Query Cache在高并发场景较高命中率下性能提升非常明显。在Case1、Case3、Case4、Case5、Case6、Case7等场景中，使用了Fast Query Cache后的缓存命中率范围在63%~99%之间，最大QPS提升范围在53%~106%之间。Fast Query Cache的内存利用率很高，Case7在大数据量的point select场景中，其命中率依然达到了99%，性能提升较明显。同时在低命中率场景中，Fast Query Cache对并发性能的影响很小，基本在3%以内。对于高并发读写混合场景，性能影响也在2%以内。

  > **说明** 本文测试场景中的所有测试数据仅针对测试集群中的主节点。

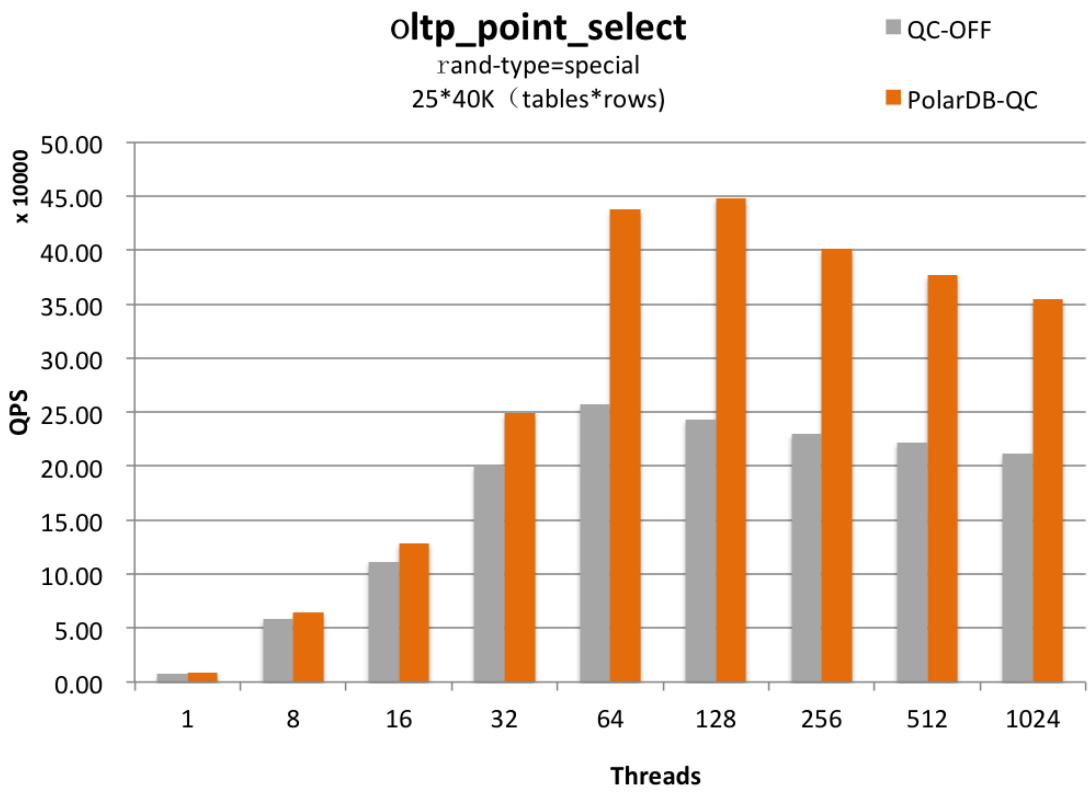
- Case1：25 x 40k（tables x rows），rand-type = special oltp_read_only



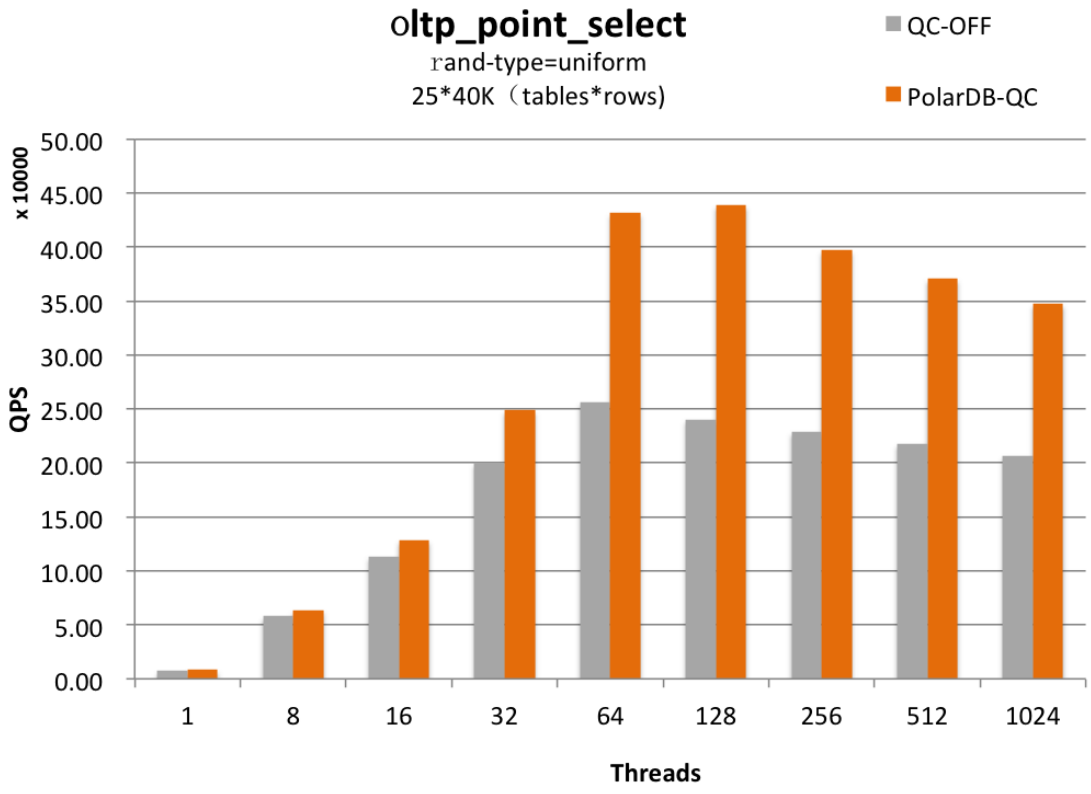
- Case2：25 x 40k（tables x rows），rand-type = uniform oltp_read_only



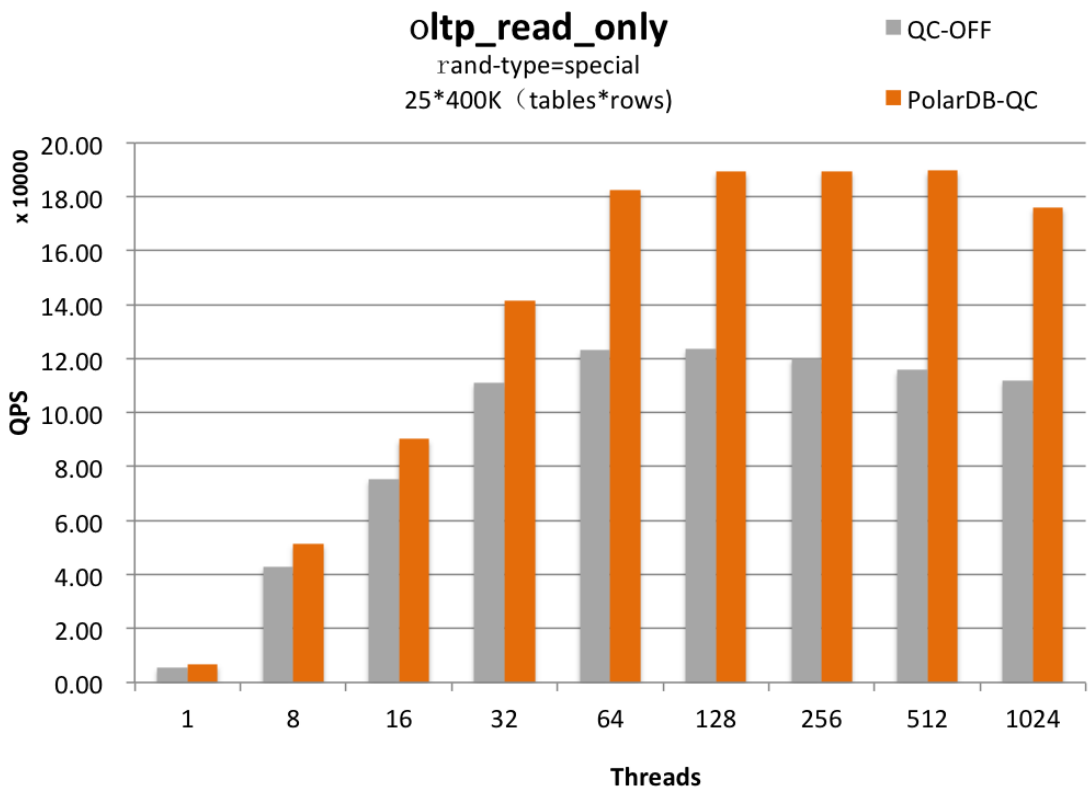
- Case3：25 x 40k（tables x rows），rand-type = special oltp_point_select



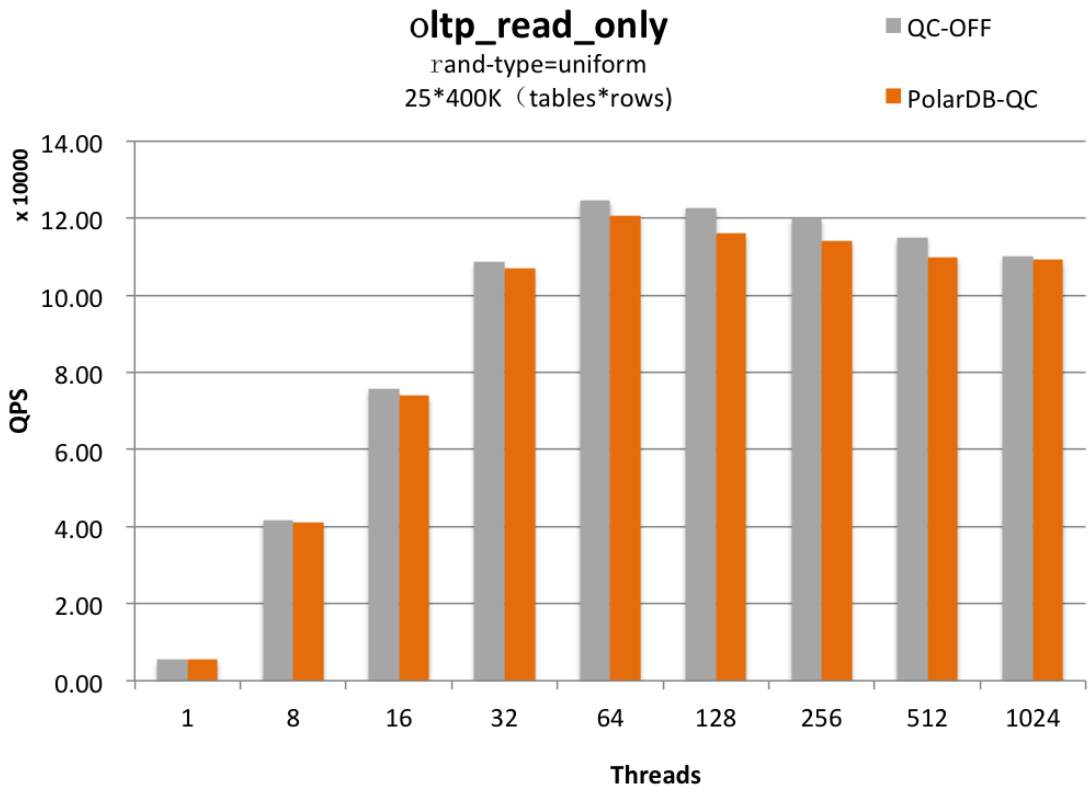
- Case4：25 x 40k（tables x rows），rand-type = uniform oltp_point_select



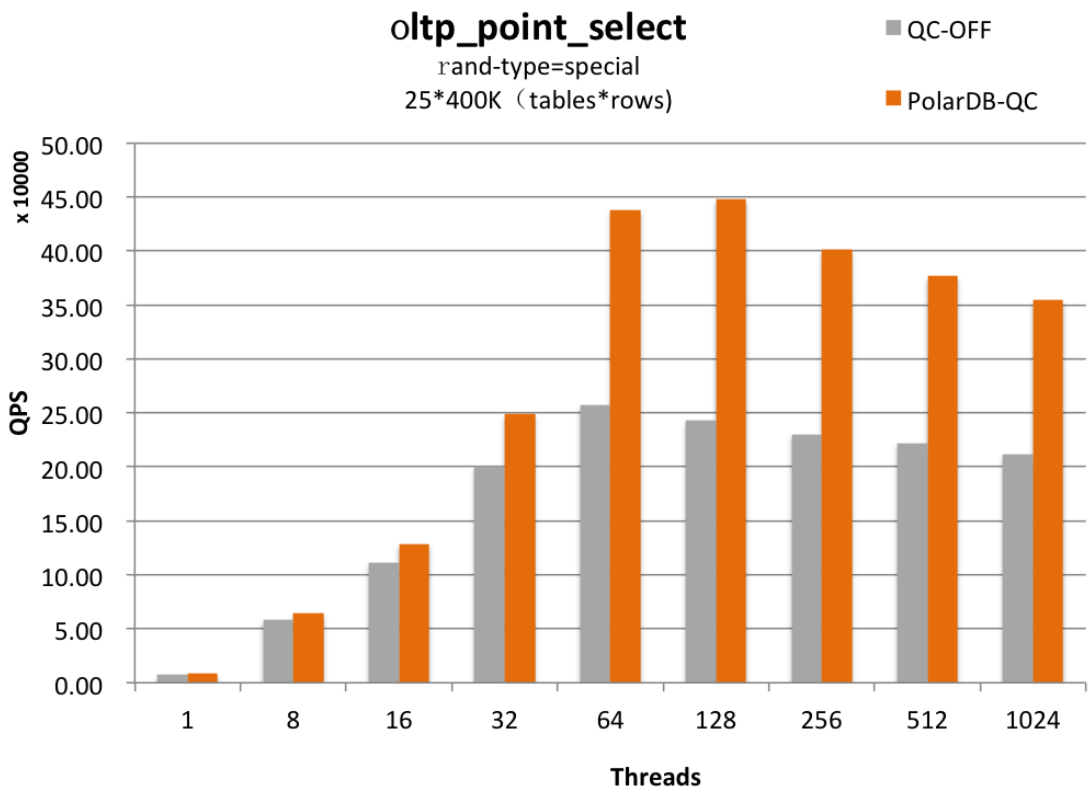
- Case5：25 x 400k（tables x rows），rand-type = special oltp_read_only



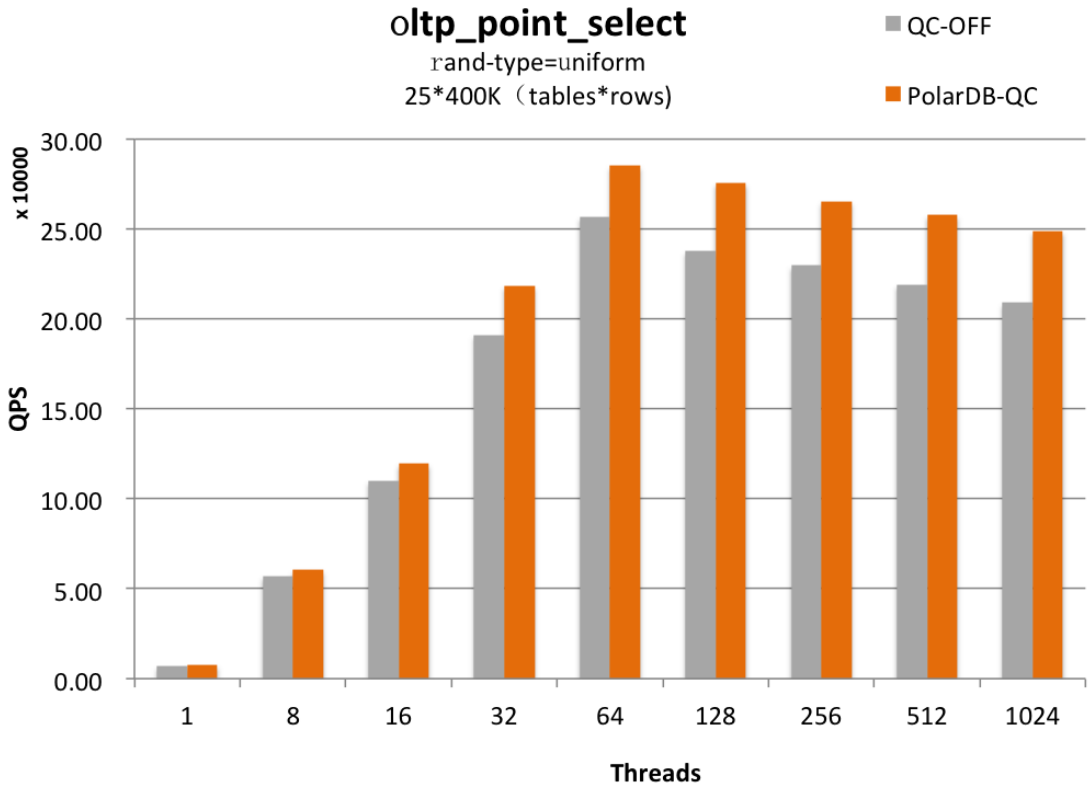
- Case6：25 x 400k（tables x rows），rand-type = uniform oltp_read_only



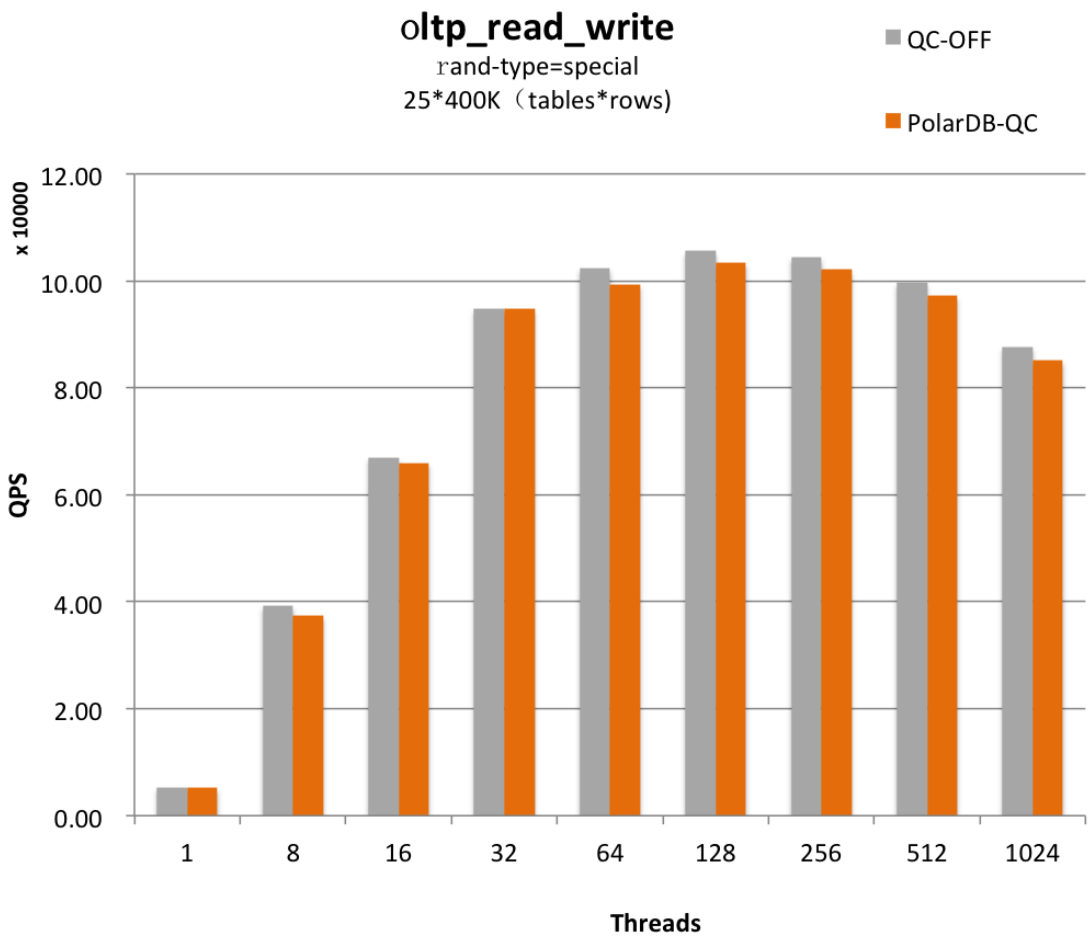
- Case7：25 x 400k（tables x rows），rand-type = special oltp_point_select



- Case8：25 x 400k（tables x rows），rand-type = uniform oltp_point_select



- Case9：25 x 400k（tables x rows）， rand-type = special oltp_read_write



实践指南

- 适用场景指南
  - Fast Query Cache主要作用是提高读操作性能，建议在读多写少的场景下开启，或者使用SQL_CACHE关键字针对读多写少的表开启。如果写多读少，数据的更新非常频繁，查询性能可能会降低2%左右。
  - Fast Query Cache内存利用率很高，适用于更新很少但访问量很大的点查场景中，大幅提升并发访问性能。
- 缓存使用方式（loose_query_cache_type）

  Fast Query Cache完全兼容原生Query Cache的使用方法，可通过loose_query_cache_type参数控制缓存。

参数名	取值	说明
loose_query_cache_type	*OFF*（默认值）	禁用Fast Query Cache。
	ON	默认在查询中使用Fast Query Cache功能，但可通过SQL_NO_CACHE关键字跳过缓存。
	DEMAND	默认在查询中不使用Fast Query Cache功能，但可通过SQL_CACHE关键字对特定语句使用缓存。

> 说明
>
> loose_query_cache_type参数支持会话级修改，您可以参考如下建议并根据真实业务场景进行灵活设置：
>
> - 对于更新频繁、写多读少等不适合Query Cache的场景，应将loose_query_cache_type全局设置为*OFF*。
> - 对于较多重复查询、缓存命中率较高或者较多重复的慢查询场景，可以将loose_query_cache_type全局设置为*ON*。
> - 对于仅存在少量重复查询的场景，可以将loose_query_cache_type设置为*DEMAND*，仅对指定的语句，通过SQL_CACHE关键字使用Fast Query Cache。

- 缓存租约时间 (query_cache_lease_time)

  Fast Query Cache会动态回收查询缓存的内存，降低内存的使用量。对于未失效的查询缓存，若在query_cache_lease_time时间（单位为秒）内没有被任何查询命中，则会在超过缓存租约时间后被释放，所对应的内存将被回收。该参数默认值为3600秒，即1小时。

4.6.2. Statement Outline

生产环境中，SQL语句的执行计划经常发生改变，导致数据库不稳定。PolarDB利用Optimizer Hint和Index Hint让MySQL稳定执行计划，该方法称为Statement Outline，并提供了工具包（DBMS_OUTLN）方便您快捷使用。本文将介绍如何使用和管理Statement Outline。

前提条件

PolarDB集群版本需为如下版本之一：

- PolarDB MySQL引擎5.7版本且Revision version为5.7.1.0.2或以上。
- PolarDB MySQL引擎8.0版本且Revision version为8.0.1.1.1或以上。

您可以参见查询版本号确认集群版本。

功能设计

Statement Outline支持官方MySQL 8.0的所有Hint类型，分为如下两类：

- Optimizer Hint

  根据作用域和Hint对象，分为Global level hint、Table-level hint、Index-level hint、Join-order hint等，详情请参见Optimizer Hints。

- Index Hint

  根据Index Hint的类型和范围进行分类，详情请参见Index Hints。

Statement Outline表介绍

PolarDB内置了一个系统表（outline）保存Hint，系统启动时会自动创建该表，无需您手动创建。该系统表的创建语句如下所示：

```
CREATE TABLE `mysql`.`outline` (
  `Id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Schema_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Digest` varchar(64) COLLATE utf8_bin NOT NULL,
  `Digest_text` longtext COLLATE utf8_bin,
  `Type` enum('IGNORE INDEX','USE INDEX','FORCE INDEX','OPTIMIZER') CHARACTER SET utf8 COLL
ATE utf8_general_ci NOT NULL,
  `Scope` enum('','FOR JOIN','FOR ORDER BY','FOR GROUP BY') CHARACTER SET utf8 COLLATE utf8
_general_ci DEFAULT '',
  `State` enum('N','Y') CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL DEFAULT 'Y',
  `Position` bigint(20) NOT NULL,
  `Hint` text COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`Id`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0 COMMENT='Statement outline'
```

参数说明如下。

参数	说明
Id	Outline ID。
Schema_name	数据库名称。
Digest	Digest_text进行hash计算得到的64字节的hash字符串，详情请参见STATEMENT_DIGEST()。
Digest_text	SQL语句的特征。
Type	• Optimizer Hint中，Hint类型的取值为OPTIMIZER。 • Index Hint中，Hint类型的取值为USE INDEX、FORCE INDEX或IGNORE INDEX。
Scope	仅Index Hint需要提供，分为如下三类： • FOR GROUP BY • FOR ORDER BY • FOR JOIN 说明 空串表示所有类型的Index Hint。
State	本规则是否启用，取值范围为N（No）和Y（Yes），默认为Y。
Position	• Optimizer Hint中，Position表示Query Block，因为所有的Optimizer Hint必须作用到Query Block上，Position从1开始，Hint作用在语句的第几个关键字上，Position就是几。 • Index Hint中，Position表示表的位置， 也是从1开始，Hint作用在第几个表上，Position就是几。

参数	说明
Hint	• Optimizer Hint中，Hint表示完整的Hint字符串，例如 `/*+ MAX_EXECUTION_TIME(1000) */` 。 • Index Hint中，Hint表示索引名字的列表， 例如 `ind_1,ind_2` 。

管理Statement Outline

为了便捷地管理Statement Outline，PolarDB在DBMS_OUTLN中定义了六个本地存储规则，详细说明如下：

- add_optimizer_outline

  增加Optimizer Hint。命令如下：

  ```
  dbms_outln.add_optimizer_outline('<Schema_name>','<Digest>','<query_block>','<hint>','<query>');
  ```

  示例：

  ```
  CALL DBMS_OUTLN.add_optimizer_outline("outline_db", '', 1, '/*+ MAX_EXECUTION_TIME(1000) */',
                                        "select * from t1 where id = 1");
  ```

  > **说明** Digest和Query（原始SQL语句）可以任选其一。如果填写Query，DBMS_OUTLN会计算Digest和Digest_text。

- add_index_outline

  增加Index Hint。命令如下：

  ```
  dbms_outln.add_index_outline('<Schema_name>','<Digest>',<Position>,'<Type>','<Hint>','<Scope>','<Query>');
  ```

  > **说明** Digest和Query（原始SQL语句）可以任选其一。如果填写Query，DBMS_OUTLN会计算Digest和Digest_text。

  示例：

  ```
  call dbms_outln.add_index_outline('outline_db', '', 1, 'USE INDEX', 'ind_1', '',
                                    "select * from t1 where t1.col1 =1 and t1.col2 ='xpchild'
  ");
  ```

- preview_outline

  查看匹配Statement Outline的情况，可用于手动验证。命令如下：

  ```
  dbms_outln.preview_outline('<Schema_name>','<Query>');
  ```

  示例：

```
mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 a
nd t1.col2 ='xpchild'");
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| SCHEMA     | DIGEST                                                               | BLOCK_T
YPE | BLOCK_NAME | BLOCK | HINT                |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE
| t1         |     1 | USE INDEX (`ind_1`) |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
1 row in set (0.00 sec)
```

- show_outline

  展示Statement Outline在内存中命中的情况。命令如下：

  ```
  dbms_outln.show_outline();
  ```

  示例：

```
mysql> call dbms_outln.show_outline();
+------+------------+-----------------------------------------------------------------+-
----------+-------+------+-------------------------------------------------------------+------+
----------+-------------------------------------------------------------------------------
-------+
| ID   | SCHEMA     | DIGEST                                                                |
TYPE      | SCOPE | POS  | HINT                                                        | HIT  |
OVERFLOW | DIGEST_TEXT
|
+------+------------+-----------------------------------------------------------------+-
----------+-------+------+-------------------------------------------------------------+------+
----------+-------------------------------------------------------------------------------
-------+
|   33 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 |
OPTIMIZER |       |    1 | /*+ SET_VAR(foreign_key_checks=OFF) */                      |    1 |
0 | SELECT * FROM `t1` WHERE `id` = ?                                                   |
|   32 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 |
OPTIMIZER |       |    1 | /*+ MAX_EXECUTION_TIME(1000) */                             |    2 |
0 | SELECT * FROM `t1` WHERE `id` = ?                                                   |
|   34 | outline_db | d4dcef634a4a664518e5fb8a21c6ce9b79fccb44b773e86431eb67840975b649 |
OPTIMIZER |       |    1 | /*+ BNL(t1,t2) */                                           |    1 |
0 | SELECT `t1` . `id` , `t2` . `id` FROM `t1` , `t2`                                   |
|   35 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f |
OPTIMIZER |       |    2 |  /*+ QB_NAME(subq1) */                                      |    2 |
0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )               |
|   36 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f |
OPTIMIZER |       |    1 | /*+ SEMIJOIN(@subq1 MATERIALIZATION, DUPSWEEDOUT) */  |    2 |
0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )               |
|   30 | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c |
USE INDEX |       |    1 | ind_1                                                       |    3 |
0 | SELECT * FROM `t1` WHERE `t1` . `col1` = ? AND `t1` . `col2` = ?                    |
|   31 | outline_db | 33c71541754093f78a1f2108795cfb45f8b15ec5d6bff76884f4461fb7f33419 |
USE INDEX |       |    2 | ind_2                                                       |    1 |
0 | SELECT * FROM `t1` , `t2` WHERE `t1` . `col1` = `t2` . `col1` AND `t2` . `col2` = ? |
+------+------------+-----------------------------------------------------------------+-
----------+-------+------+-------------------------------------------------------------+------+
----------+-------------------------------------------------------------------------------
-------+
7 rows in set (0.00 sec)
```

关于HIT和OVERFLOW的说明如下。

参数	说明
HIT	该Statement Outline命中的次数。
OVERFLOW	该Statement Outline没有找到Query Block或相应的表的次数。

- del_outline

  删除内存和表中的某一条Statement Outline。命令如下：

```
dbms_outln.del_outline(<Id>);
```

示例：

```
mysql> call dbms_outln.del_outline(32);
```

> **说明** 如果删除的规则不存在，系统会报相应的警告，您可以使用 `show warnings;` 查看警告内容。

```
mysql> call dbms_outln.del_outline(1000);
Query OK, 0 rows affected, 2 warnings (0.00 sec)
mysql> show warnings;
+---------+------+-----------------------------------------------+
| Level   | Code | Message                                       |
+---------+------+-----------------------------------------------+
| Warning | 7521 | Statement outline 1000 is not found in table |
| Warning | 7521 | Statement outline 1000 is not found in cache |
+---------+------+-----------------------------------------------+
2 rows in set (0.00 sec)
```

- flush_outline

  如果您直接操作了表outline修改Statement Outline，您需要使用如下命令让Statement Outline重新生效：

```
dbms_outln.flush_outline();
```

示例：

```
mysql> update mysql.outline set Position = 1 where Id = 18;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> call dbms_outln.flush_outline();
Query OK, 0 rows affected (0.01 sec)
```

功能测试

您可以使用以下两种方式中的任意一种验证Statement Outline是否有效果。

- 通过preview_outline进行预览。

```
mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 a
nd t1.col2 ='xpchild'");
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| SCHEMA     | DIGEST                                                               | BLOCK_T
YPE | BLOCK_NAME | BLOCK | HINT                |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE
| t1         |     1 | USE INDEX (`ind_1`) |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
1 row in set (0.01 sec)
```

- 通过explain查看。

```
mysql> explain select * from t1 where t1.col1 =1 and t1.col2 ='xpchild';
+----+-------------+-------+------------+------+---------------+-------+---------+-------
+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref
| rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+-------+---------+-------
+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ref  | ind_1         | ind_1 | 5       | const
|    1 |   100.00 | Using where |
+----+-------------+-------+------------+------+---------------+-------+---------+-------
+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
mysql> show warnings;
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select `outline_db`.`t1`.`id` AS `id`,`outline_db`.`t1`.`
col1` AS `col1`,`outline_db`.`t1`.`col2` AS `col2` from `outline_db`.`t1` USE INDEX (`ind
_1`) where ((`outline_db`.`t1`.`col1` = 1) and (`outline_db`.`t1`.`col2` = 'xpchild')) |
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

4.6.3. 表回收站

由于DDL语句无法回滚，开发或运维人员如果误操作（例如DROP TABLE）可能会导致数据丢失。PolarDB支持回收站（Recycle Bin）功能，用于将删除的表临时转移到回收站，您可以自定义删除表的保留时间，方便您找回数据。

前提条件

PolarDB集群版本需为PolarDB MySQL引擎8.0版本且Revision version为8.0.1.1.2或以上，您可以参见查询版本号确认集群版本。

Recycle Bin介绍

- 回收和清理机制
  - 回收机制

    当执行 `DROP TABLE` 语句来删除数据表，或执行 `DROP DATABASE` 语句来删除数据库时，PolarDB只会保留相关的表对象，并将表对象移动到专门的Recycle Bin目录中。其它对象的删除策略如下：

    - 如果是与表无关的对象，根据操作语句决定是否保留，不做回收。
    - 如果是可能会修改表数据的表附属对象，做删除处理，例如Trigger和Foreign key。 但Column statistics不会被删除，而是随表进入回收站。

- 清理机制

  回收站会启动一个后台线程，来异步清理超过recycle_bin_retention时间的表对象。在清理回收站表的时候，如果遇到大表，会再启动一个后台线程异步删除大表。

- 权限

  PolarDB集群启动时，会初始化一个名为__recycle_bin__的数据库，作为回收站使用的专有数据库。__recycle_bin__是系统级数据库，您无法直接进行修改和删除。

  对于回收站内的表，虽然您无法直接执行 `DROP TABLE` 语句，但是可以使用 `CALL DBMS_RECYCLE.purge_table('table name');` 进行清理。

  > **说明** 执行清理操作的账号在原表和回收站表都需要具有DROP权限。

- 回收站表命名规则

  Recycle Bin会从不同的数据库回收到统一的__recycle_bin__数据库中，因此定义了如下命名格式，用来保证目标表表名的唯一性：

  ```
  "__" + <Storage Engine> + <SE private id>
  ```

  参数说明如下：

参数	说明
Storage Engine	存储引擎名称。
SE private id	存储引擎为每一个表生成的唯一值。例如在InnoDB引擎中就是 `table id` 。

- 独立回收

  例如我们可以在主节点上设置回收，保留7天；在只读节点上设置回收，保留14天。

  > **说明** 回收站保留周期不同，将导致集群的空间占用差别比较大。

注意事项

- 如果回收站数据库（__recycle_bin__）和待回收的表跨了文件系统，执行 `DROP TABLE` 语句将会搬迁表空间文件，耗时较长。
- 如果表空间为General，可能会存在多个表共享同一个表空间的情况，当回收其中一张表的时候，不会搬迁相关的表空间文件。

参数

使用Recycle Bin功能前，您需要先配置如下参数。

> **说明** 关于修改集群参数的操作步骤，详情请参见设置集群参数。

参数	说明
recycle_bin	是否打开表回收站功能，默认取值为OFF。包括session级别和global级别。

参数	说明
recycle_bin_retention	回收站内数据的最长保留周期。取值范围为86400~1209600，单位为秒。默认取值为604800（即7天）。

管理语句

Recycle Bin提供了如下管理语句：

- 查看回收站中所有临时保存的表。

```
CALL DBMS_RECYCLE.show_tables()
```

示例

使用如下语句进行查看：

```
mysql> CALL DBMS_RECYCLE.show_tables();
```

返回结果如下：

```
+-----------------+---------------+---------------+--------------+---------------------+-
--------------------+
| SCHEMA          | TABLE         | ORIGIN_SCHEMA | ORIGIN_TABLE | RECYCLED_TIME       |
PURGE_TIME          |
+-----------------+---------------+---------------+--------------+---------------------+-
--------------------+
| __recycle_bin__ | __innodb_1063 | product_db    | t1           | 2019-08-08 11:01:46 |
2019-08-15 11:01:46 |
| __recycle_bin__ | __innodb_1064 | product_db    | t2           | 2019-08-08 11:01:46 |
2019-08-15 11:01:46 |
| __recycle_bin__ | __innodb_1065 | product_db    | parent       | 2019-08-08 11:01:46 |
2019-08-15 11:01:46 |
| __recycle_bin__ | __innodb_1066 | product_db    | child        | 2019-08-08 11:01:46 |
2019-08-15 11:01:46 |
+-----------------+---------------+---------------+--------------+---------------------+-
--------------------+
4 rows in set (0.00 sec)
```

参数	说明
SCHEMA	回收站的Schema。
TABLE	进入回收站后的表名。
ORIGIN_SCHEMA	原始表的Schema。
ORIGIN_TABLE	原始表的表名。
RECYCLED_TIME	回收时间。
PURGE_TIME	预计在回收站中被清理的时间。

- 手动清理回收站中的某张表。

```
CALL DBMS_RECYCLE.purge_table('TABLE_NAME')
```

> **说明**
> - `TABLE_NAME` 为进入回收站后的表名。
> - 执行清理操作的账号在原表和回收站表都需要具有DROP权限。

示例

```
mysql> CALL DBMS_RECYCLE.purge_table('__innodb_1063');
```

- 快速恢复回收站内的表。

```
CALL DBMS_RECYCLE.restore_table('RECYCLE_TABLE','DEST_DB','DEST_TABLE');
```

> **说明** PolarDB MySQL引擎8.0版本集群版的Revision version为8.0.1.1.12或以上才支持通过 `restore_table` 命令，快速恢复回收站内的表。您可以参见查询版本号确认集群版本。

参数说明如下：

参数	说明
`RECYCLE_TABLE`	需要恢复的表在回收站内的表名。 **说明** 如果仅传入此参数，会恢复到原始表。
DEST_DB	为恢复后的表指定目标数据库。
DEST_TABLE	为恢复后的表指定新的表名。

> **说明** 执行此命令需要有数据库 `__recycle_bin__` 的ALTER_ACL和DROP_ACL权限，以及目标表的CREATE_ACL和INSERT_ACL权限。

示例

```
mysql> call dbms_recycle.restore_table('__innodb_1063','testDB','testTable');
```

4.6.4. Returning

PolarDB提供Returning功能，支持在使用DML语句后返回结果集（Result set）报文。本文介绍如何使用PolarDB的Returning功能。

前提条件

PolarDB集群版本需为PolarDB MySQL引擎5.7版本且Revision version为5.7.1.0.6或以上，您可以参见查询版本号确认集群版本。

背景信息

MySQL的DML语句执行结果报文通常为OK或ERR，返回的结果报文中仅包括影响的记录数、扫描记录等信息，而不会返回结果集（Resultset）以显示具体的记录。但在很多业务场景中，执行完INSERT、UPDATE、DELETE等DML语句后，通常还需要再执行一次SELECT查询，来确认当前的记录内容，方便接下来的业务处理。

为减少客户端和服务器的交互次数并保证与MySQL语法的兼容性，PolarDB提供了Returning功能，支持在使用DML语句后返回结果集报文。

语法

```
CALL DBMS_TRANS.RETURNING(Field_list=>, Statement=>);
```

> **说明** `CALL DBMS_TRANS.RETURNING()` 不是事务性语句，会根据DML语句继承事务上下文，需要通过显式的提交或回滚来结束事务。

参数说明如下。

参数	说明
`Field_list`	期望返回的字段，多个字段间用英文逗号（,）分隔。字段需满足如下要求： • 仅支持表中的原生字段或用星号（*）表示所有字段。 • 不支持对字段进行计算或聚合等操作。
`Statement`	需要执行的DML语句，支持如下语句： • INSERT • UPDATE • DELETE

示例

本文以表 `t` （建表语句如下所示）为例，介绍如何使用Returning功能，返回DML语句的结果集（Resultset）报文。

```
CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `col1` int(11) NOT NULL DEFAULT '1',
  `col2` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

- INSERT

  > **说明** INSERT returning只支持 `insert values` 形式的语法，不支持 `create as` 、 `insert select` 等形式的语法（如 `CALL DBMS_TRANS.RETURNING("", "insert into t select * from t");` ），否则将报错（如 `ERROR 7527 (HY000): Statement didn't support RETURNING clause` ）。

  使用如下命令查看需要使用INSERT语句插入的记录：

```
CALL DBMS_TRANS.RETURNING("*", "insert into t(id) values(NULL),(NULL)");
```

返回结果如下：

```
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:39:05 |
|  2 |    1 | 2019-09-03 10:39:05 |
+----+------+---------------------+
2 rows in set (0.01 sec)
```

若 `Field_list` 留空，例如使用了 `call dbms_trans.returning("", "insert into t(id) values(NULL),(NULL)");` 命令，返回结果将只显示OK或ERR报文。示例如下：

```
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

此时如需查看表当前的记录内容，请使用 `select * from t;` 命令进行查看，返回结果如下：

```
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:40:55 |
|  2 |    1 | 2019-09-03 10:40:55 |
|  3 |    1 | 2019-09-03 10:41:06 |
|  4 |    1 | 2019-09-03 10:41:06 |
+----+------+---------------------+
4 rows in set (0.00 sec)
```

- UPDATE

> **说明** UPDATE Returning不支持涉及多表的UPDATE语句。

使用如下命令查看使用UPDATE语句更新后的记录：

```
CALL DBMS_TRANS.RETURNING("id, col1, col2", "update t set col1 = 2 where id >2");
```

返回结果如下：

```
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  3 |    2 | 2019-09-03 10:41:06 |
|  4 |    2 | 2019-09-03 10:41:06 |
+----+------+---------------------+
2 rows in set (0.01 sec)
```

- DELETE

使用如下命令查看需要使用DELETE语句删除的记录：

```
CALL DBMS_TRANS.RETURNING("id, col1, col2", "delete from t where id < 3");
```

返回结果如下：

```
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:40:55 |
|  2 |    1 | 2019-09-03 10:40:55 |
+----+------+---------------------+
2 rows in set (0.00 sec)
```

5.SDK下载

PolarDB支持Java、Node.js、Go、PHP、.NET和Python开发。

下表列举了各语言SDK的下载地址和开发指南，更多SDK的信息，请访问阿里云开放平台。

> **说明** 本文介绍的SDK仅用于调用PolarDB的OpenAPI。例如您可以通过SDK调用创建集群或修改集群参数等OpenAPI，但不可用于数据的增删改查等操作。更多关于PolarDB的OpenAPI信息，请参见API概览。

Alibaba Cloud SDK	PolarDB SDK	说明文档
Alibaba Cloud SDK for Java	Alibaba Cloud PolarDB SDK for Java	快速开始
Alibaba Cloud SDK for Node.js	Alibaba Cloud PolarDB SDK for Node.js	快速开始
Alibaba Cloud SDK for Go	Alibaba Cloud PolarDB SDK for Go	快速开始
Alibaba Cloud SDK for PHP	Alibaba Cloud PolarDB SDK for PHP	快速开始
Alibaba Cloud SDK for .NET	Alibaba Cloud PolarDB SDK for .NET	快速开始
Alibaba Cloud SDK for Python	Alibaba Cloud PolarDB SDK for Python	快速开始

6.常见问题

本文介绍PolarDB MySQL引擎的常见问题和解答。

基本问题

- Q：什么是PolarDB?

  A：PolarDB是一个关系型数据库云服务，目前已在全球十多个地域（Region）的数据中心部署，向用户提供开箱即用的在线数据库服务。PolarDB目前支持3种独立的引擎，分别可以100%兼容MySQL、100%兼容PostgreSQL、高度兼容Oracle语法，存储容量最高可达100 TB。详情请参见什么是PolarDB。

- Q：为什么云原生关系型数据库PolarDB优于传统数据库?

  A：相较于传统数据库，云原生关系型数据库PolarDB支持上百TB级别海量数据存储，提供高可用和高可靠保障、快速弹性升降级、无锁备份等功能，详情请参见产品优势。

- Q：PolarDB是什么时候发布？什么时候开始商用?

  A：2017年9月发布公测，2018年3月开始商用。

- Q：集群和节点分别指的是什么?

  A：PolarDB集群版采用多节点集群的架构，集群中有一个主节点和多个只读节点。单个PolarDB集群支持跨可用区，但不能跨地域，面向集群进行管理和计费。详情请参见术语。

- Q：支持哪些编程语言?

  A：PolarDB支持Java、Python、PHP、Golang、C、C++、.NET、Node.js等编程语言。只要支持原生MySQL的编程语言都可以直接使用PolarDB MySQL引擎，详情请参见MySQL官网。

- Q：支持哪些存储引擎?

  A：PolarDB支持3种概述，不同系列支持的存储引擎详情如下：

  - PolarDB MySQL引擎集群版和单节点全部表均使用InnoDB存储引擎。创建表的时候，PolarDB MySQL引擎会自动将非InnoDB引擎（如 MyISAM、Memory、CSV 等）转换为InnoDB引擎，因此即使迁移之前的数据表不是InnoDB，也仍然能够正常迁移至PolarDB MySQL引擎。
  - PolarDB MySQL引擎历史库默认使用X-Engine，可以提供强大的数据压缩能力，以满足归档数据库低存储成本的要求。更多详情，请参见历史库概述。

- Q：是否支持自建Slave实例，是否有推荐的实现方式?

  A：支持。启用Binlog后可以将PolarDB MySQL引擎同步到其他MySQL库，构成Master-Slave架构。为方便后续维护，建议您使用数据传输服务DTS（Data Transmission Service），关于如何使用DTS实现同步，请参见从PolarDB MySQL同步至RDS MySQL。

- Q：PolarDB是分布式数据库吗?

  A：是的，PolarDB是基于Parallel Raft一致性协议的分布式存储集群，计算引擎是由1~16个分布在不同服务器上的计算节点构成，存储容量最高可达100 TB，最高支持88核710 GB内存，可在线动态扩容存储和计算资源，扩容时不会影响业务的正常运行。

- Q：购买PolarDB后，如果需要分库分表是否还需要购买PolarDB-X数据库中间件?

  A：是的。

- Q：PolarDB是否支持表的分区?

  A：支持。

- Q：PolarDB是否已经自动包含了分区机制?

A：PolarDB在存储层做了分区，对用户透明，无感知。

- Q：对比原生MySQL，PolarDB单表最多支持存储多少数据量？

  A：PolarDB不限制单表大小，但单表大小受磁盘空间大小限制，详情请参见使用限制。

兼容性

- Q：是否兼容社区版MySQL?

  A：PolarDB MySQL引擎可以100%兼容社区版MySQL。

- Q：支持哪些事务隔离级别？

  A：PolarDB MySQL引擎支持READ_UNCOMMITTED、READ_COMMITTED（默认）、REPEATABLE_READ这三种隔离级别，不支持SERIALIZABLE隔离级别。

- Q：SHOW PROCESSLIST与社区版MySQL是否存在差异？

  A：如果是通过主地址查询，两者没有区别。但如果是通过集群地址查询，略有差异，此时会出现有多条相同Thread ID的记录，分别对应PolarDB MySQL引擎集群中的每一个节点。

- Q：锁机制和社区版MySQL是否存在差异？

  A：PolarDB MySQL引擎会将DDL中涉及到的Exclusive MDL锁同步到读节点上（同样通过Redo日志），并且使读节点持有MDL锁直到DDL操作结束，来阻止读节点上其它用户线程在DDL执行过程中访问表数据。与社区版MySQL不同，PolarDB MySQL引擎的主节点和读节点是基于共享存储的，这会导致主节点在做DDL的时候，读节点可能会查询到DDL过程中的中间数据而出现错误。

- Q：Binlog格式和MySQL原生格式是否存在差异？

  A：没有差异。

- Q：是否支持performance schema和sys schema?

  A：支持。

- Q：表统计信息收集和社区版MySQL是否存在差异？

  A：PolarDB MySQL引擎主节点的表统计信息和社区版MySQL一致。为了保证主节点和只读节点执行计划的一致性，主节点每次更新统计信息时，会同步到只读节点。此外，只读节点还可以通过ANALYZE TABLE操作，主动从磁盘加载最新的统计信息。

- Q：PolarDB是否支持XA事务，和官方MySQL是否存在差异？

  A： 支持，没有差异。

- Q：PolarDB是否支持全文索引？

  A：支持。

  > **说明** 目前，用户使用全文索引时，只读节点存在一定的索引缓存数据延迟，建议读写全文索引的操作都使用主地址，以读到最新的数据。

- Q：是否支持Percona工具集？

  A： 支持，但是建议您使用online DDL。

- Q：是否支持gh-ost？

  A：支持，但是建议您使用online DDL。

费用

- Q：PolarDB的费用都包含哪些？

  A：包含存储空间、计算节点、备份（附赠免费额度）、SQL洞察（可选），详情请参见计费项概览。

- Q：收费的存储空间都包含哪些内容？

  A：包含数据库表文件、索引文件、undo日志文件、Redo日志文件、binlog文件、slowlog文件及少量的系统文件，详情请参见存储空间计费规则。

- Q：PolarDB的存储包怎么用？

  A：购买的包年包月或按量付费的集群，均可使用存储包抵扣存储费用。例如您有3个存储容量均为40 GB的集群（即总容量为120 GB），这3个集群可以共享一个100 GB的存储包，多出的20 GB则按量计费，详情请参见购买存储包。

集群访问（读写分离）

- Q：如何实现PolarDB的读写分离？

  A：只需在应用程序中使用集群地址，即可根据配置的读写模式实现读写分离，详情请参见配置数据库代理。

- Q：一个PolarDB集群内最多可以支持多少个只读节点？

  A：PolarDB采用分布式集群架构，一个集群包含一个主节点和最多15个只读节点（至少一个，用于保障高可用）。

- Q：多个只读节点间负载不均衡的原因是什么？

  A：只读节点间负载不均衡的原因有只读节点连接数较少、自定义集群地址分配时未包括某个只读节点等。

- Q：造成主节点负载高或低的原因是什么？

  A：造成主节点（主库）负载高的原因有直连主地址、主库接受读请求、存在大量的事务请求、主从复制延迟高导致请求被路由到主库、只读节点异常导致读请求被路由到主库等。

  而主节点负载较低的原因可能是主库开启了不接受读选项。

- Q：怎么降低主节点的负载？

  A： 您可以使用如下几种方式降低主节点负载：

  - 使用集群地址来连接PolarDB集群，详情请参见配置数据库代理。
  - 如果由于事务较多导致主节点压力大，您可以通过控制台打开事务拆分功能，把事务中的部分查询路由到只读节点，详情请参见事务拆分。
  - 如果由于复制延迟导致请求被路由到主库，您可以考虑降低一致性等级（如使用最终一致性），详情请参见一致性级别。
  - 主库接受读请求，可能也会导致主库负载高，您可以通过控制台开启主库不接受读功能，减少读请求被路由到主库，详情请参见主库不接受读。

- Q：为什么读不到刚插入的数据？

  A：该问题可能是由于一致性级别的配置导致的，PolarDB的集群地址支持如下几种一致性级别：

  - 最终一致性：不论是同一会话（连接）或不同会话，最终一致性都不保证读能够马上读到刚插入的数据。
  - 会话一致性：一定能够读到同一会话插入之后的数据。
  - 全局一致性：保证同一会话和不同会话都能够读到最新数据。

> **说明** 一致性等级越高，性能越差，对主库的压力越大，请谨慎选择。对于大多数应用场景会话一致性能够保证业务正常工作，对于少数有强一致性的需求的语句，可以通过Hint `/* FORCE_MASTER */` 来实现，详情请参见一致性级别。

- Q：如何强制SQL到主节点执行？

  使用集群地址时，在SQL语句前加上 `/* FORCE_MASTER */` 或 `/* FORCE_SLAVE */` ，即可强制指定这条SQL的路由方向，详情请参见HINT语法。

  - `/* FORCE_MASTER */` 强制请求被路由到主库。该用法可以用于解决少数一致性要求较高的读请求的场景。
  - `/* FORCE_SLAVE */` 强制请求被路由到从库。该用法可以用于解决少数PolarDB代理由于保证正确性，要求特殊语法被路由到从库的场景（比如存储过程的调用，multistatement的使用等语句默认是会被路由到从库）。

  > **说明**
  > - 若您需要通过MySQL官方命令行执行上述Hint语句，请在命令行中加上-c参数，否则该Hint会被MySQL官方命令行过滤导致Hint失效，具体请参见MySQL官方命令行。
  > - Hint的路由优先级最高，不受一致性级别和事务拆分的约束，使用前请进行评估。
  > - Hint语句里不要有改变环境变量的语句，例如 `/*FORCE_SLAVE*/ set names utf8;` 等，这类语句可能导致查询结果非预期。

- Q：是否可以给不同的业务分配不同的地址？不同地址间是否可以达到隔离的效果？

  A：您可以创建多个自定义地址给不同的业务使用，若底层节点不同则自定义地址间可同时具备隔离的效果，不会互相影响。关于如何创建自定义地址，详情请参见新增自定义集群地址。

- Q：如果有多个只读节点，如何为其中某个只读节点单独创建单节点地址？

  A：仅当集群地址读写模式为只读且集群内拥有三个及以上节点时，才支持创建单节点地址，详细操作步骤请参见设置集群地址。

  > **警告** 创建单节点地址后，当此节点故障时，该地址可能会出现最多1小时不可用的情况，请勿用于生产环境。

- Q：一个集群内最多允许创建多少个单节点地址？

  A：如果您的集群内有3个节点，则只允许为其中1个只读节点创建单节点地址；若集群内有4个节点，则允许为其中2个只读节点创建各自的单节点地址，以此类推。

- Q：只用了主地址，但是发现只读节点也有负载，是否主地址也支持读写分离？

  A：主地址不支持读写分离，始终只连接到主节点。只读节点有少量QPS是正常现象，与主地址无关。

管理与维护

- Q：如何在线添加字段和索引？

  A：支持原生自带的online DDL、pt-osc和gh-ost等工具，建议您使用自带的online DDL操作。

> **说明** 使用pt-osc工具时，请不要使用用于设置主从检测的相关参数，如参数 `recursion-method` 。因为pt-osc工具是基于binlog复制进行主从检测的，但PolarDB内部采用的是物理复制，没有基于binlog的复制信息。

- Q：是否支持bulk insert功能？

  A：支持。

- Q：若只向只写节点写入数据，是否支持bulk insert？一次最多支持insert多少个values？

  A：支持。一次最多支持的values数量由max_allowed_packet参数值决定，详细请参见Replication and max_allowed_packet。

- Q：是否支持通过集群地址执行bulk insert操作？

  A：支持。

- Q：主节点（主）与只读节点（备）是否存在复制延迟？

  A：是，它们之间存在毫秒级延迟。

- Q：什么情况下会导致复制延迟增大？

  A：出现如下情况时会导致复制延迟增大：

  - 主节点写入负载高，产生了过多的Redo日志，导致只读节点来不及应用。
  - 只读节点负载过高，抢占了过多原本属于应用Redo日志的资源。
  - I/O出现瓶颈，导致读写Redo日志过慢。

- Q：存在复制延迟的情况下，如何保证查询的一致性？

  A：您可以使用集群地址并为其选择合适的一致性级别。目前一致性从高到低分别为全局一致性（强一致性）、会话一致性和最终一致性，详情请参见一致性级别。

- Q：单节点故障的情况下是否可以保证RPO为0？

  A：可以。

- Q：升级规格配置（比如从2核8 GB升级到4核16 GB）后端是怎么实现的？对业务有什么影响？

  A：PolarDB的代理（Proxy）和数据库节点（Node）均需要升级到最新的配置，采用多个节点滚动升级的方式尽量减少对业务的影响。目前每次升级大概需要10~15分钟，对业务的影响时间不超过30秒，期间可能会产生1~3次连接闪断，详情请参见手动变配。

- Q：添加节点要多久？是否会影响业务？

  A：每增加一个节点需要5分钟，对业务无影响。关于如何添加节点，详情请参见增加只读节点。

  > **说明** 新增只读节点之后新建的读写分离连接会转发请求到该只读节点。新增只读节点之前建立的读写分离连接不会转发请求到新增的只读节点，需要断开该连接并重新建立连接，例如，重启应用。

- Q：升级到最新修订版本需要多久？是否会影响业务？

  A：PolarDB采用多节点滚动升级的方式尽量减少对业务的影响。版本升级一般不超过30分钟，升级过程中会重启数据库代理Proxy或内核引擎DB，可能会导致数据库连接闪断。请您尽量在业务低峰期执行升级操作，并且确保您的应用有自动重连机制。详情请参见版本管理。

- Q：如何进行故障自动切换？

A：PolarDB采用双活（Active-Active）的高可用集群架构，可读写的主节点和只读节点之间自动进行故障切换（Failover），系统自动选举新的主节点。PolarDB每个节点都有一个故障切换（Failover）优先级，决定了故障切换时被选举为主节点的概率高低。当多个节点的优先级相同时，则有相同的概率被选举为主节点，详情请参见自动/手动主备切换。

备份与恢复

- Q：PolarDB采用什么备份方式？

  A：PolarDB采用快照（Snapshot）的方式进行备份，详情请参见备份方式1：自动备份和备份方式2：手动备份。

- Q：数据库恢复的速度如何？

  A：目前，基于备份集（快照）进行恢复（克隆）的速度是40分钟/TB。如果是恢复到任意时间点，则需要包含应用Redo日志的时间，这部分的恢复速度大概是20~70秒/GB，整个恢复时间是这两部分之和。

性能和容量

- Q：为什么PolarDB MySQL引擎相比RDS MySQL性能提升不明显？

  A：在您对PolarDB MySQL引擎和RDS MySQL进行性能对比前，请了解以下注意事项，以便能获得比较准确、合理的性能对比结果。

  - 使用相同规格配置的PolarDB MySQL引擎和RDS MySQL进行性能对比。
  - 使用相同版本的PolarDB MySQL引擎和RDS MySQL进行性能对比。

    因为不同版本的实现机制不一样，例如MySQL 8.0针对多核数CPU做优化，单独抽象出来Log_writer、log_fluser、log_checkpoint、log_write_notifier等线程，但在CPU核数较少的情况下性能则不如MySQL 5.6或5.7。不推荐使用PolarDB MySQL引擎5.6和RDS MySQL 5.7或8.0进行对比，因为MySQL 5.6的优化器比较旧，不如新版本。

  - 推荐使用模拟线上压力的场景进行实际性能对比，或者使用sysbench进行对比，这样获得的数据更接近线上实际场景。
  - 在对比读性能的时候，不推荐您使用单条SQL进行比较。

    因为PolarDB是计算存储分离的架构，所以单条语句有网络延迟的影响，导致读性能不如RDS。线上数据库的缓存命中率基本都在99%以上，只有第一次的读会调用I/O，因此读取性能会降低；后续数据都在缓存池（Buffer Pool）中，并不需要调用I/O，因此性能是一样的。

  - 在对比写性能的时候，同样不推荐您使用单条SQL进行比较，推荐模拟线上环境进行压力测试。

    如果要对比RDS性能，请使用PolarDB（主节点+只读节点）和RDS（主实例+半同步的只读实例）进行对比。这是因为PolarDB的架构在写入数据的时候默认采用Quorum机制，即写入数据时默认写入到三副本里面的大多数（在三个副本中的两个或两个以上写入成功，就认为写操作成功了）。PolarDB已经在存储层面做数据冗余，并保证三副本强同步高可靠，使用RDS MySQL的半同步复制（而不是异步复制）进行对比更合理。

  PolarDB MySQL引擎与RDS MySQL的性能对比结果，请参见与RDS MySQL的对比。

- Q：删除数据库后为什么还是占用很多空间？

  A：这是由于Redo日志文件占用了空间，通常在2 GB~11 GB左右，最多时会占用11 GB，其中包括缓冲池中8个Redo日志（8 GB）、正在写的Redo日志（1 GB）、提前创建的Redo日志（1 GB）以及最后一个Redo日志（1 GB）。

  缓冲池内的Redo日志文件数量由参数 `loose_innodb_polar_log_file_max_reuse` 控制，默认值为8。您可以修改这个参数从而减少日志空间占用量，但在压力大的情况下，性能可能会出现周期性的小幅波动。

- Q：表个数上限是多少？表个数到多少时有可能会引起性能下降？

  A：表个数的上限受文件数量限制，详情请参见使用限制。

- Q：表分区能够提高PolarDB的查询性能吗？

  A：通常来说，如果查询SQL能够落在某个分区内，是可以提升性能的。

- Q：PolarDB是否支持创建1万个数据库？数据库个数上限是多少？

  A：PolarDB支持创建1万个数据库。数据库个数上限受文件数量限制，详情请参见使用限制。

- Q：只读节点的数量与最大连接数有关系吗？可以通过增加只读节点来增加最大连接数吗？

  A：只读节点的数量与最大连接数无关，PolarDB的最大连接数由节点规格决定，详情请参见使用限制。若需更大的连接数，请升级规格。

- Q：IOPS是怎么限制和隔离的？是否会出现多个PolarDB集群节点的I/O争抢？

  A：PolarDB集群的每个节点根据规格大小设置IOPS，每个节点之间IOPS独立隔离，互不影响。

- Q：只读节点的性能变慢是否会影响主节点？

  A：只读节点的负载过高、复制延迟增高时，可能会少量增加主节点的内存消耗。

- Q：打开Binlog之后，对性能有什么影响？

  A：开启Binlog不会影响查询（SELECT）性能，只会影响写入更新（INSERT、UPDATE、DELETE）性能。一般情况下，在读写均衡的数据库中，开启Binlog后对性能影响不超过10%。

- Q：打开SQL洞察（全量SQL日志审计），对性能有什么影响？

  A：无影响。

- Q：PolarDB使用了什么高速网络协议？

  A：PolarDB的数据库计算节点和存储节点之间，以及存储数据多副本之间，都使用了双25 Gbps RDMA技术，提供低延迟、高吞吐的强劲I/O性能。

- Q：PolarDB外网连接的带宽上限是多少？

  A：PolarDB外网连接的带宽上限为10 Gbit/s。

- Q：重启节点需要的时间很长怎么办？

  A：当您的集群中存在的文件数量越多，节点重启需要的时间会越长。此时您可以通过修改innodb_fast_startup参数值为*ON*来加速重启，关于如何修改参数，请参见设置集群参数。

  > **说明** 仅PolarDB MySQL引擎8.0和5.6版本支持配置该参数。

大表问题

- Q：对比传统本地盘的数据库，PolarDB MySQL引擎中的大表存储有什么优势？

A：PolarDB MySQL引擎中的一张表，物理上会被拆分到N台存储服务器上存储，因此对一张表的I/O会被分摊到多块存储磁盘中，I/O读取的整体吞吐性能（而不是I/O延迟）要远优于集中式的本地盘数据库。

- Q：大表如何优化？

  A：推荐使用分区表。

- Q：哪种情况下适合使用分区表？

  A：需要通过裁剪大表来控制查询访问的数据量并且希望该裁剪对业务代码透明（无需修改业务代码）的场景下，适合使用分区表。例如您可以通过使用分区表来定期清理业务历史数据（如删除最早的一个月份分区并新建下个月份分区，实现只保留最近6个月份数据的效果）。

- Q：在同一个PolarDB MySQL引擎的数据库里复制一个数据量很大的表（如将整张表A复制到表B中），什么方式比较合适？

  A：您可以使用如下SQL语句直接复制：

  ```
  create table B as select * from A
  ```

稳定性

- Q：是否可以对高并发下的PHP短连接进行优化？

  A：可以。在集群地址中，可以通过开启会话级连接池进行优化，详情请参见设置集群地址。

- Q：如何规避个别执行效率低下的SQL拖垮整个数据库？

  A：如果您的PolarDB MySQL引擎集群是5.6或8.0版本，您可以使用语句并发控制Statement Concurrency Control特性来实现针对指定语句的限流。

- Q：PolarDB是否支持空闲会话超时？

  A：支持。您可以通过修改wait_timeout参数来自定义空闲会话的超时时间，具体操作步骤请参见设置集群参数。

- Q：如何发现慢SQL？

  A：您可以通过如下两种方式发现慢SQL：

  - 直接在控制台上查询慢SQL，详情请参见慢SQL。
  - 连接数据库集群后执行 `show processlist;` ，找出执行时间过长的SQL，关于如何连接数据库集群，请参见连接数据库集群。

    ```
    mysql> show processlist;
    +----------+------+------------------------------------+------+---------+------+------------+------------------+
    | Id       | User | Host                               | db   | Command | Time | State      | Info             |
    +----------+------+------------------------------------+------+---------+------+------------+------------------+
    | 33554490 | acc  | [redacted]:19358                   | NULL | Query   |    0 | starting   | show processlist |
    | 33554499 | acc  | [redacted]                         | NULL | Query   |  250 | User sleep | select sleep(600) |
    | 33554499 | acc  | [redacted]                         | NULL | Sleep   |  253 |            | NULL             |
    +----------+------+------------------------------------+------+---------+------+------------+------------------+
    3 rows in set, 13312 warnings (0.00 sec)
    ```

- Q：如何终止慢SQL？

  A：发现慢SQL后，您可以先查看慢SQL的ID，然后执行 `kill <Id>` 终止慢SQL。

  ```
  mysql> kill 33554499;
  Query OK, 0 rows affected (0.01 sec)
  ```

7.视频专区

7.1. 全球数据库网络GDN

更多关于GDN的文字说明，请参见如下文档：

- 创建与删除全球数据库网络
- 添加与移除从集群

7.2. 历史库

更多关于历史库的文字说明，请参见如下文档：

- 历史库
- 使用说明

7.3. Instant DDL和Parallel DDL

更多关于Instant DDL和Parallel DDL的文字说明，请参见如下文档：

- Instant DDL
- Parallel DDL

7.4. 快速恢复

更多关于快速恢复的文字说明，请参见全量恢复方式1：从备份集恢复。

7.5. 并行查询和Hash Join的并行执行

更多关于并行查询和Hash Join的并行执行的文字说明，请参见并行查询（Parallel Query）和Hash Join的并行执行。

8.服务等级协议

请参见云原生关系型数据库PolarDB MySQL引擎服务等级协议。