ALIBABA CLOUD

阿里云

云原生数据仓库AnalyticDB MySQL版 开发指南

文档版本：20220711

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.欢迎

1.1. 数据库开发人员必读

如果您是数据库用户、数据库设计者、数据库开发人员或数据库管理员，下表将帮助您快速找到所需的帮助文档。

如果要...	我们建议
了解什么是ADB	云原生数据仓库AnalyticDB MySQL版介绍云原生数据仓库AnalyticDB MySQL版（简称ADB，原ADB）的产生背景和具备的产品优势。 如果您想要更全面地了解ADB，请参见产品详情页。
快速开始使用ADB	根据业务需要，按照ADB 2.0快速入门或者ADB 3.0快速入门中的步骤完成创建数据库、创建账号、设置白名单、连接数据库、加载数据并尝试进行一些查询。
ADB 3.0产品增强明细	功能特性帮助您快速了解ADB 3.0具备哪些功能。
创建数据库、表组、表、用户等数据库对象	ADB 2.0： • SQL包含ADB支持的SQL命令和相关SQL示例。 • 账号和权限介绍ADB中的账号类型、权限管理方式。 • 函数介绍ADB支持的函数和相关示例。 ADB 3.0： • SQL包含ADB支持的SQL命令和相关SQL示例。 • 账号和权限介绍ADB中的账号类型、权限管理方式以及基于RAM的子账号体系。 • 系统函数介绍ADB支持的函数和相关示例。
了解如何设计表和表设计的最佳实践	表结构设计及最佳实践详细介绍如何在ADB2.0中选择列和分区列及其注意事项。
数据接入	ADB 2.0：数据迁移介绍将MySQL、OSS、日志数据或者本地Excel等数据加载到ADB2.0的方法。
数据库安全	ADB 2.0：网络、用户管理和ACL权限体系涵盖ADB安全主题。 ADB 3.0：白名单、申请和释放公网地址、细粒度的权限控制保证数据的安全。
数据库高级功能	介绍ADB 2.0中二级分区、聚集列、分页等高级功能。
向量分析	向量介绍ADB 2.0中向量的适用场景、应用案例以及详细的使用手册。

1.2. 先决条件

使用本指南前，您应该完成以下任务。

- 安装MySQL客户端或者其他客户端（例如Navicat）。
- 购买AnalyticDB MySQL。
- 客户端工具连接AnalyticDB MySQL。

如需分步指导，请参见AnalyticDB MySQL使用流程。

您还应该知道如何使用客户端，并且对SQL语言有基本的了解。

2.数据导入导出

2.1. 支持的数据源

AnalyticDB MySQL版提供多种数据导入方案，可满足不同场景下的数据导入需求。

数据源		导入数据方式
数据库	RDS MySQL	• DTS：从RDS MySQL同步到云原生数据仓库AnalyticDB MySQL。 • INSERT外表：通过外表将RDS MySQL数据导入至AnalyticDB MySQL。 • DataWorks：配置RDS MySQL数据源。
	PolarDB-X（原DRDS）	• DTS：使用DTS同步PolarDB-X（原DRDS）数据。 • DataWorks：配置PolarDB-X（原DRDS）读取。
	PolarDB MySQL	DTS：使用DTS同步PolarDB MySQL数据。
	Oracle	DataWorks：配置Oracle数据源。
	SQL Server	DataWorks：配置SQL Server数据源。
OSS		• INSERT外表：通过外表将OSS数据导入至AnalyticDB MySQL。 • DataWorks：配置OSS数据源。
大数据	MaxCompute	• INSERT外表：通过外表将MaxCompute数据导入至AnalyticDB MySQL。 • DataWorks：配置MaxCompute数据源。
	Hadoop、EMR	DataWorks：配置HDFS数据源。
	Flink	通过Flink导入数据：通过开源Flink导入数据至AnalyticDB MySQL。
消息队列	Kafka	• 使用Logstash插件：使用Logstash将Kafka数据写入AnalyticDB MySQL。 • DataWorks：配置Kafka读取。
日志类数据	日志服务（SLS）	• SLS：将SLS数据投递到AnalyticDB MySQL。 • DataWorks：配置SLS读取。
	日志数据	使用Logstash插件：使用Logstash实时采集日志数据。
本地数据		• `LOAD DATA`：使用LOAD DATA导入本地数据。 • DataWorks：先将数据导入OSS或者FTP，再使用OSS读取或者FTP读取的方式导入，详情请参见OSS读取和FTP读取。 • 导入工具：使用AnalyticDB MySQL版导入工具导入本地数据。

另外，AnalyticDB MySQL版也支持通过异步方式提交数据导入导出任务，详情请参见异步提交导入导出任务。

2.2. 数据导入方式介绍

为满足多样化的数据导入需求，云原生数据仓库AnalyticDB MySQL版提供了多种数据导入方式，包括：通过外表导入数据、使用DataWorks导入数据和利用JDBC通过程序导入数据等。本文介绍各导入方式的特性及适用场景，帮助您选择正确的数据导入方式。

通过外表导入数据

AnalyticDB MySQL内置不同数据源的访问链路，支持通过创建外表来映射外部数据源，并发地读取外部数据并导入到AnalyticDB MySQL。通过外表导入数据会最大限度地利用集群资源，实现高性能数据导入。

基本特性

- 适合大批量数据：导入链路批量操作，适合单任务进行大量数据导入的场景。
- 资源消耗大：利用集群资源进行高性能导入，建议在业务低峰期使用。
- 批量可见：数据导入任务完成前数据不可见，任务完成后导入的数据批量可见。
- 分区覆盖：通过外表导入的数据分区会覆盖表中已存在的同一分区。
- 构建索引：通过外表导入会同步构建索引，导入任务完成则生成索引，可提升查询性能。

常见使用场景

- **数仓初始化**

  当存在TB级数据需要初始化导入到AnalyticDB MySQL进行分析，建议先将数据存放在OSS或者HDFS，再通过外表高效导入。

- **离线数仓加速**

  离线数据运行在MaxCompute等离线数仓上，单天数据增量达到几十GB甚至TB级，需要每天导入数据到AnalyticDB MySQL进行数据加速分析。

支持的外表数据源与使用方法

AnalyticDB MySQL支持多种外表数据源，包括MaxCompute、HDFS、OSS、RDS MySQL。使用方法请参见下面链接。

- 通过外表将MaxCompute数据导入至AnalyticDB MySQL
- 通过外表将HDFS数据导入至AnalyticDB MySQL
- 通过外表将OSS数据导入至AnalyticDB MySQL
- 通过外表将RDS MySQL数据导入至AnalyticDB MySQL

导入性能调优

如何提升外表数据的导入性能，请参见通用外表导入数据调优。

通过DataWorks导入数据

DataWorks提供了可视化的数据导入方式，可以将多种数据源导入到AnalyticDB MySQL。相对于通过外表导入数据的方法，DataWorks导入数据更为轻量化，适合数据量相对较小的数据导入场景。

> 说明 不建议通过DataWorks导入大量数据。如果存在数百GB以上的数据导入，建议通过外表导入数据。详情请参见通过外表导入数据。

常见使用场景

- **分钟/小时级数据导入**

  需要每分钟或每小时抽取少量数据到AnalyticDB MySQL进行数据分析。

- **多种异构数据源导入**

  需要导入OTS、Redis、PostgreSQL等多种数据源的数据到AnalyticDB MySQL。

使用方法

通过DataWorks导入数据分为3个步骤。

1. 配置源端数据源。支持的数据源包括：RDS MySQL、Oracle、SQL Server、OSS、MaxCompute及HDFS。
   - 配置RDS MySQL数据源
   - 配置Oracle数据源
   - 配置SQL Server数据源
   - 配置OSS数据源
   - 配置MaxCompute数据源
   - 配置HDFS数据源
2. 配置AnalyticDB MySQL 3.0数据源。
3. 配置同步任务中的数据来源和去向。

导入性能调优

如何提升DataWorks导入数据的性能，请参见通过DataWorks导入数据调优。

通过JDBC使用程序导入数据

在数据清洗或复杂非结构化数据场景下，当外表和DataWorks导入无法满足定制化导入需求时，可以编写程序通过JDBC导入数据。

常见使用场景

- **数据预处理后导入**

  业务端实时产生日志文件，需要对日志文件进行自动化解析并实时导入AnalyticDB MySQL。

- **非云上数据导入**

  当数据无法上传到OSS、HDFS或者MaxCompute时，需要将本地数据导入AnalyticDB MySQL。

使用方法与建议

- 应用程序连接AnalyticDB MySQL，需要配置支持的JDBC驱动，详情请参见MySQL JDBC驱动版本。
- 导入数据量大，且需长时间操作时，建议配置连接池，详情请参见Druid连接池配置。
- 应用导入支持批量导入和并发导入，以获得更高的导入性能。
- 关于流式数据导入，请参见通过开源Flink导入数据至AnalyticDB MySQL。
- 关于非定制化本地数据导入，请参见使用LOAD DATA导入本地数据、使用AnalyticDB MySQL版导入工具导入本地数据。

导入性能调优

如何提升使用应用程序导入数据的性能，请参见通过JDBC使用程序导入数据调优。

2.3. 数据导入性能优化

云原生数据仓库AnalyticDB MySQL版提供的多种数据导入方法，满足不同场景下的数据导入需求。然而数据导入性能依然受各种各样的因素影响，如表的建模不合理导致长尾、导入配置低无法有效利用资源等。本文介绍不同场景下的数据导入调优方法。

通用外表导入数据调优

检查分布键

分布键决定着数据导入的一级分区，每个表在导入时以一级分区为粒度并发导入。当数据分布不均匀时，导入数据较多的一级分区将成为长尾节点，影响整个导入任务的性能，因此要求导入时数据均匀分布。如何选择分布键，请参见选择分布键。

判断分布键合理性：

- 导入前，根据导入数据所选分布键的业务意义判断是否合理。以表Lineitem为例，当选择l_discount列为分布键，订单折扣值区分度很低，仅有11个不同值，l_discount值相同的数据会分布到同一分区，造成严重倾斜，导入会有长尾，影响性能。选择l_orderkey列则更为合适，订单ID互不相同，数据分布相对较为均匀。
- 导入后，数据建模诊断中如有分布字段倾斜，则说明选择的分布键不均匀。如何查看分布键诊断信息，请参见分布字段合理性诊断。

检查分区键

`INSERT OVERWRITE INTO SELECT` 导入数据的基本特性为分区覆盖，即导入的二级分区会覆盖原表的同名二级分区。每个一级分区内的数据会再按二级分区定义导入各个二级分区。导入时需要避免一次性导入过多二级分区，多个二级分区同时导入可能引入外排序过程，影响导入性能。如何选择分区键，请参见选择分区键。

判断分区键合理性：

- 导入前，根据业务数据需求及数据分布判断分区键是否合理。如Lineitem表按l_shipdate列做二级分区，数据范围横跨7年，按年做分区有7个分区，按日做分区有2000多个分区，单分区约3000万条记录，选择按月或者按年做分区则更合适。
- 导入后，数据建模诊断中如有不合理的二级分区，则选择的分区键不合适。如何查看分区键诊断信息，请参见分区字段合理性诊断。

检查索引

AnalyticDB MySQL建表时默认全列索引，而构建宽表的全列索引会消耗部分资源。导入数据到宽表时，建议使用主键索引。主键索引用于去重，主键列数过多影响去重性能。如何选择主键索引，请参见选择主键。

判断索引合理性：

- 离线导入场景通常已经通过离线计算进行去重，无需指定主键索引。
- 在**监控信息 > 表信息统计**页签，查看表数据量、索引数据量和主键索引数据量。当索引数据量超过表数据量时，需要检查表中是否有较长的字符串列，这种索引列不仅构建耗时，还占用存储空间，可以删除索引，请参见删除索引。

> **说明** 主键索引无法删除。需要重建表。

增加Hint加速导入

在导入任务前增加Hint（`direct_batch_load=true`）可以加速导入任务。

> **说明** 该Hint仅数仓版（3.0）弹性模式集群3.1.5版本支持，若使用后导入性能无明显提升，请提交工单。

示例如下：

```
submit job /* direct_batch_load=true*/insert overwrite adb_table
select * from adb_external_table;
```

通过DataWorks导入数据调优

优化任务配置

- 优化**批量插入条数**

  表示单次导入的批大小，默认为2048，一般不建议修改。

  如果单条数据量过大达到数百KB，如高达512 KB，则建议修改此配置为16，保证单次导入量不超过8 MB，防止占用过多前端节点内存。

- 优化**通道控制**
  - 数据同步性能与**任务期望最大并发数**配置项大小成正比，建议尽可能增加**任务期望最大并发数**。

    > **注意** 任务期望最大并发数越高，占用DataWorks资源会越多，请合理选择。

- 建议打开**分布式处理能力**，以取得更好的同步性能。

常见问题及解决方法

- 当客户端导入压力不足时，会导致集群CPU使用率、磁盘IO使用率及写入响应时间处于较低水位。数据库服务器端虽然能够及时消费客户端发送的数据，但由于总发送量较小，导致写入TPS不满足预期。

  解决方法：调大单次导入的**批量插入条数**及增加**任务期望最大并发数**，数据导入性能会随着导入压力的增加而线性增加。

- 当导入的目标表存在数据倾斜时，集群部分节点负载过高，影响导入性能。此时，集群CPU使用率、磁盘IO使用率处于较低水位，但写入响应时间较高，同时您可以在**诊断优化 > 数据建模诊断**页面的倾斜诊断表中发现目标表。

  解决方法：重新设计表结构后再导入数据，详情请参见表结构设计。

通过JDBC使用程序导入数据调优

客户端优化

- 应用端攒批，多条批量导入
  - 在通过JDBC使用程序导入数据过程中，为减少网络和链路上的开销，建议攒批导入。无特殊要求，请避免单条导入。
  - 批量导入条数建议为2048条。如果单条数据量过大达到数百KB，建议攒批数据大小不超过8 MB，可通过8 MB/单条数据量得到攒批条数。否则单批过大容易占用过多前端节点内存，影响导入性能。
- 应用端并发配置
  - 应用端导入数据时，建议多个并发同时导入数据。单进程无法完全利用系统资源，且一般客户端需要处理数据、攒批等操作，难以跟上数据库的导入速度，通过多并发导入可以加快导入速度。
  - 导入并发受攒批、数据源、客户端机器负载等影响，没有最合适的数值，建议通过测试逐步计算合适的并发能力。如导入不达预期，请翻倍加大并发，导入速度下降再逐步降低并发，寻找最合适的并发数。

常见问题及解决方法

当通过程序导入数据到AnalyticDB MySQL性能不佳时，首先排查客户端性能是否存在瓶颈。

- 保证数据源的数据生产速度足够大，如果数据源来自其他系统或文件，排查客户端是否有输出瓶颈。
- 保证数据处理速度，排查数据生产消费是否同步，保证有足够的数据等待导入AnalyticDB MySQL。
- 保证客户端机器负载，检查CPU使用率或磁盘IO使用率等系统资源是否充足。

2.4. 数据库

2.4.1. 使用DTS导入数据

2.4.1.1. 从RDS MySQL同步到云原生数据仓库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，使得您可以在毫秒级针对千亿级数据进行即时的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RDS 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

- RDS MySQL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 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。
- 如果同步的目标为云原生数据仓库AnalyticDB MySQL（2.0），那么源RDS MySQL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。

费用说明

同步类型	链路配置费用
库表结构同步和全量数据同步	不收费。
增量数据同步	收费，详情请参见产品定价。

术语/概念对应关系

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，请参见基本概念。

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 MySQL 2.0	• DDL操作：ADD COLUMN • DML操作：INSERT、UPDATE、DELETE
云原生数据仓库AnalyticDB MySQL 3.0	• DDL操作：CREATE TABLE、DROP TABLE、RENAME TABLE、TRUNCATE TABLE、ADD COLUMN、DROP 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参照文末的方法来手动修复，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
RDS MySQL	REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和所有同步对象的SELECT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。

数据类型映射关系

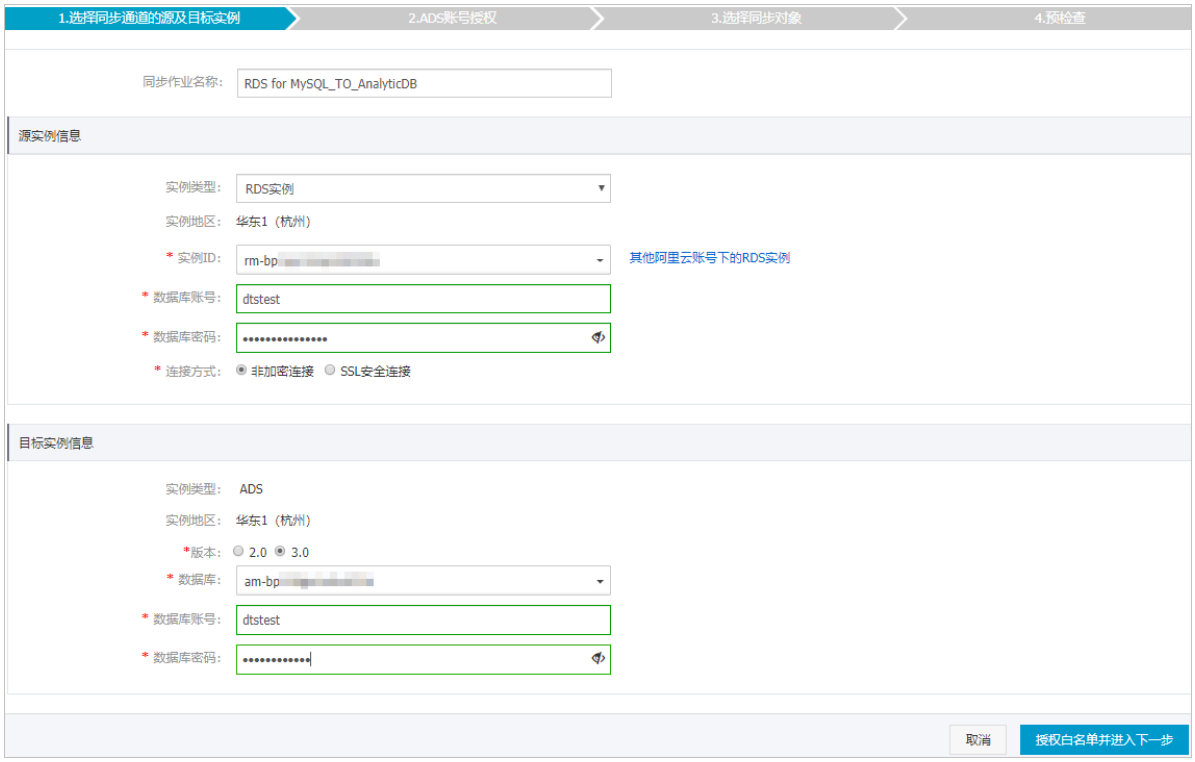
由于MySQL和云原生数据仓库AnalyticDB MySQL的数据类型并不是一一对应的，所以DTS在进行结构初始化时，会根据数据类型定义进行类型映射，详情请参见结构初始化涉及的数据类型映射关系。

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**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，选择数据同步实例所属地域。
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。

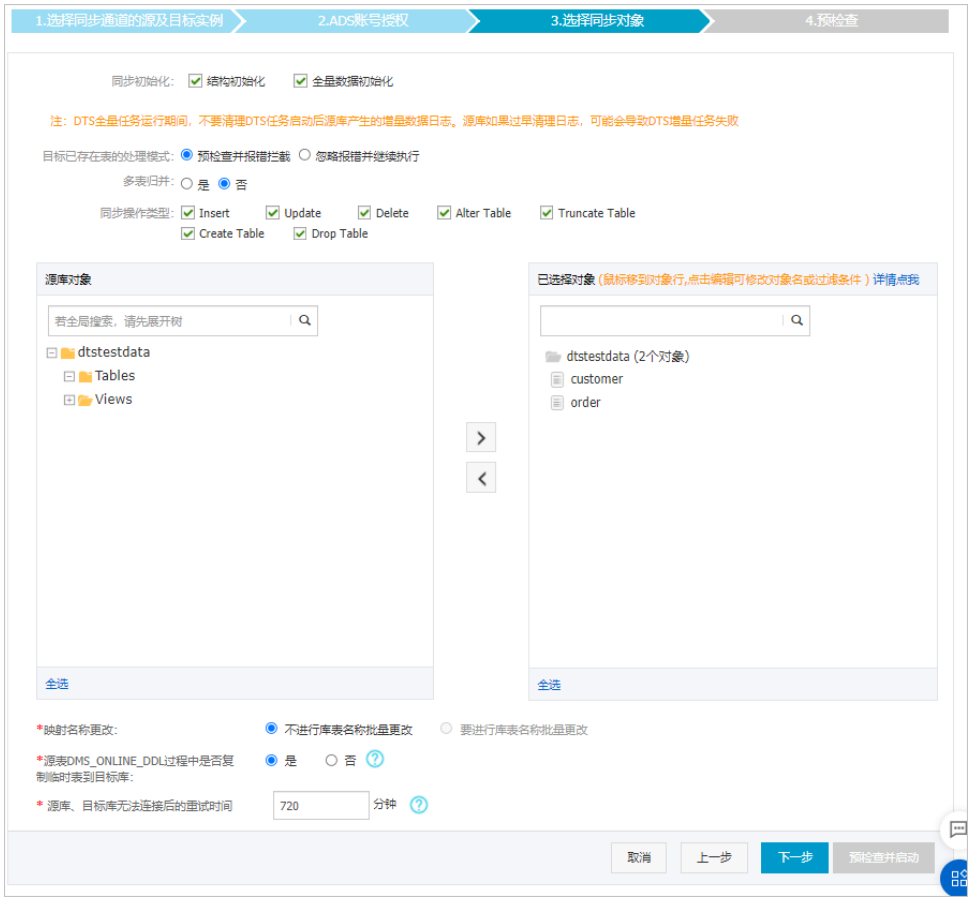


类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	选择**RDS实例**。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	实例ID	选择源RDS实例ID。
	数据库账号	填入源RDS的数据库账号，权限要求请参见数据库账号的权限要求。 说明 当源RDS实例的数据库类型为**MySQL 5.5**或**MySQL 5.6**时，无需配置**数据库账号**和**数据库密码**。
	数据库密码	填入该数据库账号对应的密码。
	连接方式	根据需求选择**非加密连接**或**SSL安全连接**。如果设置为**SSL安全连接**，您需要提前开启RDS实例的SSL加密功能，详情请参见设置SSL加密。
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，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**，权限要求请参见数据库账号的权限要求。
	数据库	选择目标云原生数据仓库AnalyticDB MySQL的集群ID。
	数据库账号	填入云原生数据仓库AnalyticDB MySQL的数据库账号。
	数据库密码	填入该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
> - 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。
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。

修复因变更字段类型导致的同步失败

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。您可参照如下的方法来手动修复，或者提交工单处理。

1. 在目标实例中，根据同步失败的表A（表名以customer为例），重新创建一个新表B（表名以customer_new为例），确保两张表的表结构保持一致。
2. 通过INSERT INTO SELECT命令，将表A的数据复制并插入到新创建的表B中，确保两张表的数据保持一致。
3. 重命名或删除同步失败的表A，然后将表B的名称修改为customer。
4. 在DTS控制台，重新启动数据同步作业。

2.4.1.2. 从PolarDB-X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-X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

- PolarDB-X中的数据库须基于RDS MySQL创建。
- 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。
- 如果同步的目标为云原生数据仓库AnalyticDB MySQL（2.0），那么源PolarDB-X中待同步的库或列的名称不能包含云原生数据仓库AnalyticDB MySQL（2.0）的保留字，否则将造成数据同步失败，详情请参见保留字。

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）。
- 数据同步期间，请勿对PolarDB-X执行扩容、缩容、迁移热点表、变更拆分键和变更DDL等操作，否则将导致数据同步失败。
- 如果需要在数据同步期间切换PolarDB-X的网络类型，在您执行完网络类型切换操作后，请提交工单调整同步链路的网络连接信息。
- 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。

费用说明

同步类型	链路配置费用
库表结构同步和全量数据同步	不收费。
增量数据同步	收费，详情请参见产品定价。

术语/概念对应关系

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，请参见基本概念。

支持同步的SQL操作

INSERT、UPDATE、DELETE。

数据库账号的权限要求

数据库	所需权限
PolarDB-X	Replication slave、Replication client及待同步对象的Select权限，由DTS自动执行授权。
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

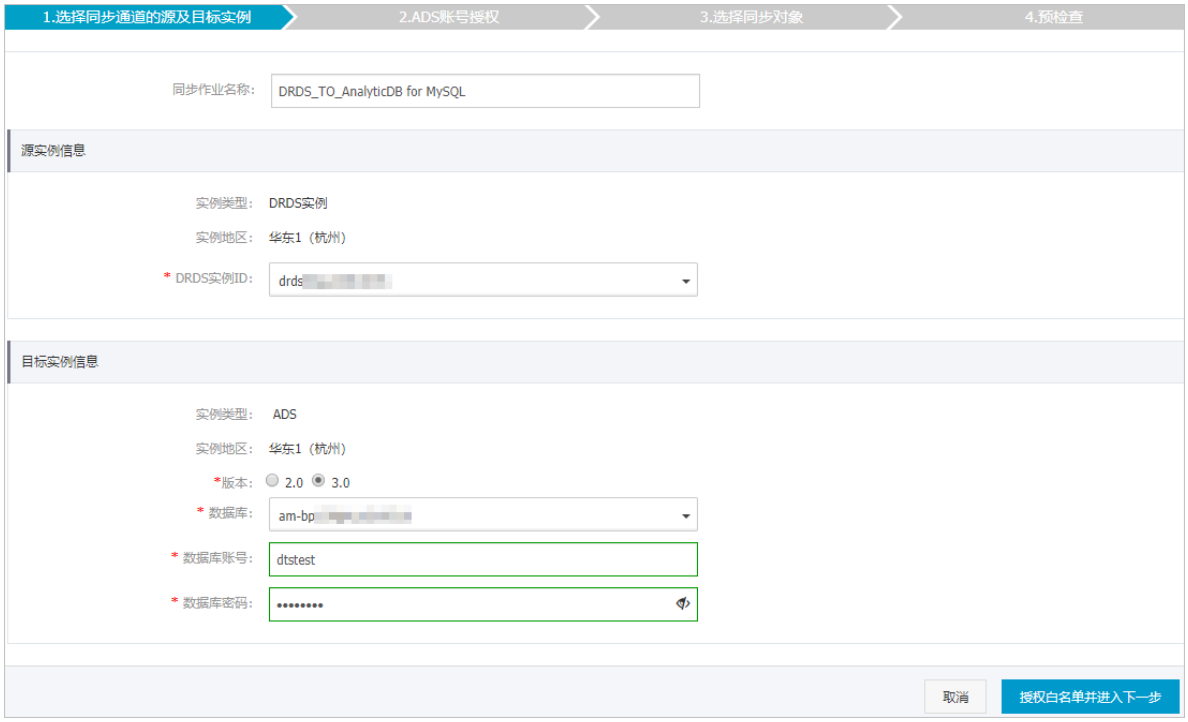
详情请参见结构初始化涉及的数据类型映射关系。

操作步骤

1. 购买数据同步作业，详情请参见购买流程。

   > 说明 购买时，选择源实例为PolarDB-X（原DRDS升级版）、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。

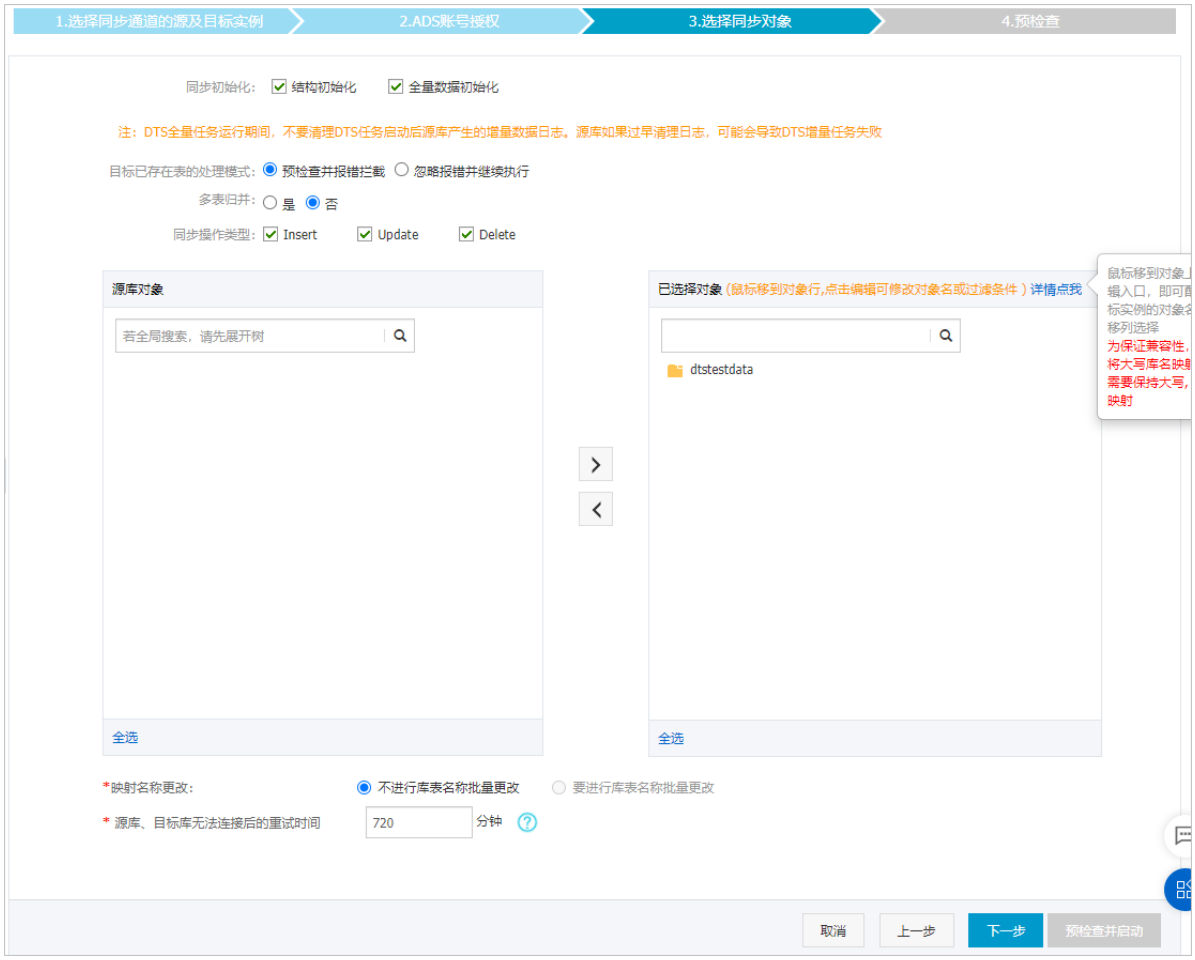


类别	配置	说明
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。
源实例信息	实例类型	固定为**DRDS实例**，不可变更。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	DRDS实例ID	选择源PolarDB-X的实例ID。
目标实例信息	实例类型	固定为**ADS**，不可变更。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	版本	根据目标云原生数据仓库AnalyticDB MySQL集群的版本，选择**2.0**或**3.0**。 说明 ○ 选择为**2.0**后，DTS将自动创建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	选择云原生数据仓库AnalyticDB MySQL实例ID。
	数据库账号	填入云原生数据仓库AnalyticDB MySQL的数据库账号。
	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。
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。

配置	说明
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。 ○ 选择为**否**：默认选项。 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，默认情况下都处于选中状态。 ○ **Insert** ○ **Update** ○ **Delete**
选择同步对象	在**源库对象**框中单击待同步的表，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。 ○ 如果需要目标表中列信息与源表不同，则需要使用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	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、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。

2.4.1.3. PolarDB MySQL迁移至AnalyticDB MySQL 3.0

通过数据传输服务DTS（Data Transmission Service），您可以将PolarDB MySQL引擎迁移至云原生数据仓库AnalyticDB MySQL版 3.0，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。
- 已创建目标云原生数据仓库AnalyticDB MySQL版 3.0，详情请参见创建集群。
- 目标云原生数据仓库AnalyticDB MySQL版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	• 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。
	  ◦ 迁移期间，请勿执行添加注释的DDL操作（如 `ALTER TABLE table_name COMMENT='表的注释';` ），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。

注意事项	• 暂不支持迁移前缀索引，如果源库存在前缀索引可能导致数据迁移失败。 • 不支持迁移源PolarDB MySQL引擎只读节点。 • 由于云原生数据仓库AnalyticDB MySQL版 3.0本身的使用限制，当云原生数据仓库AnalyticDB MySQL版 3.0中的节点磁盘空间使用量超过80%，会导致DTS任务异常，产生延迟。请提前根据待迁移的对象预估所需空间，确保目标集群具备充足的存储空间。 • 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。 • DTS会在源库定时执**CREATE DATABASE IF NOT EXISTS `test`**命令以推进Binlog位点。

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** 此场景属于异构数据库间的数据迁移，DTS在执行结构迁移时数据类型无法完全对应，请谨慎评估数据类型的映射关系对业务的影响，详情请参见异构数据库间的数据类型映射关系。

- 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同步到目标库中。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	CREATE TABLE、DROP TABLE、RENAME TABLE、TRUNCATE TABLE、ADD COLUMN、DROP COLUMN

警告 如果在数据迁移过程中变更了源表的字段类型，迁移任务将报错并中断。您可以提交工单处理或参照以下方法手动修复。

1. 在迁移至目标库AnalyticDB MySQL时，源表（例如customer）因字段类型变更而导致迁移任务失败。
2. 在AnalyticDB MySQL 3.0中创建一个新表（customer_new），表结构与customer表保持一致。
3. 通过INSERT INTO SELECT命令，将customer表的数据复制并插入到新创建的customer_new表中，确保两张表的数据保持一致。
4. 重命名或删除迁移失败的表customer，然后将customer_new表的名称修改为customer。
5. 在DTS控制台，重新启动数据迁移任务。

数据库账号的权限要求

数据库	权限要求
PolarDB MySQL引擎	待迁移对象的读权限
AnalyticDB MySQL 3.0	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。
- 云原生数据仓库AnalyticDB MySQL版 3.0，请参见创建数据库账号。

操作步骤

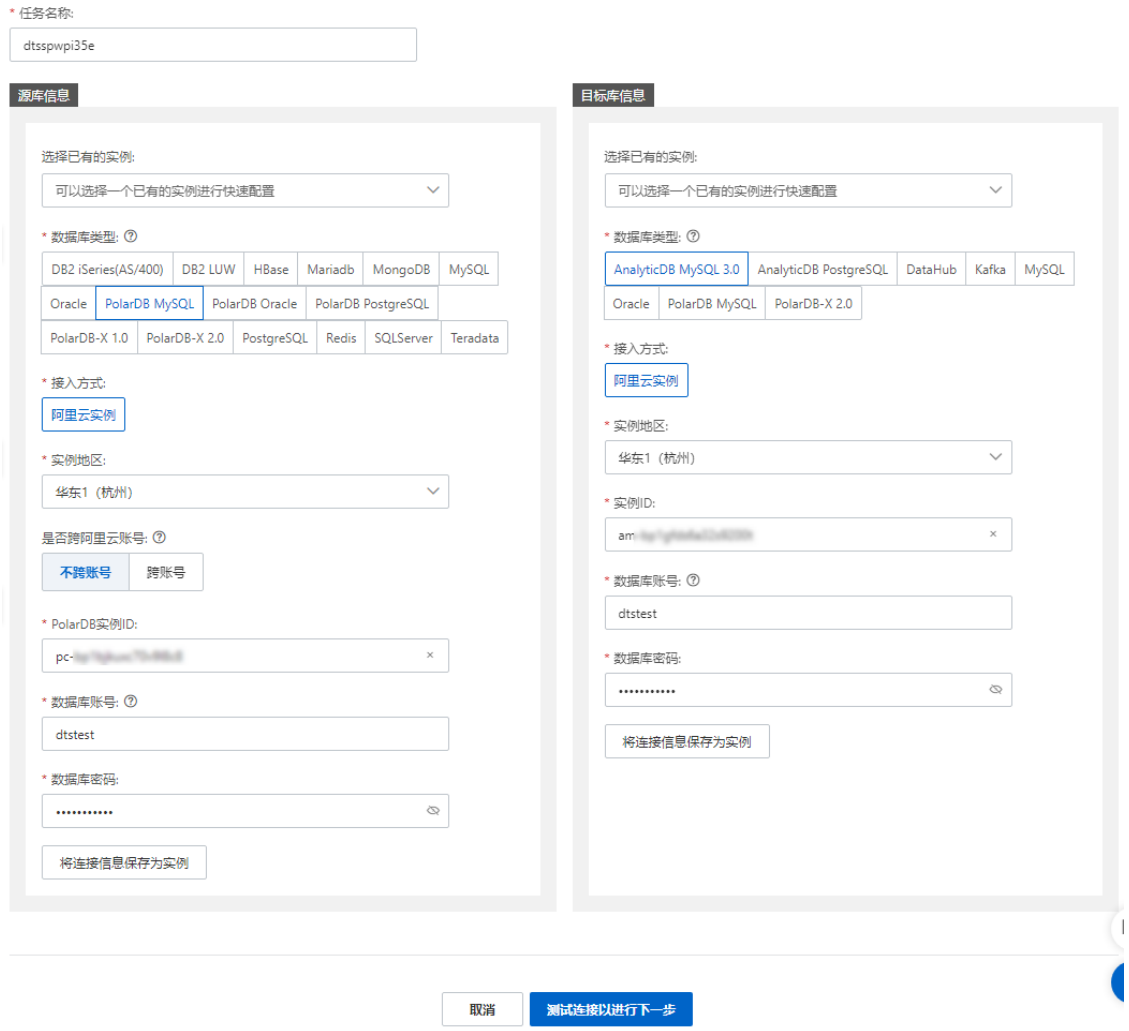
1. 登录新版DTS迁移任务的列表页面。

   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	**选择已有的实例**	您可以按实际需求，选择是否使用已有实例。 ○ 如使用已有实例，数据库信息将自动填入，您无需重复输入。 ○ 如不使用已有实例，您需要输入下方的数据库信息。
	数据库类型	选择**PolarDB MySQL**。
	接入方式	选择为**阿里云实例**。
	实例地区	选择源PolarDB MySQL引擎实例所属地域。
	PolarDB MySQL实例ID	选择源PolarDB MySQL引擎实例ID。
	数据库账号	填入源PolarDB MySQL引擎实例的数据库账号，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。

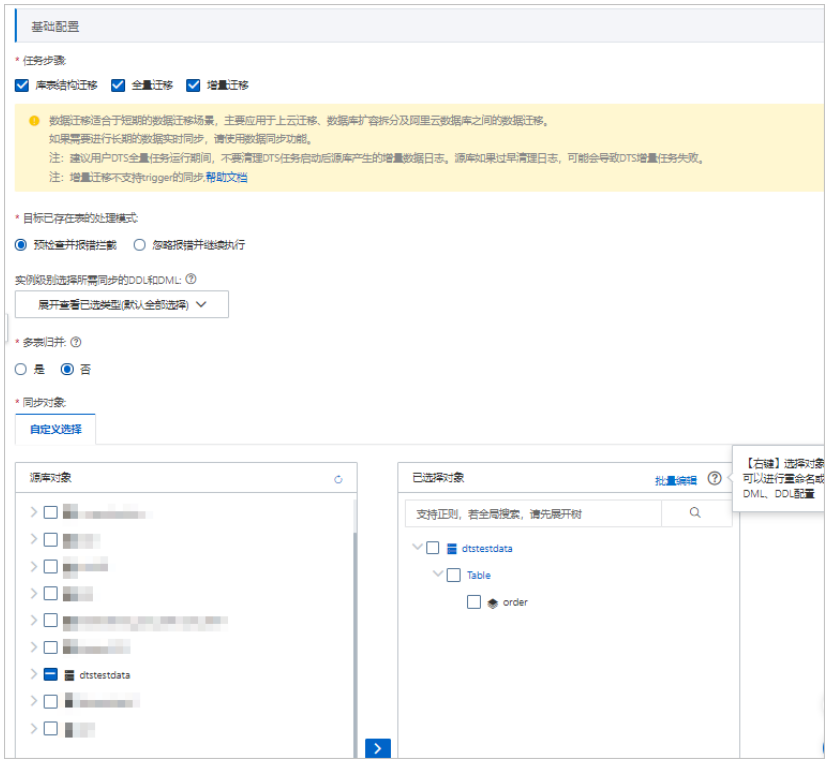
类别	配置	说明
	将连接信息保存为实例或编辑连接信息模板	在**选择已有的实例**处是否选择了已有数据库的连接模板。 i. 是：您可以单击**编辑连接信息模板**自定义连接信息模板的名称。 说明 修改后的连接信息模板，将在下一次选择该连接模板作为实例连接信息时生效，不会影响原先已使用该模板配置的实例连接信息。 ii. 否：单击**将连接信息保存为实例**并在弹跳框中设置连接名称，可将该实例的连接信息保存为模板，便于下次自动传入。 说明 建议连接名称配置为具有业务意义的名称（无唯一性要求），便于后续识别。
目标库信息	**选择已有的实例**	您可以按实际需求，选择是否使用已有实例。 ○ 如使用已有实例，数据库信息将自动填入，您无需重复输入。 ○ 如不使用已有实例，您需要输入下方的数据库信息。
	数据库类型	选择**AnalyticDB 3.0**。
	接入方式	选择**阿里云实例**。
	实例地区	选择目标云原生数据仓库AnalyticDB MySQL版 3.0所属地域。
	实例ID	选择目标云原生数据仓库AnalyticDB MySQL版 3.0集群的ID。
	数据库账号	填入目标云原生数据仓库AnalyticDB MySQL版 3.0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。
	将连接信息保存为实例或编辑连接信息模板	在**选择已有的实例**处是否选择了已有数据库的连接模板。 i. 是：您可以单击**编辑连接信息模板**自定义连接信息模板的名称。 说明 修改后的连接信息模板，将在下一次选择该连接模板作为实例连接信息时生效，不会影响原先已使用该模板配置的实例连接信息。 ii. 否：单击**将连接信息保存为实例**并在弹跳框中设置连接名称，可将该实例的连接信息保存为模板，便于下次自动传入。 说明 建议连接名称配置为具有业务意义的名称（无唯一性要求），便于后续识别。

4. 配置完成后，单击页面下方的**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，请同时选中**库表结构迁移**和**全量迁移**。 ■ 如果需要进行不停机迁移，请同时选中**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。
实例级别选择所需同步的DDL和DML	按实例级别选择增量迁移的SQL操作，支持的操作，请参见支持增量迁移的SQL操作。 **说明** 如需按库或表级别选择增量迁移的SQL操作，请在**已选择对象**中右击同步对象，在弹跳框中勾选所需同步的SQL操作。
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。 ■ 选择为**否**：默认选项。 **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迁移任务。 **警告** 源库请勿执行库或表结构变更的DDL操作，否则会导致数据不一致或者迁移任务失败。

配置	说明
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SQL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SQL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720分钟，您也可以在取值范围（10~720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，网络重试时间以后创建任务的设置为准。 ■ 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. 设置待迁移的表在目标云原生数据仓库AnalyticDB MySQL版中主键列和分布键信息。

说明 选择了**库表结构迁移**后，您需要定义待迁移的表在云原生数据仓库AnalyticDB MySQL版 3.0中的**类型**、**主键列**、**分布键**等信息，详情请参见CREATE TABLE。

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，您可在数据迁移界面查看具体进度。

2.4.2. 通过外表导入数据

2.4.2.1. 通过外表将RDS MySQL数据导入至AnalyticDB MySQL

AnalyticDB MySQL版支持通过外表导入导出数据。本文介绍如何通过AnalyticDB MySQL的外表将云数据库RDS MySQL中的数据导入至AnalyticDB MySQL。

前提条件

- RDS MySQL实例需与AnalyticDB MySQL集群在同一个VPC下，且该VPC网段需加入RDS MySQL实例的白名单中。详细操作步骤，请参见设置RDS的IP白名单。
- 需在RDS MySQL实例上完成创建数据库和连接实例等操作，并准备好相关测试数据。详细操作步骤，请参见创建数据库和账号和连接RDS MySQL实例。

  本文示例中，测试所用的RDS MySQL源库名为 `test_adb` ，并在该库中创建了一张名为 `goods` 的源表，建表语句如下：

```
CREATE TABLE goods (
goods_id bigint(20) NOT NULL,
price double NOT NULL,
class bigint(20) NOT NULL,
name varchar(32) NOT NULL,
update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
PRIMARY KEY (goods_id)
);
```

- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

操作步骤

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 创建目标数据库。详细操作步骤，请参见创建数据库。

   本示例中，AnalyticDB MySQL集群的目标库名为 `adb_demo` 。
3. 创建外部映射表。

   使用以下命令在目标库 `adb_demo` 中创建一张名为 `goods_external_table` 的外部映射表。

```
CREATE TABLE IF NOT EXISTS goods_external_table (
        goods_id bigint(20) NOT NULL,
        price double NOT NULL,
        class bigint(20) NOT NULL,
        name varchar(32) NOT NULL,
        update_time timestamp,
        PRIMARY KEY (goods_id)
 )
        ENGINE='mysql'
        TABLE_PROPERTIES='{
        "url":"jdbc:mysql://mysql-vpc-address:3306/test_adb",
        "tablename":"goods",
        "username":"mysql-user-name",
        "password":"mysql-user-password"
  }';
```

参数	说明
`ENGINE='mysql'`	外部表的存储引擎说明，本文使用的是MySQL。
`TABLE_PROPERTIES`	AnalyticDB MySQL访问RDS MySQL数据的访问方式。
`url`	RDS MySQL实例中的内网地址（即VPC连接地址）和源库名（本文示例中为 `test_adb` ）。RDS地址信息的查看方法，请参见查看或修改内外网地址和端口。 格式： `"jdbc:mysql://mysql-vpc-address:3306/rds-database-name"` 。 示例： `jdbc:mysql://rm-**********.mysql.rds.aliyuncs.com:3306/test_adb` 。
`tablename`	RDS MySQL中源表名，本文示例中为 `goods` 。
`username`	需要访问RDS MySQL源库的账号。
`password`	以上账号对应的密码。

4. 创建目标表。

   使用以下命令在目标数据库 `adb_demo` 中创建一张名为 `mysql_import_test` 的目标表，用于存储从RDS MySQL导入的数据。

```
CREATE TABLE IF NOT EXISTS  mysql_import_test (
        goods_id bigint(20) NOT NULL,
        price double NOT NULL,
        class bigint(20) NOT NULL,
        name varchar(32) NOT NULL,
        update_time timestamp,
        PRIMARY KEY (goods_id)
 )
DISTRIBUTED BY HASH(goods_id);
```

5. 将源RDS MySQL实例中的数据导入至目标AnalyticDB MySQL集群中。

   语法如下：

```
REPLACE INTO mysql_import_test
SELECT * FROM goods_external_table;
```

后续步骤

导入完成后，您可以登录AnalyticDB MySQL的目标库 `adb_demo` 中，执行如下命令查看并验证源表数据是否成功导入至目标 `mysql_import_test` 表中：

```
SELECT * FROM mysql_import_test LIMIT 100;
```

2.4.2.2. 通过外表将自建MySQL数据库导入至AnalyticDB MySQL

AnalyticDB MySQL版支持通过外表导入导出数据。本文介绍如何通过AnalyticDB MySQL的外表将ECS自建MySQL数据库的数据导入至AnalyticDB MySQL。

前提条件

- 自建MySQL需安装在ECS实例上。
- ECS实例需与AnalyticDB MySQL集群在同一个VPC下。
- 需在ECS自建MySQL上完成创建数据库并准备好相关测试数据等操作。

  本文示例中，测试所用的ECS自建MySQL源库名为 `test_adb` ，并在该库中创建了一张名为 `goods` 的源表，建表语句如下：

```
CREATE TABLE goods (
goods_id bigint(20) NOT NULL,
price double NOT NULL,
class bigint(20) NOT NULL,
name varchar(32) NOT NULL,
update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
PRIMARY KEY (goods_id)
);
```

- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

操作步骤

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 创建目标数据库。详细操作步骤，请参见创建数据库。

   本示例中，AnalyticDB MySQL集群的目标库名为 `adb_demo` 。
3. 创建外部映射表。

   使用以下命令在目标库 `adb_demo` 中创建一张名为 `goods_external_table` 的外部映射表。

```
CREATE TABLE IF NOT EXISTS goods_external_table (
        goods_id bigint(20) NOT NULL,
        price double NOT NULL,
        class bigint(20) NOT NULL,
        name varchar(32) NOT NULL,
        update_time timestamp,
        PRIMARY KEY (goods_id)
 )
        ENGINE='mysql'
        TABLE_PROPERTIES='{
        "url":"jdbc:mysql://mysql-vpc-address:3306/test_adb",
        "tablename":"goods",
        "username":"mysql-user-name",
        "password":"mysql-user-password"
  }';
```

参数	说明
`ENGINE='mysql'`	外部表的存储引擎说明，本文使用的是MySQL。
`TABLE_PROPERTIES`	AnalyticDB MySQL访问自建MySQL数据库的访问方式。
`url`	ECS实例的**主私网IP**（即VPC地址）和源库名（本文示例中为 `test_adb` ）。查看步骤如下： i. 登录ECS管理控制台并找到目标实例。 ii. 在**实例详情**页签的**网络信息**区域，查看**主私网IP**。 格式： `"jdbc:mysql://mysql-vpc-address:3306/ecs-database-name"` 。 示例： `jdbc:mysql://192.168.128.***:3306/test_adb` 。
`tablename`	ECS自建MySQL中源表名，本文示例中为 `goods` 。
`username`	需要访问ECS自建MySQL源库的账号。
`password`	以上账号对应的密码。

4. 创建目标表。

使用以下命令在目标数据库 `adb_demo` 中创建一张名为 `mysql_import_test` 的目标表，用于存储从ECS自建MySQL导入的数据。

```
CREATE TABLE IF NOT EXISTS  mysql_import_test (
        goods_id bigint(20) NOT NULL,
        price double NOT NULL,
        class bigint(20) NOT NULL,
        name varchar(32) NOT NULL,
        update_time timestamp,
        PRIMARY KEY (goods_id)
 )
DISTRIBUTED BY HASH(goods_id);
```

5. 将源ECS自建MySQL实例中的数据导入至目标AnalyticDB MySQL集群中。

   语法如下：

   ```
   REPLACE INTO mysql_import_test
   SELECT * FROM goods_external_table;
   ```

后续步骤

导入完成后，您可以登录AnalyticDB MySQL的目标库 `adb_demo` 中，执行如下命令查看并验证源表数据是否成功导入至目标 `mysql_import_test` 表中：

```
SELECT * FROM mysql_import_test LIMIT 100;
```

2.4.3. 通过外表导出数据

2.4.3.1. 通过外表导出AnalyticDB MySQL数据至RDS MySQL

AnalyticDB MySQL版支持通过外表导入导出数据。本文介绍如何通过AnalyticDB MySQL的外表导出数据至云数据库RDS MySQL。

前提条件

- RDS MySQL实例需与AnalyticDB MySQL集群在同一个VPC下，且该VPC网段需加入RDS MySQL实例的白名单中。详细操作步骤，请参见设置RDS的IP白名单。
- 需在RDS MySQL实例上完成创建数据库和连接实例等操作，并准备好相关测试数据。详细操作步骤，请参见创建数据库和账号和连接RDS MySQL实例。

  本文示例中，测试所用的RDS MySQL目标库名为 `test_adb` ，并在该库中创建了一张名为 `courses` 的目标表，用于存储从AnalyticDB MySQL集群中导出的数据。建表语句如下：

  ```
  CREATE TABLE courses (
      id bigint NOT NULL,
      name varchar(32) NOT NULL,
      grade varchar(32) NOT NULL,
      submission_date  timestamp NOT NULL,
      PRIMARY KEY (id)
  );
  ```

- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

操作步骤

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 创建源数据库。详细操作步骤，请参见创建数据库。

   本示例中，AnalyticDB MySQL集群的源库名为 `adb_demo` 。
3. 创建源表并插入源数据。

   使用以下命令在源库 `adb_demo` 中创建一张名为 `courses` 的源表，并将该表中的数据导出至RDS MySQL目标库 `test_adb` 的 `courses` 表中。

   ```
   CREATE TABLE courses (
   id bigint AUTO_INCREMENT,
   name varchar NOT NULL,
   grade varchar DEFAULT '1st Grade',
   submission_date timestamp
   ) DISTRIBUTE BY HASH(id);
   ```

   使用以下命令往源表 `courses` 中插入一行数据：

   ```
   INSERT INTO courses (name,submission_date) VALUES("Jams",NOW());
   ```

4. 创建外部映射表。

使用以下命令在源库 `adb_demo` 中创建一张名为 `courses_external_table` 的外部映射表。

```
CREATE TABLE IF NOT EXISTS courses_external_table(
id bigint NOT NULL,
name varchar(32) NOT NULL,
grade varchar(32) NOT NULL,
submission_date  timestamp NOT NULL,
PRIMARY KEY (id)
)
ENGINE='mysql'
TABLE_PROPERTIES='{
"url":"jdbc:mysql://mysql-vpc-address:3306/test_adb",
"tablename":"courses",
"username":"mysql-user-name",
"password":"mysql-user-password"
}';
```

参数	说明
`ENGINE='mysql'`	外部表的存储引擎说明，本文使用的是MySQL。
`TABLE_PROPERTIES`	AnalyticDB MySQL访问RDS MySQL数据的访问方式。
`url`	RDS MySQL实例中的内网地址（即VPC连接地址）和目标库名（本文示例中为 `test_adb` ），查看RDS地址信息的方法，请参见查看或修改内外网地址和端口。 格式： `"jdbc:mysql://mysql-vpc-address:3306/rds-database-name"` 。 示例： `jdbc:mysql://rm-***************.mysql.rds.aliyuncs.com:3306/test_adb` 。
`tablename`	RDS MySQL中目标表名，本文示例中为 `courses` 。
`username`	需要访问RDS MySQL目标库的账号。
`password`	以上账号对应的密码。

5. 将源AnalyticDB MySQL集群中的数据导入至目标RDS MySQL实例中。

   语法如下：

   ```
   REPLACE INTO courses_external_table
   SELECT * FROM courses;
   ```

后续步骤

导入完成后，您可以登录RDS MySQL的目标 `test_adb` 库中，执行如下命令查看并验证源表数据是否成功导入至目标 `courses` 表中：

```
SELECT * FROM courses LIMIT 100;
```

2.4.3.2. 通过外表导出AnalyticDB MySQL数据至自建MySQL

AnalyticDB MySQL版支持通过外表导入导出数据。本文介绍如何通过AnalyticDB MySQL的外表导出数据至ECS自建MySQL。

前提条件

- 自建MySQL需安装在ECS实例上。
- ECS实例需与AnalyticDB MySQL集群在同一个VPC下。
- 需在ECS自建MySQL上完成创建数据库并准备好相关测试数据等操作。

  本文示例中，测试所用的ECS自建MySQL目标库名为 `test_adb` ，并在该库中创建了一张名为 `courses` 的目标表，用于存储从AnalyticDB MySQL集群中导出的数据。建表语句如下：

  ```
  CREATE TABLE courses (
      id bigint NOT NULL,
      name varchar(32) NOT NULL,
      grade varchar(32) NOT NULL,
      submission_date  timestamp NOT NULL,
      PRIMARY KEY (id)
  );
  ```

- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

操作步骤

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 创建源数据库。详细操作步骤，请参见创建数据库。

   本示例中，AnalyticDB MySQL集群的源库名为 `adb_demo` 。
3. 创建源表并插入源数据。

   使用以下命令在源库 `adb_demo` 中创建一张名为 `courses` 的源表，并将该表中的数据导出至自建MySQL目标库 `test_adb` 的 `courses` 表中。

```
CREATE TABLE courses (
id bigint AUTO_INCREMENT,
name varchar NOT NULL,
grade varchar DEFAULT '1st Grade',
submission_date timestamp
) DISTRIBUTE BY HASH(id);
```

   使用以下命令往源表 `courses` 中插入一行数据：

```
INSERT INTO courses (name,submission_date) VALUES("Jams",NOW());
```

4. 创建外部映射表。

   使用以下命令在源库 `adb_demo` 中创建一张名为 `courses_external_table` 的外部映射表。

```
 CREATE TABLE IF NOT EXISTS courses_external_table(
 id bigint NOT NULL,
 name varchar(32) NOT NULL,
 grade varchar(32) NOT NULL,
 submission_date  timestamp NOT NULL,
 PRIMARY KEY (id)
 )
ENGINE='mysql'
TABLE_PROPERTIES='{
"url":"jdbc:mysql://mysql-vpc-address:3306/test_adb",
"tablename":"courses",
"username":"mysql-user-name",
"password":"mysql-user-password"
}';
```

参数	说明
`ENGINE='mysql'`	外部表的存储引擎说明，本文使用的是MySQL。
`TABLE_PROPERTIES`	AnalyticDB MySQL访问自建MySQL数据库的访问方式。
`url`	ECS实例的**主私网IP**（即VPC地址）和目标库名（本文示例中为 `test_adb` ）。查看步骤如下： i. 登录ECS管理控制台并找到目标实例。 ii. 在**实例详情**页签的**网络信息**区域，查看**主私网IP**。 格式： `"jdbc:mysql://mysql-vpc-address:3306/ecs-database-name"` 。 示例： `jdbc:mysql://192.168.128.***:3306/test_adb` 。
`tablename`	ECS自建MySQL中目标表名，本文示例中为 `courses` 。
`username`	需要访问ECS自建MySQL目标库的账号。
`password`	以上账号对应的密码。

5. 将源AnalyticDB MySQL集群中的数据导出至目标自建MySQL中。

   语法如下：

```
REPLACE INTO courses_external_table
SELECT * FROM courses;
```

后续步骤

您可以登录自建MySQL的目标 `test_adb` 库中，执行如下命令查看并验证源表数据是否成功导入至目标 `courses` 表中：

```
SELECT * FROM courses LIMIT 100;
```

2.5. OSS

2.5.1. 使用DataWorks导入OSS数据

2.5.1.1. 配置OSS数据源

本文介绍如何在DataWorks中配置OSS数据源。

操作步骤

1. 登录DataWorks控制台。
2. 单击左侧导航栏**工作空间列表**。
3. 单击操作栏中的**进入数据集成**。
4. 在**数据集成**页面，单击左侧导航栏的**数据源>数据列表**。
5. 在页面右上角单击**新增数据源**。
6. 在**新增数据源**页面，选择**OSS**。
7. 在**新增OSS数据源**页面，按照页面提示进行参数配置。

参数	说明
数据源名称	为数据源指定一个名字，便于后续管理。
数据源描述	添加数据源描述，该项为可选填项。
Endpoint	OSS EndPoint（地域节点），格式为 `http://oss.aliyuncs.com` 。
Bucket	OSS Bucket名字。
Access Id	OSS Bucket创建者的Access Id。
AccessKey Secret	AccessKeyID对应的Key。如何获取AK信息，请参见获取账号的AK信息。
资源组连通性	根据需要选择**数据集成**或**任务调度**。

8. 完成上述参数配置后，进行连通性测试，测试通过后单击**完成**。

2.5.1.2. 配置AnalyticDB for MySQL数据源

您可以参照数据集成同步RDS MySQL数据中的操作步骤完成AnalyticDB for MySQL数据源配置。

详情请参见配置AnalyticDB MySQL 3.0数据源。

2.5.1.3. 配置同步任务中的数据来源和去向

您可以参照DataWorks同步RDS MySQL中的配置同步任务步骤，将OSS中的数据同步至AnalyticDB for MySQL。

详情请参见使用DataWorks同步RDS MySQL数据。

2.5.2. 通过外表将OSS数据导入至AnalyticDB MySQL

本文介绍如何通过AnalyticDB MySQL版的外部映射表直接查询OSS数据文件，以及如何将OSS中的数据文件导入AnalyticDB MySQL版。目前支持的OSS数据文件格式有Parquet、CSV和ORC。

前提条件

- 通过以下步骤在对象存储（Object Storage Service，简称OSS）中创建存储AnalyticDB MySQL版数据的目录。
  i. 开通OSS服务

  > **说明** OSS与AnalyticDB MySQL版所属地域相同。

  ii. 创建存储空间
  iii. 创建目录
  iv. 上传测试数据文件

  本示例将 `oss_import_test_data.txt` 文件上传至OSS中的 `<bucket-name>.oss-cn-hangzhou.aliyuncs.com/adb/` 目录，数据行分隔符为换行符，列分隔符为 `;` ，文件示例数据如下所示：

```
uid;other
12;hello_world_1
27;hello_world_2
28;hello_world_3
33;hello_world_4
37;hello_world_5
40;hello_world_6
...
```

- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

操作步骤

本示例将 `oss_import_test_data.txt` 中的数据导入AnalyticDB MySQL版的 `adb_demo` 库中。

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 创建目标数据库。详细操作步骤，请参见创建数据库。

   本示例中，AnalyticDB MySQL集群的目标库名为 `adb_demo` 。
3. 创建外部映射表。使用 `CREATE TABLE` 语句在目标库 `adb_demo` 中创建CSV、Parquet或ORC格式的OSS外部映射表。具体语法，请参见创建OSS外表语法。
4. 查询OSS数据（若仅需导入数据则可跳过此步骤）。

   查询外表映射表和查询AnalyticDB MySQL版内表语法没有区别，您可以方便地直接进行查询，查询语句如下：

```
SELECT uid, other FROM oss_import_test_external_table WHERE uid < 100 LIMIT 10;
```

   > **说明**
   > - 对于数据量较大的CSV格式数据文件，强烈建议您按照后续步骤导入AnalyticDB MySQL版后再做查询，否则查询性能可能会较差。
   > - 对于Parquet和ORC格式数据文件，直接查询的性能一般也比较高，您可以根据需要决定是否进一步导入到AnalyticDB MySQL版后再做查询。

5. 创建目标表。您可以在目标数据库 `adb_demo` 中创建一张目标表 `adb_oss_import_test` ，用于存储从OSS导入的数据。建表语句如下：

```
CREATE TABLE IF NOT EXISTS adb_oss_import_test
(
    uid string,
    other string
)
DISTRIBUTED BY HASH(uid);
```

6. 执行**INSERT**语句将OSS数据导入AnalyticDB MySQL版。
   - 方式一：执行**INSERT INTO**导入数据。

```
INSERT INTO adb_oss_import_test
SELECT * FROM oss_import_test_external_table;
```

- 方式二：执行**INSERT OVERWRITE INTO**导入数据。

```
INSERT OVERWRITE INTO adb_oss_import_test
SELECT * FROM oss_import_test_external_table;
```

- 方式三：异步执行**INSERT OVERWRITE INTO**导入数据。

```
SUBMIT job INSERT OVERWRITE INTO adb_oss_import_test
SELECT * FROM oss_import_test_external_table;
```

返回结果如下：

```
+----------------------------------------+
| job_id                                 |
+----------------------------------------+
| 2020112122202917203100908203303****** |
```

关于异步提交任务详情，请参见异步提交导入导出任务。

创建OSS外表语法

- **创建OSS CSV格式外表**

语法如下。

```
CREATE TABLE IF NOT EXISTS oss_import_test_external_table
(
    uid string,
    other string
)
ENGINE='OSS'
TABLE_PROPERTIES='{
    "endpoint":"oss-cn-hangzhou-internal.aliyuncs.com",
    "url":"oss://<bucket-name>/adb/oss_import_test_data.txt",
    "accessid":"LTAIF****5FsE",
    "accesskey":"Ccw****iWjv",
    "delimiter":";",
    "skip_header_line_count":1
}';
```

参数	是否必填	说明
ENGINE='OSS'	必填	表示该表是外部表，使用的存储引擎是OSS。
TABLE_PROPERTIES		用于告知AnalyticDB MySQL版如何访问OSS中的数据。
endpoint		OSS的**EndPoint（地域节点）**。 **说明** 目前仅支持AnalyticDB MySQL版通过ECS的VPC网络访问OSS。 登录OSS控制台，单击目标Bucket，在Bucket**概览**页面查看**EndPoint（地域节点）**。
url		OSS中源数据文件或文件夹的绝对地址。建议文件夹地址以 `/` 结尾。 示例： ○ 文件： `oss://<bucket-name>/adb/oss_import_test_data.txt` 。 ○ 文件夹： `oss://<bucket-name>/adb_data/` 。
accessid		您在访问OSS中的文件或文件夹时所持有的AccessKey ID。 如何获取您的AccessKey ID和AccessKey Secret，请参见获取账号的AK信息。
accesskey		您在访问OSS中的文件或文件夹时所持有的AccessKey Secret。
delimiter		定义CSV数据文件的列分隔符。例如您可以将列分隔符设置为英文逗号（,）。
null_value	选填	定义CSV数据文件的 `NULL` 值。默认将空值定义为 `NULL` ，即 `"null_value": ""` 。 **说明** AnalyticDB MySQL版集群需为V3.1.4.2或以上版本才支持配置该参数。关于版本信息，请参见新功能发布记录。

参数	是否必填	说明
ossnull	选填	选择CSV数据文件中 `NULL` 值的对应规则。取值范围如下： ○ *1*（默认值）：表示 `EMPTY_SEPARATORS` ，即仅将空值定义为 `NULL` 。 示例： `a,"",,c --> "a","",NULL,"c"` ○ *2*：表示 `EMPTY_QUOTES` ，即仅将 `""` 定义为 `NULL` 。 示例： `a,"",,c --> "a",NULL,"","c"` ○ *3*：表示 `BOTH` ，即同时将空值和 `""` 定义为 `NULL` 。 示例： `a,"",,c --> "a",NULL,NULL,"c"` ○ *4*：表示 `NEITHER` ，即空值和 `""` 均不定义为 `NULL` 。 示例： `a,"",,c --> "a","","","c"` **说明** 上述各示例的前提为 `"null_value": ""` 。
skip_header_line_count		定义导入数据时需要在开头跳过的行数。CSV文件第一行为表头，若设置该参数为*1*，导入数据时可自动跳过第一行的表头。 默认取值为*0*，即不跳过。
oss_ignore_quote_and_escape		是否忽略字段值中的引号和转义。默认取值为 `false` ，即不忽略字段值中的引号和转义。 **说明** AnalyticDB MySQL版集群需为V3.1.4.2或以上版本才支持设置该参数。关于版本信息，请参见新功能发布记录。

AnalyticDB MySQL支持通过OSS的CSV格式的外表读写Hive TEXT文件。建表语句示例如下：

```
CREATE TABLE adb_csv_hive_format_oss (
  a tinyint,
  b smallint,
  c int,
  d bigint,
  e boolean,
  f float,
  g double,
  h varchar,
  i varchar, -- binary
  j timestamp,
  k DECIMAL(10, 4),
  l varchar, -- char(10)
  m varchar, -- varchar(100)
  n date
) ENGINE = 'OSS' TABLE_PROPERTIES='{
    "format": "csv",
    "endpoint":"oss-cn-hangzhou-internal.aliyuncs.com",
    "accessid":"LTAIF****5FsE",
    "accesskey":"Ccw****iWjv",
    "url":"oss://<bucket-name>/adb/adb_csv_hive_format_oss.txt",
    "delimiter": "\\1",
    "null_value": "\\\\N",
    "oss_ignore_quote_and_escape": "true",
    "ossnull": 2,
}';
```

说明 在创建OSS的CSV格式的外表来读取Hive TEXT文件时，需注意如下几点：

- Hive TEXT文件的默认列分隔符为 `\1` 。若您需要通过OSS的CSV格式的外表读写Hive TEXT文件，您可以在配置 `delimiter` 参数时将其转义为 `\\1` 。
- Hive TEXT文件的默认 `NULL` 值为 `\N` 。若您需要通过OSS的CSV格式的外表读写Hive TEXT文件，您可以在配置 `null_value` 参数时将其转义为 `\\\\N` 。
- Hive的其他基本数据类型（如 `BOOLEAN` ）与AnalyticDB MySQL版的数据类型一一对应，但 `BINARY` 、 `CHAR(n)` 和 `VARCHAR(n)` 类型均对应AnalyticDB MySQL版中的 `VARCHAR` 类型。

- **创建OSS Parquet格式/OSS ORC格式外表**

  以Parquet格式为例，创建OSS外表的语句如下：

```
CREATE TABLE IF NOT EXISTS oss_import_test_external_table
(
    uid string,
    other string
)
ENGINE='OSS'
TABLE_PROPERTIES='{
    "endpoint":"oss-cn-hangzhou-internal.aliyuncs.com",
    "url":"oss://<bucket-name>/adb/oss_import_test_data.txt",
    "accessid":"LTAIF****5FsE",
    "accesskey":"Ccw****iWjv",
    "format":"parquet"
}';
```

参数	说明
ENGINE='OSS'	表示该表是外部表，使用的存储引擎是OSS。
TABLE_PROPERTIES	用于告知AnalyticDB MySQL版如何访问OSS中的数据。
endpoint	OSS的**EndPoint（地域节点）**（域名节点）。 说明 目前仅支持AnalyticDB MySQL版通过OSS中ECS的VPC网络（内网）访问OSS。 登录OSS控制台，单击目标Bucket，在Bucket**概览**页面查看**EndPoint（地域节点）**。
url	OSS中源数据文件或文件夹的绝对地址。建议文件夹地址以 `/` 结尾。 示例： ○ 文件： `oss://<bucket-name>/adb/oss_import_test_data.txt` 。 ○ 文件夹： `oss://<bucket-name>/adb_data/` 。
accessid	您在访问OSS中的文件或文件夹时所持有的AccessKey ID。 如何获取您的AccessKey ID和AccessKey Secret，请参见获取账号的AK信息。
accesskey	您在访问OSS中的文件或文件夹时所持有的AccessKey Secret。
format	数据文件的格式。 ○ 创建Parquet格式文件的外表时，必须将其设置为 `parquet` 。 ○ 创建ORC格式文件的外表时，必须将其设置为 `orc` 。

注意
- Parquet和ORC格式外表定义中列的名称应与Parquet和ORC文件的中该列的名称必须完全相同（可忽略大小写），建议列的顺序保持一致。
- Parquet和ORC格式外表定义中的列可以只选择Parquet和ORC文件的部分列，未被外表定义的Parquet和ORC文件的列将被忽略；反之如果定义了Parquet和ORC文件中未包含的列，该列的查询结果为NULL。

Parquet文件与AnalyticDB MySQL版3.0的数据类型映射关系如下表。

Parquet基本类型	Parquet的logicalType类型	AnalyticDB MySQL版3.0中对应的数据类型
BOOLEAN	无	BOOLEAN
INT32	INT_8	TINYINT
INT32	INT_16	SMALLINT
INT32	无	INT或INTEGER
INT64	无	BIGINT
FLOAT	无	FLOAT
DOUBLE	无	DOUBLE
○ FIXED_LEN_BYTE_ARRAY ○ BINARY ○ INT64 ○ INT32	DECIMAL	DECIMAL
BINARY	UTF-8	○ VARCHAR ○ STRING ○ JSON（如果已知Parquet该列内容为JSON格式）

Parquet基本类型	Parquet的logicalType类型	AnalyticDB MySQL版3.0中对应的数据类型
INT32	DATE	DATE
INT64	TIMESTAMP_MILLIS	TIMESTAMP或DATETIME
INT96	无	TIMESTAMP或DATETIME

> 注意 Parquet格式外表暂不支持 `STRUCT` 类型，会导致建表失败。

ORC文件与AnalyticDB MySQL版3.0的数据类型映射关系如下表。

ORC文件中的数据类型	AnalyticDB MySQL版3.0中对应的数据类型
BOOLEAN	BOOLEAN
BYTE	TINYINT
SHORT	SMALLINT
INT	INT或INTEGER
LONG	BIGINT
DECIMAL	DECIMAL
FLOAT	FLOAT
DOUBLE	DOUBLE
◦ BINARY ◦ STRING ◦ VARCHAR	◦ VARCHAR ◦ STRING ◦ JSON（如果已知ORC该列内容为JSON格式）
TIMESTAMP	TIMESTAMP或DATETIME
DATE	DATE

> 注意 ORC格式外表暂不支持 `LIST` 、 `STRUCT` 和 `UNION` 等复合类型，会导致建表失败；ORC格式外表的列使用 `MAP` 类型可以建表，但ORC的查询会失败。

针对带有分区的数据文件创建OSS外表

如果OSS数据源是包含分区的，会在OSS上形成一个分层目录，类似如下内容：

```
parquet_partition_classic/
├── p1=2020-01-01
│   ├── p2=4
│   │   ├── p3=SHANGHAI
│   │   │   ├── 000000_0
│   │   │   └── 000000_1
│   │   └── p3=SHENZHEN
│   │       └── 000000_0
│   └── p2=6
│       └── p3=SHENZHEN
│           └── 000000_0
├── p1=2020-01-02
│   └── p2=8
│       ├── p3=SHANGHAI
│       │   └── 000000_0
│       └── p3=SHENZHEN
│           └── 000000_0
└── p1=2020-01-03
    └── p2=6
        ├── p2=HANGZHOU
        └── p3=SHENZHEN
            └── 000000_0
```

上述数据中p1为第1级分区，p2为第2级分区，p3为第3级分区。对应这种数据源，一般都希望以分区的模式进行查询，那么就需要在创建OSS外表时指明分区列。以Parquet格式为例，创建带有分区的OSS外表的语句如下：

```
CREATE TABLE IF NOT EXISTS oss_parquet_partition_table
(
  uid varchar,
  other varchar,
  p1 date,
  p2 int,
  p3 varchar
)
ENGINE='OSS'
TABLE_PROPERTIES='{
  "endpoint":"oss-cn-hangzhou-internal.aliyuncs.com",
  "url":"oss://<bucket-name>/adb/oss_parquet_data_dir",
  "accessid":"LTAIF****5FsE",
  "accesskey":"Ccw****iWjv",
  "format":"parquet",
  "partition_column":"p1, p2, p3"
}';
```

说明

- `TABLE_PROPERTIES` 中的partition_column属性必须声明分区列（本例中的p1，p2，p3）且partition_column属性里必须严格按照第1级, 第2级, 第3级的顺序声明（本例中p1为第1级分区，p2为第2级分区，p3为第3级分区）。
- 列定义中必须定义分区列（本例中的p1，p2，p3）及类型，且分区列需要置于列定义的末尾。
- 列定义中分区列的先后顺序需要与partition_column中分区列的顺序保持一致。
- 可以作为分区列的数据类型有：`BOOLEAN` 、`TINYINT` 、`SMALLINT` 、`INT` 、`INTEGER` 、`BIGINT` 、`FLOAT` 、`DOUBLE` 、`DECIMAL` 、`VARCHAR` 、`STRING` 、`DATE` 、`TIMESTAMP` 。
- 查询时分区列和其它数据列的表现和用法没有区别。

2.5.3. 通过外表导出AnalyticDB MySQL数据至OSS

云原生数据仓库AnalyticDB MySQL版支持通过外表和INSERT INTO方式将AnalyticDB MySQL版中的数据导出到对象存储OSS（Object Storage Service）中。将数据导出到OSS功能只支持CSV和Parquet格式文件。

前提条件

- 在对象存储OSS中创建存储AnalyticDB MySQL版数据的目录：
  i. 开通OSS服务。

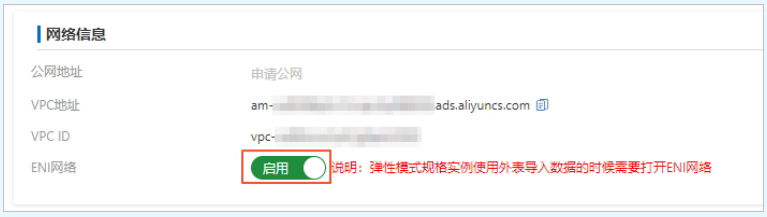
  说明 OSS与AnalyticDB MySQL版所属Region相同。

  ii. 创建存储空间。
  iii. 在OSS中新建目录。

  例如，在OSS中新建目录 `adb_data/` ，从AnalyticDB MySQL版中导出的数据将存储在该目录下。

- 完成创建集群、设置白名单、创建账号和数据库等准备工作。详情请参见AnalyticDB for MySQL快速入门。

说明 如果AnalyticDB MySQL版集群是弹性模式，请先登录AnalyticDB MySQL控制台，查看集群信息，在网络信息栏启用ENI（Elastic Network Interface，弹性网卡）网络。



操作步骤

本示例将AnalyticDB MySQL版的 `adb_demo` 库中的 `source_table` 表数据导出至OSS的 `adb_data` 文件夹下。

1. 连接目标集群，进入源数据库。
2. 在 `adb_demo` 数据库中创建外表，详情请参见创建OSS外表语法。

3. 根据外表类型选择执行写入语句，将源数据写入到步骤2创建的外表中。不同的外表类型支持的语法请参见未做分区的普通外表语法支持和分区外表语法支持。
4. 待步骤3的写入任务结束后，您可登录OSS控制台，在目标文件夹下查看导出到OSS的数据文件。您也可以直接通过AnalyticDB MySQL版查询导出到外表的数据。

未做分区的普通外表语法支持

INSERT SELECT FROM

功能：如果您的数据在其他表中已经存在，可以通过 `INSERT SELECT FROM` 将数据复制到外表。将源表的数据写入外表对应的OSS位置，每次写入会产生新的OSS数据文件。

> **说明** 写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列。

语法：

```
INSERT INTO table_name
SELECT select_statement FROM from_statement;
```

例句：

```
insert into oss_table select col1, col2, col3 from source_table;
```

REPLACE SELECT FROM

功能：由于OSS外表不支持定义主键，因此 `REPLACE SELECT FROM` 的写入表现与 `INSERT SELECT FROM` 保持一致，都是将数据复制到另外一张表；如果目标表已有数据，已有数据保持不变，新数据追加到新的OSS数据文件。

INSERT OVERWRITE INTO SELECT

功能： `INSERT OVERWRITE INTO SELECT` 用于向表中批量插入数据。如果目标外表中已存在数据，则每次写入会先删除原外表路径下全部数据文件，再产生新的OSS数据文件。

> **说明** 写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列。

语法：

```
INSERT OVERWRITE INTO table_name
SELECT select_statement FROM from_statement;
```

例句：

```
insert overwrite into oss_table
select col1, col2, col3 from source_table;
```

导出到OSS单文件（仅限CSV格式，Parquet格式不允许导出到单一文件）

功能：通过hint指定唯一的OSS文件，将数据导出到此文件中。包含overwrite关键字时，覆盖外表TABLE_PROPERTIES中定义的目录下旧的同名文件，不影响该目录下其他文件。

> **说明** 写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列。

版本说明：

- 3.1.2版本之前的AnalyticDB MySQL版集群：不支持此功能，文件名由系统自动命名，将会导出多个文件。根据任务并发速度动态确定目标文件数目。
- 3.1.2及之后版本的AnalyticDB MySQL版集群：支持通过hint将AnalyticDB MySQL版CSV格式的普通外表导出到OSS单文件，用户在导出时可自定义文件名。不带hint的情况下，与3.1.2之前的版本保持一致（导出多个文件）。

语法：

```
/*+output_filename=adb.txt*/INSERT [OVERWRITE] table_name
SELECT select_statement FROM from_statement;
```

例句：

```
/*+output_filename=adb.txt*/INSERT [OVERWRITE] oss_table
SELECT * FROM source_table;
```

未带hint导出时：

导出到单一文件时：

分区外表语法支持

分区表写出数据时，数据文件内不包含分区列的数据，分区列的数据信息以OSS目录的形式展现。

例如：分区表中定义了2个分区列，3个普通列。其中一级分区名称为pcol1，分区数值为1；二级分区名称为pcol2，分区数值为a。分区表数据导出到OSS的保存路径为*adb_data/*，则向pcol1=1且pcol2=a的外表分区写出数据时，数据文件相对路径目录为：*adb_data/pcol1=1/pcol2=a/*；且外表CSV/Parquet数据文件内不包含pcol1与pcol2这两列的值，只包含3列普通列的值。

> 说明 分区外表不支持导出到OSS单文件。

INSERT INTO PARTITION SELECT FROM

功能：`INSERT INTO PARTITION SELECT FROM` 用于向带分区的外表中批量插入数据。写入时在PARTITION字段中指明所有分区列和分区值；也可以只写明高层分区目录的分区值，低层分区动态生成；或完全不写分区值，所有层次分区动态生成，此时也就可以无需PARTITION字段。

写入时，数据将在对应分区追加写入，每次写入会产生新的OSS数据文件。

写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列。

全静态分区

语法：

```
INSERT into table_name PARTITION(par1=val1,par2=val2,...)
SELECT select_statement FROM from_statement;
```

例句：

```
insert into oss_table_par PARTITION(par1=val1,par2=val2)
select col1, col2, col3, from source_table;
```

半静态半动态分区

语法

```
INSERT into table_name PARTITION(par1=val1,par2,...)
SELECT select_statement FROM from_statement;
```

例句

```
insert into oss_table_par PARTITION(par1=val1,par2)
select col1, col2, col3, par2col from source_table;
```

全动态分区

语法

```
INSERT into table_name
SELECT select_statement FROM from_statement;
```

例句

```
insert into oss_table_par
select col1, col2, col3, par1col, par2col from source_table;
```

REPLACE INTO PARTITION SELECT FROM

功能：由于外表不支持主键，`REPLACE INTO PARTITION SELECT FROM` 行为表现与 `INSERT INTO PARTITION SELECT FROM` 一样。

INSERT OVERWRITE [INTO] PARTITION SELECT

功能：和 `INSERT INTO PARTITION SELECT` 使用方法相同，功能上的不同在于会清除掉本次执行中涉及到的目标分区中之前已有的数据文件，对于没有新数据写入的分区，则不会清除其中的数据文件。

语法：

```
INSERT OVERWRITE [INTO] table_name PARTITION(par1=val1,par2=val2,...)[IF NOT EXISTS]
SELECT select_statement FROM from_statement;
```

例句：

```
INSERT OVERWRITE into oss_table_par PARTITION(par1=val1,par2=val2) IF NOT EXISTS
select col1, col2, col3 from source_table;
```

不支持语法

AnalyticDB MySQL版 3.0 不支持自定义行级写入的插入语法，具体不支持的语法包括：`INSERT INTO VALUES` 和 `REPLACE INTO VALUES` 。

2.6. HDFS

2.6.1. 通过外表将HDFS数据导入至AnalyticDB MySQL

AnalyticDB MySQL版支持通过外表导入导出数据。本文介绍如何通过AnalyticDB MySQL的外部映射表查询HDFS数据，并将HDFS数据导入至AnalyticDB MySQL。

前提条件

- AnalyticDB MySQL版集群需为V3.1.4.4或以上版本。

  > 说明
  > - 如何查看集群版本，请参见查看版本。
  > - 如需升级版本，请提交工单。

- HDFS数据文件格式需为CSV、Parquet或ORC。
- 已创建HDFS集群并在HDFS文件夹中准备需要导入的数据，本文示例中所用文件夹为*hdfs_import_test_data.csv*。
- 已在HDFS集群中为AnalyticDB MySQL集群配置如下服务访问端口：
  - `namenode` ：用于读写文件系统元信息。您可以在fs.defaultFS参数中配置端口号，默认端口号为8020。

    详细配置方式，请参见core-default.xml。
  - `datanode` ：用于读写数据。您可以在dfs.datanode.address参数中配置端口号，默认端口号为50010。

    详细配置方式，请参见hdfs-default.xml。
- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

操作步骤

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 创建目标数据库。详细操作步骤，请参见创建数据库。

   本示例中，AnalyticDB MySQL集群的目标库名为 `adb_demo` 。
3. 使用 `CREATE TABLE` 语句在目标库 `adb_demo` 中创建CSV、Parquet或ORC格式的外部映射表。
   - 创建普通外部映射表。具体语法，请参见创建HDFS外表。
   - 创建带分区外部映射表。具体语法，请参见创建带分区的HDFS外表。

4. 创建目标表。

   您可以使用以下语句在目标数据库 `adb_demo` 中创建一张目标表，用于存储从HDFS导入的数据：

   - 创建普通外部映射表对应的目标表（本文示例中目标表名为 `adb_hdfs_import_test` ），语法如下。

     ```
     CREATE TABLE IF NOT EXISTS adb_hdfs_import_test
     (
         uid string,
         other string
     )
     DISTRIBUTED BY HASH(uid);
     ```

   - 创建带分区外部映射表对应的目标表时（本文示例中目标表名为 `adb_hdfs_import_parquet_partition` ），需要同时在创建语句中定义普通列（如 `uid` 和 `other` ）和分区列（如 `p1` 、 `p2` 和 `p3` ），语法如下。

     ```
     CREATE TABLE IF NOT EXISTS adb_hdfs_import_parquet_partition
     (
         uid string,
         other string,
         p1 date,
         p2 int,
         p3 varchar
     )
     DISTRIBUTED BY HASH(uid);
     ```

5. 将HDFS中的数据导入至目标AnalyticDB MySQL集群中。

   您可以根据业务需要选择如下几种方式导入数据（分区表导入数据语法与普通表一致，如下示例中以普通表为例）：

   - （推荐）方式一：使用 `INSERT OVERWRITE INTO` 导入数据。数据批量导入，性能好。导入成功后数据可见，导入失败数据会回滚，示例如下。

     ```
     INSERT OVERWRITE INTO adb_hdfs_import_test
     SELECT * FROM hdfs_import_test_external_table;
     ```

   - 方式二：使用 `INSERT INTO` 导入数据。数据插入实时可查，数据量较小时使用，示例如下。

     ```
     INSERT INTO adb_hdfs_import_test
     SELECT * FROM hdfs_import_test_external_table;
     ```

   - 方式三：异步执行导入数据，示例如下。

     ```
     SUBMIT JOB INSERT OVERWRITE INTO adb_hdfs_import_test
     SELECT * FROM hdfs_import_test_external_table;
     ```

     返回结果如下。

     ```
     +---------------------------------------+
     | job_id                                |
     +---------------------------------------+
     | 2020112122202917203100908203303****** |
     +---------------------------------------+
     ```

     您还可以根据上述 `job_id` 查看异步任务的状态，更多详情，请参见异步提交导入导出任务。

后续步骤

导入完成后，您可以登录AnalyticDB MySQL的目标库 `adb_demo` 中，执行如下语句查看并验证源表数据是否成功导入至目标表 `adb_hdfs_import_test` 中：

```
SELECT * FROM adb_hdfs_import_test LIMIT 100;
```

创建HDFS外表

- **创建文件格式为CSV的外表**

  语句如下：

  ```
  CREATE TABLE IF NOT EXISTS hdfs_import_test_external_table
  (
      uid string,
      other string
  )
  ENGINE='HDFS'
  TABLE_PROPERTIES='{
      "format":"csv",
      "delimiter":",",
      "hdfs_url":"hdfs://172.17.***.***:9000/adb/hdfs_import_test_csv_data/hdfs_import_test_data.csv"
  }';
  ```

参数	是否必填	说明
`ENGINE='HDFS'`		外部表的存储引擎说明。本示例使用的存储引擎为HDFS。
`TABLE_PROPERTIES`		AnalyticDB MySQL访问HDFS数据的方式。

参数	是否必填	说明
format	必填	数据文件的格式。创建CSV格式文件的外表时需设置为 csv 。
delimiter		定义CSV数据文件的列分隔符。本示例使用的分隔符为英文逗号（,）。
hdfs_url		HDFS集群中目标数据文件或文件夹的绝对地址，需要以 hdfs:// 开头。 示例： hdfs://172.17.***.***:9000/adb/hdfs_import_test_csv_data/hdfs_import_test_data.csv
partition_column	选填	定义表的分区列，用英文逗号（,）切分各列。定义分区列的方法，请参见创建带分区的HDFS外表。
compress_type		定义数据文件的压缩类型，CSV格式的文件目前仅支持Gzip压缩类型。
skip_header_line_count		定义导入数据时需要在开头跳过的行数。CSV文件第一行为表头，若设置该参数为1，导入数据时可自动跳过第一行的表头。 默认为0，即不跳过。

- **创建HDFS Parquet格式/HDFS ORC格式的外表**

  以Parquet格式为例，创建HDFS外表语句如下：

```
CREATE TABLE IF NOT EXISTS hdfs_import_test_external_table
(
    uid string,
    other string
)
ENGINE='HDFS'
TABLE_PROPERTIES='{
    "format":"parquet",
    "hdfs_url":"hdfs://172.17.***.***:9000/adb/hdfs_import_test_parquet_data/"
}';
```

参数	是否必填	说明
ENGINE='HDFS'	必填	外部表的存储引擎说明。本示例使用的存储引擎为HDFS。
TABLE_PROPERTIES		AnalyticDB MySQL访问HDFS数据的方式。
format		数据文件的格式。 ○ 创建Parquet格式文件的外表时需设置为 parquet 。 ○ 创建ORC格式文件的外表时需设置为 orc 。
hdfs_url		HDFS集群中目标数据文件或文件夹的绝对地址，需要以 hdfs:// 开头。
partition_column	选填	定义表的分区列，用英文逗号（,）切分各列。定义分区列的方法，请参见创建带分区的HDFS外表。

> **说明**
> - 外表创建语句中的列名需与Parquet或ORC文件中该列的名称完全相同（可忽略大小写），且列的顺序需要一致。
> - 创建外表时，可以仅选择Parquet或ORC文件中的部分列作为外表中的列，未被选择的列不会被导入。
> - 如果创建外表创建语句中出现了Parquet或ORC文件中不存在的列，针对该列的查询结果均会返回NULL。

Parquet文件与AnalyticDB MySQL版3.0的数据类型映射关系如下表。

Parquet基本类型	Parquet的logicalType类型	AnalyticDB MySQL版3.0中对应的数据类型
BOOLEAN	无	BOOLEAN
INT32	INT_8	TINYINT
INT32	INT_16	SMALLINT
INT32	无	INT或INTEGER
INT64	无	BIGINT
FLOAT	无	FLOAT
DOUBLE	无	DOUBLE

Parquet基本类型	Parquet的logicalType类型	AnalyticDB MySQL版3.0中对应的数据类型
○ FIXED_LEN_BYTE_ARRAY ○ BINARY ○ INT64 ○ INT32	DECIMAL	DECIMAL
BINARY	UTF-8	○ VARCHAR ○ STRING ○ JSON（如果已知Parquet该列内容为JSON格式）
INT32	DATE	DATE
INT64	TIMESTAMP_MILLIS	TIMESTAMP或DATETIME
INT96	无	TIMESTAMP或DATETIME

ORC文件与AnalyticDB MySQL版3.0的数据类型映射关系如下表。

ORC文件中的数据类型	AnalyticDB MySQL版3.0中对应的数据类型
BOOLEAN	BOOLEAN
BYTE	TINYINT
SHORT	SMALLINT
INT	INT或INTEGER
LONG	BIGINT
DECIMAL	DECIMAL
FLOAT	FLOAT
DOUBLE	DOUBLE
○ BINARY ○ STRING ○ VARCHAR	○ VARCHAR ○ STRING ○ JSON（如果已知ORC该列内容为JSON格式）
TIMESTAMP	TIMESTAMP或DATETIME
DATE	DATE

创建带分区的HDFS外表

HDFS支持对Parquet、CSV和ORC文件格式的数据进行分区，包含分区的数据会在HDFS上形成一个分层目录。在下方示例中，`p1`为第1级分区，`p2`为第2级分区，`p3`为第3级分区：

```
parquet_partition_classic/
├── p1=2020-01-01
│   ├── p2=4
│   │   ├── p3=SHANGHAI
│   │   │   ├── 000000_0
│   │   │   └── 000000_1
│   │   └── p3=SHENZHEN
│   │       └── 000000_0
│   └── p2=6
│       └── p3=SHENZHEN
│           └── 000000_0
├── p1=2020-01-02
│   └── p2=8
│       ├── p3=SHANGHAI
│       │   └── 000000_0
│       └── p3=SHENZHEN
│           └── 000000_0
└── p1=2020-01-03
    └── p2=6
        ├── p2=HANGZHOU
        └── p3=SHENZHEN
            └── 000000_0
```

以Parquet格式为例，创建外表时指定列的建表语句示例如下：

```
CREATE TABLE IF NOT EXISTS hdfs_parquet_partition_table
(
  uid varchar,
  other varchar,
  p1 date,
  p2 int,
  p3 varchar
)
ENGINE='HDFS'
TABLE_PROPERTIES='{
  "hdfs_url":"hdfs://172.17.***.**:9000/adb/parquet_partition_classic/",
  "format":"parquet",  //如需创建CSV或ORC格式外表，仅需将format的取值改为csv或orc。
  "partition_column":"p1, p2, p3"  //针对包含分区的HDFS数据，如需以分区的模式进行查询，那么在导入数据至AnalyticDB MySQL时就需要在外表创建语句中指定分区列partition_column。
}';
```

> **说明**
> - `TABLE_PROPERTIES` 中的partition_column属性必须声明分区列及数据类型（本例中的 `p1 date` 、 `p2 int` 和 `p3 varchar` ）且partition_column属性里必须严格按照第1级, 第2级, 第3级的顺序声明（本例中p1为第1级分区，p2为第2级分区，p3为第3级分区）。
> - 定义分区列的先后顺序需要与partition_column中分区列的顺序保持一致。
> - 支持作为分区列的数据类型为： `BOOLEAN` 、 `TINYINT` 、 `SMALLINT` 、 `INT` 、 `INTEGER` 、 `BIGINT` 、 `FLOAT` 、 `DOUBLE` 、 `DECIMAL` 、 `VARCHAR` 、 `STRING` 、 `DATE` 、 `TIMESTAMP` 。
> - 查询分区列的语法和结果展示与其他数据列没有区别。

2.6.2. 通过外表导出AnalyticDB MySQL数据至HDFS

AnalyticDB MySQL版支持通过外表导入导出数据。本文介绍如何通过AnalyticDB MySQL的外部映射表将AnalyticDB MySQL数据导出至HDFS。

前提条件

- AnalyticDB MySQL版集群需为V3.1.4.4或以上版本。

  > **说明**
  > - 如何查看集群版本，请参见查看版本。
  > - 如需升级版本，请提交工单。

- 已创建HDFS集群，并在HDFS集群中创建了一个新的文件夹（本示例中文件夹名为*hdfs_output_test_csv_data*），用于保存导入的AnalyticDB MySQL版数据。

  > **说明** 使用 `INSERT OVERWRITE` 进行导入时，系统会覆盖目标文件夹下的原始文件。为避免原始文件被覆盖，建议在导出时创建一个新的目标文件夹。

- 已在HDFS集群中为AnalyticDB MySQL集群配置如下服务访问端口：
  - `namenode` ：用于读写文件系统元信息。您可以在fs.defaultFS参数中配置端口号，默认端口号为8020。

    详细配置方式，请参见core-default.xml。
  - `datanode` ：用于读写数据。您可以在dfs.datanode.address参数中配置端口号，默认端口号为50010。

    详细配置方式，请参见hdfs-default.xml。

- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

注意事项

- AnalyticDB MySQL版集群仅支持导出文件格式为CSV和Parquet的数据至HDFS。不支持导出文件格式为ORC的数据。
- AnalyticDB MySQL版集群不支持自定义行级写入的 `INSERT` 语法，如 `INSERT INTO VALUES` 和 `REPLACE INTO VALUES` 。
- 不支持通过分区外表导出单个文件至HDFS。
- 通过分区外表导出数据时，数据文件内不包含分区列的数据，分区列的数据信息以HDFS目录的形式展现。

  例如，已在分区外表中定义了3个普通列和2个分区列。其中一级分区列的列名为 `p1` ，分区列的值为 `1` 。二级分区名称为 `p2` ，分区数值为 `a` ，现需要通过分区外表将数据导出到HDFS的*adb_data/*路径下。

  那么当 `p1=1` 且 `p2=a` 的外表分区导出数据时，数据文件相对路径目录为*adb_data/p1=1/p2=a/*，且外表CSV或Parquet数据文件内不包含 `p1` 和 `p2` 这两列，只包含3列普通列的值。

操作步骤

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 创建源数据库。详细操作步骤，请参见创建数据库。

   本示例中，AnalyticDB MySQL集群的源库名为 `adb_demo` 。
3. 创建源表并插入源数据。

   您可以使用以下语句在源库 `adb_demo` 中创建一张源表 `adb_hdfs_import_source` ，建表语句如下：

   ```
   CREATE TABLE IF NOT EXISTS adb_hdfs_import_source
   (
       uid string,
       other string
   )
   DISTRIBUTED BY HASH(uid);
   ```

   往源表 `adb_hdfs_import_source` 中插入一行测试数据，语句如下：

   ```
   INSERT INTO adb_hdfs_import_source VALUES ("1", "a"), ("2", "b"), ("3", "c");
   ```
4. 创建外部映射表。

   您可以使用以下语法在源库 `adb_demo` 中创建一张外部映射表，用于将AnalyticDB MySQL数据导出至HDFS：

   - 创建普通外部映射表（本文示例中目标表名为 `hdfs_import_external` ），语法如下。

     ```
     CREATE TABLE IF NOT EXISTS hdfs_import_external
     (
         uid string,
         other string
     )
     ENGINE='HDFS'
     TABLE_PROPERTIES='{
         "format":"csv",
         "delimiter":",",
         "hdfs_url":"hdfs://172.17.***.***:9000/adb/hdfs_output_test_csv_data"
     }';
     ```
   - 创建带分区的外部映射表时（本文示例中目标表名为 `hdfs_import_external_par` ），需要同时在创建语句中定义普通列（如 `uid` 和 `other` ）和分区列（如 `p1` 、 `p2` 和 `p3` ），语法如下。

     ```
     CREATE TABLE IF NOT EXISTS hdfs_import_external_par
     (
         uid string,
         other string,
         p1 date,
         p2 int,
         p3 varchar
     )
     ENGINE='HDFS'
     TABLE_PROPERTIES='{
         "format":"csv",
         "delimiter":",",
         "hdfs_url":"hdfs://172.17.***.***:9000/adb/hdfs_output_test_csv_data"
         "partition_column":"p1, p2, p3"
     }';
     ```

   > **说明**
   > - AnalyticDB MySQL版集群仅支持导出文件格式为CSV和Parquet的数据至HDFS。不支持导出文件格式为ORC的数据。
   > - 创建外表的详细语法说明，请参见创建HDFS外表和创建带分区的HDFS外表。
5. 将源AnalyticDB MySQL版集群中的数据导出至目标HDFS中。
   - 如需通过普通外表导出数据，具体语法，请参见附录1：数据导出语法（普通外表）。
   - 如需通过分区外表导出数据，具体语法，请参见附录2：数据导出语法（分区外表）。

后续步骤

导出完成后，您可以通过Hadoop客户端到目标文件夹 `hdfs_output_test_csv_data` 中查看导出的数据文件。您也可以登录AnalyticDB MySQL版集群，在外表中（分区外表与普通外表查询语句一致，本示例以普通外表 `hdfs_import_external` 为例）执行如下语句查询已导出的数据：

```
SELECT * FROM hdfs_import_external LIMIT 100;
```

附录1：数据导出语法（普通外表）

若创建外表时未指定分区列，您可以根据业务需要选择如下几种方式导出数据：

- 方式一：如果您的数据已存在于目标表中，可以通过 `INSERT INTO` 语句将数据导入外表。使用该语句会将源表的数据写入外表对应的HDFS位置，每次写入会产生新的HDFS数据文件。

> **说明** 外表里的列和需要导出的列，必须保持列个数的完整。`INSERT INTO`为增量写入，会额外产生新的文件，不会覆盖旧的历史文件。

语法如下。

```
INSERT INTO <target_table>
SELECT <col_name> FROM <source_table>;
```

示例如下。

```
INSERT INTO hdfs_import_external
SELECT col1, col2, col3 FROM adb_hdfs_import_source;
```

> **说明** `col1, col2, col3`表示外表中的所有列。

- 方式二：HDFS外表不支持定义主键，因此`REPLACE INTO`的写入表现与`INSERT INTO`一致，都会将数据复制到外表。如果目标表内已有数据，执行`REPLACE INTO`语句导入时，已有数据保持不变，新数据会被追加到目标数据文件中。

  > **说明**
  > - 写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列。
  > - `REPLACE INTO`为增量写入，会额外产生新的文件，不会覆盖旧的历史文件。

  语法如下。

  ```
  REPLACE INTO <target_table>
  SELECT <col_name> FROM <source_table>;
  ```

  示例如下。

  ```
  REPLACE INTO hdfs_import_external
  SELECT col1, col2, col3 FROM adb_hdfs_import_source;
  ```

- 方式三：您可以使用`INSERT OVERWRITE INTO`语法向外表中批量插入数据。如果目标外表中已存在数据，每次写入会先删除外表路径下的全部数据文件，再产生新的HDFS数据文件。

  > **注意**
  > - 写入的外表必须保持列个数的完整，不允许指定只写入部分的列。
  > - `INSERT OVERWRITE INTO`为覆盖写入，会覆盖导出目录内已有的历史数据，谨慎使用。

  语法如下。

  ```
  INSERT OVERWRITE INTO <target_table>
  SELECT <col_name> FROM <source_table>;
  ```

  示例如下。

  ```
  INSERT OVERWRITE INTO hdfs_import_external
  SELECT col1, col2, col3 FROM adb_hdfs_import_source;
  ```

- 方式四：异步执行`INSERT OVERWRITE INTO`导出数据，语法如下。

  ```
  SUBMIT job INSERT OVERWRITE INTO <target_table>
  SELECT <col_name> FROM <source_table>;
  ```

  示例如下。

  ```
  SUBMIT JOB INSERT OVERWRITE INTO hdfs_import_external
  SELECT col1, col2, col3 FROM adb_hdfs_import_source;
  ```

  返回结果如下。

  ```
  +---------------------------------------+
  | job_id                                |
  +---------------------------------------+
  | 2020112122202917203100908203303****** |
  +---------------------------------------+
  ```

  您还可以根据上述`job_id`查看异步任务的状态。更多详情，请参见异步提交导入导出任务。

附录2：数据导出语法（分区外表）

分区外表在语法中加入`PARTITION`字段导出数据，您还可以通过指定`PARTITION`字段中的分区列和分区值来确定是否使用静态或者动态分区。

- 方式一：您可以使用`INSERT INTO PARTITION`语法往带分区的外表中批量插入数据。

  > **说明** 写入时，数据将在对应分区追加写入，每次写入会产生新的HDFS数据文件，历史数据不会被覆盖；写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列。

- 全静态分区

  语法如下。

  ```
  INSERT INTO <target_table> PARTITION(par1=val1,par2=val2,...)
  SELECT <col_name> FROM <source_table>;
  ```

  示例如下。

  ```
  INSERT INTO hdfs_import_external_par PARTITION(p1='2021-05-06',p2=1,p3='test')
  SELECT col1, col2, col3, FROM adb_hdfs_import_source;
  ```

- 半静态半动态分区

  > **说明** 静态列必须位于动态列的前面，不允许穿插使用。

  语法如下。

  ```
  INSERT INTO <target_table> PARTITION(par1=val1,par2,...)
  SELECT <col_name> FROM <source_table>;
  ```

  示例如下。

  ```
  INSERT INTO hdfs_import_external_par PARTITION(p1='2021-05-27',p2,p3)
  SELECT col1, col2, col3, FROM adb_hdfs_import_source;
  ```

- 全动态分区（即不需要 `PARTITION` 字段）

  语法如下。

  ```
  INSERT INTO <target_table>
  SELECT <col_name> FROM <source_table>;
  ```

  示例如下。

  ```
  INSERT INTO hdfs_import_external_par
  SELECT col1, col2, col3, FROM adb_hdfs_import_source;
  ```

- 方式二：HDFS外表不支持定义主键，因此 `REPLACE INTO PARTITION` 的写入表现与 `INSERT INTO PARTITION` 一致。

  > **说明** 写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列；`REPLACE INTO PARTITION` 为增量写入，会额外产生新的文件，不会覆盖旧的历史文件。

  语法如下：

  - 全静态分区

    语法如下。

    ```
    REPLACE INTO <target_table> PARTITION(par1=val1,par2=val2,...)
    SELECT <col_name> FROM <source_table>;
    ```

    示例如下。

    ```
    REPLACE INTO hdfs_import_external_par PARTITION(p1='2021-05-06',p2=1,p3='test')
    SELECT col1, col2, col3, FROM adb_hdfs_import_source;
    ```

  - 半静态半动态分区

    > **说明** 静态列必须位于动态列的前面，不允许穿插使用。

    语法如下。

    ```
    REPLACE INTO <target_table> PARTITION(par1=val1,par2,...)
    SELECT <col_name> FROM <source_table>;
    ```

    示例如下。

    ```
    REPLACE INTO hdfs_import_external_par PARTITION(p1='2021-05-06',p2,p3)
    SELECT col1, col2, col3, FROM adb_hdfs_import_source;
    ```

  - 全动态分区（即不需要 `PARTITION` 字段）

    语法如下。

    ```
    REPLACE INTO <target_table>
    SELECT <col_name> FROM <source_table>;
    ```

    示例如下。

    ```
    REPLACE INTO hdfs_import_external_par
    SELECT col1, col2, col3, FROM adb_hdfs_import_source;
    ```

- 方式三：`INSERT OVERWRITE [INTO] PARTITION` 与 `INSERT INTO PARTITION` 使用方法相同，但使用 `INSERT OVERWRITE [INTO] PARTITION` 时，会覆盖掉本次执行中涉及到的目标分区中之前已有的数据文件，对于没有新数据写入的分区，则不会清除其中的数据文件。

  语法如下。

  ```
  INSERT OVERWRITE [INTO] <target_table> PARTITION(par1=val1,par2=val2,...)[IF NOT EXISTS]
  SELECT <col_name> FROM <source_table>;
  ```

  > 注意
  > - 写入的外表必须保持列个数的完整，不允许用户指定只写入一部分的列；`INSERT OVERWRITE [INTO] PARTITION` 为覆盖写入，会覆盖导出目录内已有的历史数据，谨慎使用。
  > - `IF NOT EXISTS`：表示如果外表分区已存在，则不会导出到这个分区。

  示例如下。

  ```
  INSERT OVERWRITE INTO hdfs_import_external_par PARTITION(p1='2021-05-06',p2=1,p3='test') IF NOT EXISTS
  SELECT col1, col2, col3 FROM adb_hdfs_import_source;
  ```

- 方式四：异步执行 `INSERT OVERWRITE INTO` 导出数据，语法如下。

  ```
  SUBMIT JOB INSERT OVERWRITE INTO <target_table>
  SELECT <col_name> FROM <source_table>;
  ```

  示例如下。

  ```
  SUBMIT JOB INSERT OVERWRITE INTO hdfs_import_external_par PARTITION(p1='2021-05-06',p2=1,p3='test') IF NOT EXISTS
  SELECT col1, col2, col3 FROM adb_hdfs_import_source;
  ```

  返回结果如下。

  ```
  +---------------------------------------+
  | job_id                                |
  +---------------------------------------+
  | 2020112122202917203100908203303****** |
  +---------------------------------------+
  ```

  您还可以根据上述 `job_id` 查看异步任务的状态。更多详情，请参见异步提交导入导出任务。

2.7. 大数据

2.7.1. 通过DataWorks同步数据

2.7.1.1. 配置HDFS数据源

本文介绍如何在DataWorks中配置HDFS数据源。

操作步骤

1. 登录DataWorks控制台。
2. 在左侧导航栏单击**工作空间列表**，单击目标工作空间**操作**列的**进入数据集成**。

3. 在**数据集成**页面，单击左侧导航栏**数据源 > 数据源列表**。
4. 单击右上角的**新增数据源**。
5. 在**新增数据源**页面，选择**HDFS**。
6. 在**新增HDFS数据源**页面，按照页面提示进行参数配置。

参数	说明
数据源类型	根据需要选择**连接串模式**或**CDH集群内置模式**。
数据源名称	数据源名称必须包含字母、数字、下划线，但不能以数字和下划线开头。
数据源描述	对数据源进行简单描述，不得超过80个字符。
DefaultFS	nameNode节点地址，格式为 `hdfs://ServerIP:Port` 。

7. 单击**测试连通性**。
8. 测试连通性通过后，单击**完成**。

   测试连通性说明
   - 经典网络ECS上自建的数据源，建议使用数据集成自定义资源组，默认资源组不保证网络可通。
   - 专有网络目前不支持数据源连通性测试，直接单击**完成**。

2.7.1.2. 配置MaxCompute数据源

本文介绍如何在DataWorks中配置MaxCompute数据源。

详情请参见配置MaxCompute数据源。

2.7.1.3. 配置AnalyticDB for MySQL数据源

您可以参照DataWorks同步RDS MySQL数据中的操作步骤完成AnalyticDB for MySQL数据源配置。

详情请参见配置AnalyticDB MySQL 3.0数据源。

2.7.1.4. 配置同步任务中的数据来源和去向

请参见DataWorks同步RDS MySQL中的配置同步任务步骤，将EMR中的数据同步至AnalyticDB for MySQL。

详情请参见使用DataWorks同步RDS MySQL数据。

2.7.2. 通过DLA导入Hadoop数据

2.7.2.1. 功能说明

Marmaray集成DLA提供Hadoop parquet文件到云原生数据仓库AnalyticDB MySQL版数据传输服务。该服务原理是将包含用户传输配置的Marmaray spark jar包运行在DLA提供的执行环境中，实现源端hadoop parquet文件传输到目的端ADB的功能。本文主要对本功能进行详细说明。

数据类型

数据类型映射

类别	Parquet数据类型	ADB3.0数据类型
简单类型	int32	INT
	int64	BIGINT
	FLOAT	FLOAT
	DOUBLE	DOUBLE
逻辑类型	STRING	VARCHAR
	DATE	DATE
	DECIMAL	DECIMAL
	TIMESTAMP	TIMESTAMP

功能列表

字段映射

源端parquet文件对应schema的字段名可映射成目标端ADB对应表的新字段名。例：

parquet文件schema有三个字段，名称a、b、c，对应ADB表中a1、b1、c1三个字段。字段映射 a|a1,b|b1,c|c1。

异常恢复

- **记录级异常恢复**

  单次写入异常温和重试策略。异常后间隔阶梯递增时间重试。第一次失败1s后重试，最多重试12次，最长重试间隔34min。超过最长重试次数后子任务失败，进入子任务级异常恢复流程。阶梯递减重试时间间隔如下所示，单位毫秒。

  ```
  [1000,2000,4000,8000,16000,32000,64000,128000,256000,512000,1024000,2048000]
  ```

- **子任务级异常恢复**

  迁移任务被切分成多个子任务并发执行，单个子任务失败后转移至其他运算节点执行，失败转移次数按需配置。

  默认失败转移次数4。可在任务配置文件conf中增加 `spark.task.maxFailures` 配置参数自定义失败转移次数。

  超过最大转移失败次数后，子任务失败，整个迁移任务缺失子任务数据。

并发流式处理

任务切分基于spark引擎，文件多分区并行读取。

每个分区读取的最大数据量默认128M，可通过参数 `spark.sql.files.maxPartitionBytes` 调整，需在任务配置文件conf中增加该参数。

更多参数配置请参见Spark官方文档。

任务执行可视化

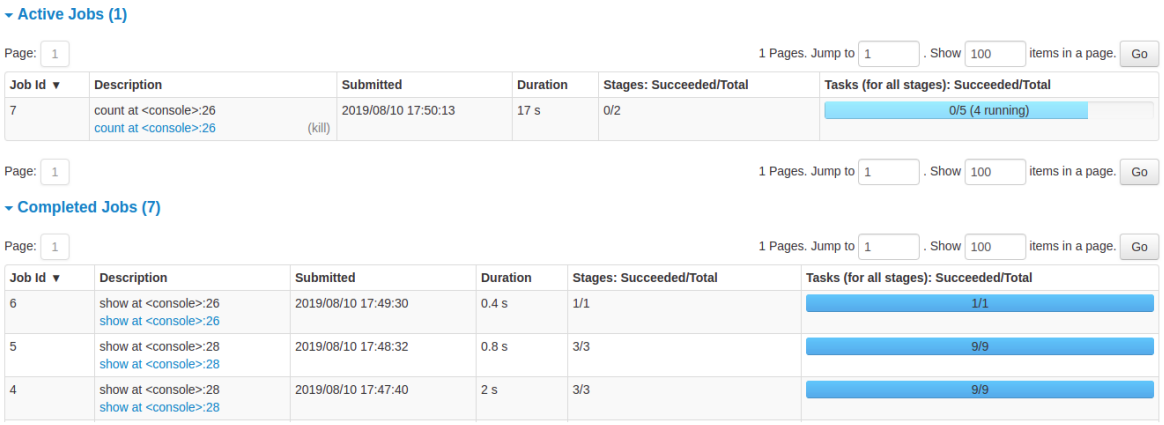
任务执行集成DLA ServerLess spark任务，可在spark web 作业监控页面实时掌控作业运行情况。示例：

- **事件流水记录**

  可查看Executor和job随时间完成情况。

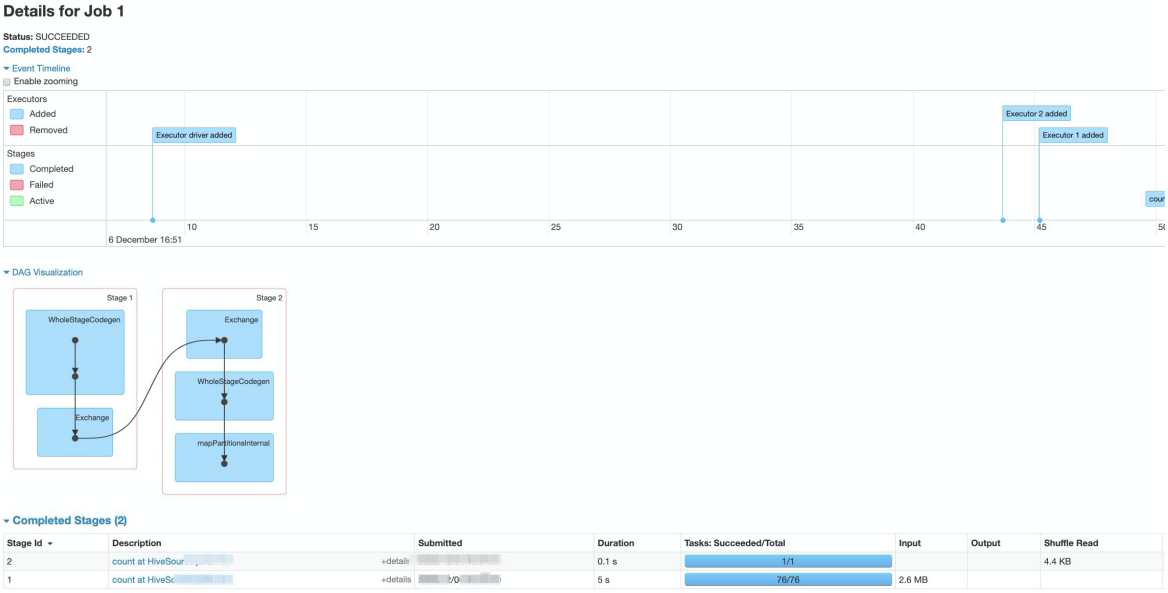
- **任务概况**

  可查看已完成和执行中任务进展。



- **任务详情**

  可查看任务dag详情，完成stage等。



使用限制

当任务失败后，不支持断点续传，需重跑任务完成迁移。

使用方法

使用方法，请参见操作步骤。

2.7.2.2. 操作步骤

云原生数据仓库AnalyticDB MySQL版支持通过DLA导入Hadoop数据。本文介绍相关操作步骤。

前提条件

1.配置网络环境

DLA服务可购买在您的Hadoop或AnalyticDB MySQL的任意VPC内，故配置他们之间连通性首先需要将Hadoop和AnalyticDB MySQL的VPC网络打通，使Hadoop和AnalyticDB MySQL之间能够互相访问。

> **说明** 验证是否打通可用一端ECS访问另一端ECS，如telnet命令等。无论是否互通，都需要将Hadoop交换机IP添加到AnalyticDB MySQL白名单中。

1. **打通Hadoop和AnalyticDB MySQL的VPC网络。**

   详情请参见建立VPC到VPC的连接，或给VPC提工单解决。

2. **查看Hadoop集群交换机和安全组信息。**

   登录E-MapReduce控制台，查看集群基础信息。在网络信息中查看Hadoop集群的交换机和安全组信息。

2.生成YAML配置文件

传输映射关系配置在YAML文件中，下面说明哪些信息需要写入YAML文件及如何获取这些信息。目前支持单个Hadoop Parquet文件到AnalyticDB MySQL单张表的数据传输服务。

YAML文件配置文件模版如下：

```
marmaray:
  adb:
    column_mapping:
    username:
    password:
    port:
    host:
    database:
    tablename:
  Hadoop:
    yarn_queue:
    fs.defaultFS:
  hive:
    data_path:
    job_name:
```

参数说明：

- **目的端AnalyticDB MySQL信息**
  - *column_mapping*：列名映射。

    parquet文件schema对应字段和目标ADB数据库表字段名映射关系，映射关系用 `|` 表示，映射对之间用 `,` 隔开。

    示例：

    Parquet文件schema有三个字段，分别为a、b、c，对应ADB表中a1、b1、c1三个字段。列名映射如下：

    ```
    a|a1,b|b1,c|c1
    ```

  - *username*：数据库用户名。
  - *password*：用户名对应密码。
  - *port*：数据库端口号。
  - *host*：数据库主机地址。
  - *database*：数据库名称。
  - *tablename*：表名称。
- **源端Hadoop集群信息**
  - *yarn_queue*：Hadoop集群yarn队列名。

- *fs.defaultFS*：HDOOP_CONF_DIR。DLA无法自动获取用户Hadoop集群上HDOOP_CONF_DIR中的配置，所以需要您自行配置读取Hadoop相关的参数。分两种情况：
  - 非高可用节点：提供绝对路径名*hdfs://<master_ip>:9000/path/to/file*

    `master_ip` 获取方式：

    a. 登录E-MapReduce控制台，单击导航栏中的**主机列表**。

    b. 找到Master角色对应的IP信息。

  - 高可用节点：需要获取HDOOP_CONF_DIR等配置，详情请参见Hadoop。
- *data_path*：Parquet文件在Hadoop中的路径。
- *job_name*：作业名称。

YAML配置文件样例：

```
marmaray:
  adb:
    column_mapping: a|b,c|d,e|f
    username: xiaoer
    password: password
    port: 3306
    host: am-xxxx.aliyuncs.com
    database: test_db
    tablename: test_table
  Hadoop:
    yarn_queue: default
    fs.defaultFS: hdfs://172.123.XX.XX:9000
  hive:
    data_path: /tmp/parquet
    job_name: testParquetToADB
```

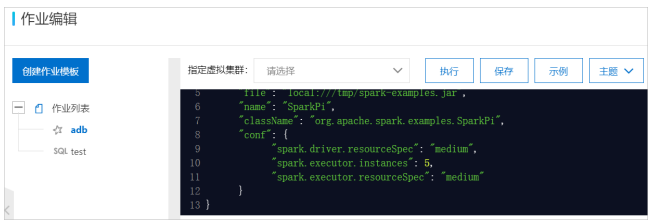
3.上传YAML配置文件

生成好YAML配置文件后，请上传到OSS云存储空间中。

操作步骤

传输服务运行在DLA上，需拉取Marmaray JAR包并加载用户传输映射配置。用户只需要将传输映射配置和Marmaray jar运行模版放在OSS目录下，并将OSS地址填写在DLA运行参数中即可。

1. 提交工单获取Marmaray JAR并上传至OSS云存储空间中。
2. 登录DLA控制台，在左侧导航栏中单击**Severless Spark > 作业管理**进入作业编辑页面。
3. 点击**创建作业模板**按钮。在创建作业对话框中输入**文件名称**、**文件类型**、**父级目录**并选择**作业类型**，单击**确定**。
4. 编辑作业。创建的作业出现在图示作业列表子菜单栏中。



配置介绍

DLA服务运行配置是JSON格式，示例模板如下：

```
{
    "args": [
        "-c",
        "./<yaml配置文件名称>
    ],
    "name": "<任务名称>",
    "className": "com.alibaba.analyticdb.marmaray.ParquetToADBJob",
    "conf": {
        "spark.driver.resourceSpec": "<规格{small、medium、large、xlarge}>",
        "spark.executor.resourceSpec": "<规格{small、medium、large、xlarge}>",
        "spark.executor.instances": "<实例个数>",
        "spark.dla.eni.enable": "<是否开启eni{true}>",
        "spark.dla.eni.vswitch.id": "<Hadoop集群交换机id>",
        "spark.dla.eni.security.group.id": "<Hadoop集群安全组id>",
    },
    "file": "oss://${oss-buck-name}/<marmaray jar路径>",
    "files": "oss://${oss-buck-name}/<yaml配置文件路径>"
}
```

参数说明：

- *name*：为本次任务命名，方便在运行页面查看跟踪。
-

参数名称	使用说明	对应社区Spark参数
spark.driver.resourceSpec	表示Spark driver的资源规格，支持如下取值： ■ `small`：1核4 GB。 ■ `medium`：2核8 GB。 ■ `large`：4核16 GB。 ■ `xlarge`：8核32 GB。	*spark.driver.cores*以及*spark.driver.memory*。
spark.executor.resourceSpec	表示spark executor的资源规格，取值范围同`spark.driver.resourceSpec`。	*spark.executor.cores*以及*spark.executor.memory*。

  - *spark.executor.instances*：执行器（executor）实例个数。
  - *spark.dla.eni.enable*：是否开启ENI，这个参数为true表示启用打通VPC功能。
  - *spark.dla.eni.vswitch.id*：Hadoop集群的交换机ID。
  - *spark.dla.eni.security.group.id*：Hadoop集群的安全组ID。
  - *file*：*marmaray jar*的路径。
  - *files*：YAML配置文件的路径。
5. 运行参数配置完成后，点击**执行**按钮即可执行作业。

2.7.3. 通过外表将MaxCompute数据导入至AnalyticDB MySQL

本文介绍如何将MaxCompute数据导入至AnalyticDB MySQL。

前提条件

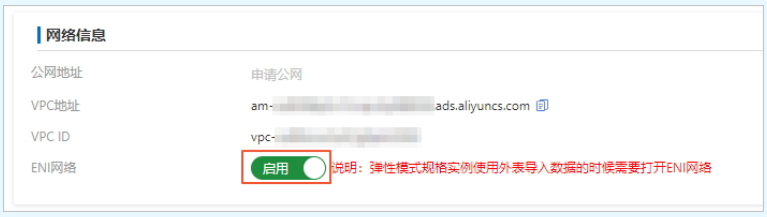
- 根据MaxCompute准备工作和快速入门准备源数据。

  例如通过表操作语句，在MaxCompute中创建以下表。如果您已经有数据源，请跳过该步骤。

```
CREATE TABLE IF NOT EXISTS odps_nopart_import_test
(
uid STRING,
other STRING
 )
LIFECYCLE 3;
```

  在DataWorks中，通过数据导入的方式将文件odps_nopart_import_test.txt中的数据导入odps_nopart_import_test表。
- 根据AnalyticDB for MySQL快速入门，完成创建实例、设置白名单、创建账号和数据库等准备工作。

> **说明** 如果AnalyticDB MySQL集群是弹性模式，请先登录AnalyticDB MySQL控制台，查看集群信息，在网络信息栏启用ENI（Elastic Network Interface，弹性网卡）网络。



操作步骤

1. 连接AnalyticDB for MySQL集群，进入目标数据库。

   本示例将 `odps_nopart_import_test` 表中的数据导入AnalyticDB MySQL的 `test_adb` 数据库中。

2. 通过CREATE TABLE，在 `test_adb` 数据库中创建外部映射表 `odps_nopart_import_test_external_table` 。

```
CREATE TABLE IF NOT EXISTS odps_nopart_import_test_external_table
(
 uid string,
 other string
)
 ENGINE='ODPS'
 TABLE_PROPERTIES='{
 "endpoint":"http://service.cn.maxcompute.aliyun-inc.com/api",
 "accessid":"L*******FsE",
 "accesskey":"CcwF********iWjv",
 "project_name":"odps_project1",
 "table_name":"odps_nopart_import_test"
 }'
```

参数	说明
ENGINE='ODPS'	表示该表是外部表，使用的存储引擎是MaxCompute。
TABLE_PROPERTIES	用于告知AnalyticDB MySQL如何访问MaxCompute中的数据。
endpoint	MaxCompute的**EndPoint（域名节点）**。 > **说明** 目前仅支持AnalyticDB MySQL通过MaxCompute的VPC网络Endpoint访问MaxCompute。 如何查看MaxCompute Endpoint，请参见Endpoint。
accessid	您访问MaxCompute源表时所持有的Access Key Secret。 如何获取您的 `accessid` 和 `accesskey` ，请参见获取账号的AK信息。
accesskey	您访问MaxCompute源表时所持有的AccessKey ID。
project_name	MaxCompute中的工作空间名称。
table_name	MaxCompute中的数据源表名。

3. 通过CREATE TABLE，在 `test_adb` 数据库中创建目标表 `adb_nopart_import_test` 存储从MaxCompute中导入的数据。

```
CREATE TABLE IF NOT EXISTS adb_nopart_import_test
(
uid string,
other string
)
DISTRIBUTE BY HASH(uid);
```

4. 执行INSERT语句将MaxCompute数据导入AnalyticDB MySQL。
   - 执行 `INSERT INTO` 导入数据。

```
insert into adb_nopart_import_test
select * from odps_nopart_import_test_external_table
```

```
select * from adb_nopart_import_test
+---------------+-----------------+
| uid           | other           |
+---------------+-----------------+
| 4             | other4          |
| 6             | other6          |
| 5             | other5          |
| 2             | other2          |
| 1             | other1          |
| 3             | other3          |
| 7             | other7          |
```

- 异步执行 `INSERT OVERWRITE INTO` 导入数据。

```
submit job insert overwrite into adb_nopart_import_test
select * from odps_nopart_import_test_external_table
+----------------------------------------+
| job_id                                 |
+----------------------------------------+
| 20201121222029172031009082033030000321 |
```

关于异步提交任务详情请参见异步提交导入导出任务。

2.7.4. 通过外表导出AnalyticDB MySQL数据至MaxCompute

本文介绍如何通过INSERT外表方式将AnalyticDB MySQL版集群的SQL数据导出到MaxCompute（原名ODPS）分区表。

前提条件

- 根据MaxCompute准备工作和快速入门准备目标数据表。

  例如通过表操作语句，在MaxCompute中创建以下表。如果您已经有目标数据表，请跳过该步骤。

  > **说明** 导出数据到MaxCompute分区表时，需要明确指定待写入的分区，不支持同时导出到多个分区，您可以通过执行多个SQL的方式实现将数据导出到多个MaxCompute分区。MaxCompute最高支持6级分区，其他多级分区的操作步骤类似。

  - 一级分区表

```
CREATE TABLE IF NOT EXISTS odps_table
(
uid STRING
 )
PARTITIONED BY (ds STRING)
LIFECYCLE 3;
```

  - 二级分区表

```
CREATE TABLE IF NOT EXISTS odps_table
(
uid STRING
 )
PARTITIONED BY (ds STRING,other STRING)
LIFECYCLE 3;
```

- 根据AnalyticDB MySQL快速入门，完成创建实例、设置白名单、创建账号和数据库等准备工作。
- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

导出到MaxCompute一级分区表

本示例将AnalyticDB MySQL版的 `adb_table` 表中的数据导出到MaxCompute的 `odps_table` 一级分区表中。

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 新建一张外表映射到MaxCompute的目标数据表。

```
CREATE TABLE odps_external_table
(
uid string,
ds string
)
ENGINE='ODPS'
TABLE_PROPERTIES='{
"endpoint":"http://service.odps.aliyun-inc.com/api",
"accessid":"xxx",
"accesskey":"xxx",
"project_name":"xxx",
"table_name":"odps_table",
"partition_column":"ds"
}'
```

参数	说明
ENGINE='ODPS'	表示该表是外部表，使用的存储引擎是MaxCompute。
TABLE_PROPERTIES	用于告知AnalyticDB MySQL版如何访问MaxCompute，并向其写入数据。
endpoint	MaxCompute的 `EndPoint（域名节点）` 。 **说明** 目前仅支持AnalyticDB MySQL版通过MaxCompute的VPC网络Endpoint访问MaxCompute。 如何查看MaxCompute的EndPoint，请参见Endpoint。
accessid	您在访问MaxCompute目标数据表时所持有的AccessKey ID。
accesskey	您在访问MaxCompute目标数据表时所持有的AccessKey Secret。
project_name	MaxCompute中目标数据表所在的工作空间名。
table_name	MaxCompute中目标数据表的表名。
partition_column	分区字段。

3. 将数据从AnalyticDB MySQL版外表导出到MaxCompute的数据表一级分区表。

说明 `adb_table_column` 不包含partition里面的字段。

```
insert [overwrite] into odps_external_table partition(ds='20200401')
select [adb_table_column, ...] from adb_table [where ...]
```

导出到MaxCompute二级分区表

本示例将AnalyticDB MySQL版的 `adb_table` 表中的数据导出到MaxCompute的 `odps_table` 二级分区表中。

1. 连接目标AnalyticDB MySQL集群。详细操作步骤，请参见连接集群。
2. 新建一张外表映射到MaxCompute的数据表。

```
CREATE TABLE odps_external_table
(
uid string,
ds string,
other string)
ENGINE='ODPS'
TABLE_PROPERTIES='{
"endpoint":"http://service.odps.aliyun-inc.com/api",
"accessid":"xxx",
"accesskey":"xxx",
"project_name":"xxx",
"table_name":"odps_table",
"partition_column":"ds,other"
}'
```

3. 将数据从AnalyticDB MySQL版外表导出到MaxCompute数据表二级分区表。

说明 `adb_table_column` 不包含partition里面的字段。

```
insert [overwrite] into odps_external_table partition(ds='20200401',other='hangzhou')
select [adb_table_column, ...] from adb_table [where ...]
```

2.7.5. 通过开源Flink导入数据至AnalyticDB MySQL

本文介绍如何将开源Flink中的数据写入AnalyticDB MySQL版集群。

前提条件

- 下载Flink驱动，并将其部署到Flink所有节点的*${flink部署目录}/lib*目录下。您可以根据Flink版本下载对应的驱动：
  - Flink 1.11版本：flink-connector-jdbc_2.11-1.11.0.jar
  - Flink 1.12版本：flink-connector-jdbc_2.11-1.12.0.jar
  - Flink 1.13版本：flink-connector-jdbc_2.11-1.13.0.jar

  如需其他版本的驱动，请前往JDBC SQL Connector 页面下载。
- 下载MySQL驱动，并将其部署到Flink所有节点的*${flink部署目录}/lib*目录下。

  > **说明** MySQL驱动版本需为5.1.40或以上，请前往MySQL驱动下载页面下载。
- 部署所有的JAR包后请重启Flink集群。启动方式，请参见Start a Cluster。
- 已在目标AnalyticDB MySQL版集群中创建数据库和数据表，用于保存需要写入的数据。数据库和数据表的创建方法，请参见CREATE DATABASE和CREATE TABLE。

  > **说明**
  > - 本文示例中创建的数据库名称为 `tpch` ，建库语句如下：
  >
  > ```
  > CREATE DATABASE IF NOT EXISTS tpch;
  > ```
  >
  > - 本文示例中创建的数据表名为 `person` ，建表语句如下：
  >
  > ```
  > CREATE TABLE IF NOT EXISTS person(user_id string, user_name string, age int);
  > ```

- 如果您的AnalyticDB MySQL集群是弹性模式，您需要在**集群信息**页面的**网络信息**区域，打开启用**ENI网络**的开关。

注意事项

- 本文仅介绍通过Flink SQL创建表并写入数据至AnalyticDB MySQL版的方法。通过Flink JDBC API写入数据的方法，请参见JDBC Connector。
- 本文介绍的方法仅适用于Flink1.11及以上版本。若您需要将其他版本的Flink数据写入AnalyticDB MySQL版集群，那么：
  - 针对Flink1.10和Flink1.09版本，数据写入方法，请参见Flink 1.10 Documentation: Connect to External Systems。
  - 针对Flink1.08及以下版本，数据写入方法，请参见Flink 1.08 Documentation: Connect to External Systems。

流程介绍

> **说明** 本文示例以CSV格式的文件作为输入源介绍数据写入流程。

步骤	说明
步骤一：数据准备	创建一个新的CSV文件并在文件中写入源数据，然后将新文件部署至Flink所有节点的*/root*下。
步骤二：数据写入	通过SQL语句在Flink中创建源表和结果表，并通过源表和结果表将数据写入AnalyticDB MySQL中。
步骤三：数据验证	登录AnalyticDB MySQL目标数据库，来查看并验证源数据是否成功导入。

步骤一：数据准备

1. 在其中一个Flink节点的root目录下，执行 `vim /root/data.csv` 命令来创建一个名为*data.csv*的CSV文件。

   文件中包含的数据如下（您可以多复制几行相同的数据来增加写入的数据量）：

   ```
   0,json00,20
   1,json01,21
   2,json02,22
   3,json03,23
   4,json04,24
   5,json05,25
   6,json06,26
   7,json07,27
   8,json08,28
   9,json09,29
   ```

2. 文件创建完成后，将其部署至Flink其他节点的*/root*目录下。

步骤二：数据写入

1. 启动并运行Flink SQL程序。详细操作步骤，请参见Starting the SQL Client CLI。
2. 创建一张名为 `csv_person` 的源表，语句如下：

```
CREATE TABLE if not exists csv_person (
  `user_id` STRING,
  `user_name` STRING,
  `age` INT
) WITH (
  'connector' = 'filesystem',
  'path' = 'file:///root/data.csv',
  'format' = 'csv',
  'csv.ignore-parse-errors' = 'true',
  'csv.allow-comments' = 'true'
);
```

> **说明**
> - 源表中的列名和数据类型需与AnalyticDB MySQL版中目标表的列名和数据类型保持一致。
> - 建表语句中填写的 `path` 是*data.csv*的本地路径（Flink各个节点的路径均需一致）。如果您的*data.csv*文件不在本地，请根据实际情况填写正确的路径。
>
> 关于建表语句中的其他参数说明，请参见FileSystem SQL Connector。

3. 创建一张名为 `mysql_person` 的结果表，语句如下：

```
CREATE TABLE mysql_person (
  user_id String,
  user_name String,
  age INT
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://<endpoint:port>/<db_name>?useServerPrepStmts=false&rewriteBatchedStatements=true',
  'table-name' = '<table_name>',
  'username' = '<username>',
  'password' = '<password>',
  'sink.buffer-flush.max-rows' = '10',
  'sink.buffer-flush.interval' = '1s'
  );
```

> **说明**
> - 结果表中的列名和数据类型需与AnalyticDB MySQL版中目标表的列名和数据类型保持一致。
> - 下表仅列举了连接AnalyticDB MySQL版集群时的必填配置项，关于选填配置项的信息，请参见Connector Options。

必填配置项	说明
`connector`	指定Flink使用的连接器类型，选择 `jdbc` 。
`url`	AnalyticDB MySQL版集群的JDBC URL。 格式： `jdbc:mysql://<endpoint:port>/<db_name>?useServerPrepStmts=false&rewriteBatchedStatements=true'` ，其中： • `endpoint` ：目标AnalyticDB MySQL版集群的连接地址。 **说明** 如果需要使用公网地址连接集群，您需要先申请公网地址，申请方法，请参见申请公网地址。 • `db_name` ：AnalyticDB MySQL版中的目标数据库名。 • `useServerPrepStmts=false&rewriteBatchedStatements=true` ：批量写入数据至AnalyticDB MySQL版的必填配置，用于提高写入性能，以及降低对AnalyticDB MySQL版集群的压力。 示例： `jdbc:mysql://am-**********.ads.aliyuncs.com:3306/tpch?useServerPrepStmts=false&rewriteBatchedStatements=true` 。
`table-name`	AnalyticDB MySQL版中的目标表名，用于存储写入的数据。本文示例中目标表名为 `person` 。
`username`	AnalyticDB MySQL版中具有写入权限的数据库账号名。 **说明** • 您可以通过SHOW GRANTS查看当前账号所拥有的权限。 • 您可以通过GRANT语句为目标账号授予权限。
`password`	AnalyticDB MySQL版中具有写入权限的数据库账号密码。

必填配置项	说明	
`sink.buffer-flush.max-rows`	从Flink写入数据至AnalyticDB MySQL版时，一次批量写入的最大行数。Flink会接收实时数据，当接收到的数据行数达到最大写入行数后，再将数据批量写入AnalyticDB MySQL版集群。可选取值如下： ○ *0*：最大行数为0时，批量写入数据功能仅考虑 `sink.buffer-flush.interval` 配置，即只要满足最大间隔时间就会开始批量写入。 ○ 具体的行数，例如*1000*、*2000*等。 **说明** 不建议将该参数设置为*0*。取值为*0*不仅会导致写入性能变差，也会导致AnalyticDB MySQL版集群执行并发查询时的压力变大。	当 `sink.buffer-flush.max-rows` 和 `sink.buffer-flush.interval` 配置均不为*0*时，批量写入功能生效规则如下： ○ 若Flink接收到的数据量已达到 `sink.buffer-flush.max-rows` 所设的值，但最大时间间隔还未到达 `sink.buffer-flush.interval` 所设的值，那么Flink无需等待间隔期满，即可直接触发批量写入数据至AnalyticDB MySQL版。 ○ 若Flink接收到的数据量未达到 `sink.buffer-flush.max-rows` 所设的值，但间隔时间已达到 `sink.buffer-flush.interval` 所设的值，那么无论Flink接收了多少数据量，都直接触发批量写入数据至AnalyticDB MySQL版。
`sink.buffer-flush.interval`	Flink批量写入数据至AnalyticDB MySQL版的最大间隔时间，即执行下一次批量写入数据前的最大等待时间，可选取值如下： ○ *0*：时间间隔为0时，批量写入数据功能仅考虑 `sink.buffer-flush.max-rows` 配置，即只要Flink接收到的数据行数达到最大写入行数后就会开始批量写入。 ○ 具体的时间间隔，例如*1d*、*1h*、*1min*、*1s*、*1ms*等。 **说明** 不建议将该参数设置为*0*，避免在业务低谷期产生源数据较少的场景下，影响数据导入的及时性。	

4. 通过源表和结果表将数据写入AnalyticDB MySQL版集群中，语句如下：

```
INSERT INTO mysql_person SELECT user_id, user_name, age FROM csv_person;
```

步骤三：数据验证

导入完成后，您可以登录AnalyticDB MySQL集群的目标库 `tpch`，执行如下语句查看并验证源数据是否成功导入至目标表 `person` 中：

```
SELECT * FROM person;
```

2.8. Kafka

2.8.1. 概述

Logstash是开源的服务器端数据处理管道，能够同时从多个数据源采集数据，然后对数据进行转换，并将数据写入指定的存储中。AnalyticDB for MySQL完全兼容MySQL，您可以将Logstash Input插件支持的任一数据源中的数据写入AnalyticDB for MySQL。本文介绍如何通过logstash-input-kafka插件将Kafka数据写入到AnalyticDB for MySQL。

Logstash组件介绍

- **输入-采集各种样式、大小和来源的数据**

  在实际业务中，数据往往以各种各样的形式分散或集中地存储在多个系统中，Logstash支持多种数据输入方式，可以在同一时间从多种数据源采集数据。Logstash能够以连续的流式传输方式轻松地从用户的日志、指标、Web应用、数据存储以及AWS服务采集数据。

- **过滤-实时解析和转换数据**

  数据从源传输到目标存储的过程中，Logstash过滤器能够解析各个事件，识别已命名的字段来构建结构，并将它们转换成通用格式，从而更轻松、快速地分析和实现商业价值。

  ○ 使用Grok从非结构化数据中派生出结构化数据。
  ○ 从IP地址破译出地理坐标。
  ○ 将PII数据匿名化，完全排除敏感字段。
  ○ 简化整体处理，不受数据源、格式或架构的影响

- **输出-导出数据**

  除了AnalyticDB for MySQL以外，Logstash提供多种数据输出方向，灵活解锁众多下游用例。

示例

Kafka是一个高吞吐量的分布式发布、订阅日志服务，具有高可用、高性能、分布式、高扩展、持久性等特点。目前Kafka已经被各大公司广泛使用，同时logstash也可以快速接入业务中，免去重复建设的麻烦。

详情请参见使用Logstash将Kafka数据写入AnalyticDB for MySQL。

2.8.2. 使用Logstash将Kafka数据写入AnalyticDB for MySQL

本文介绍如何使用Logstash将Kafka数据写入AnalyticDB for MySQL。

操作步骤

1. 执行以下命令安装和更新插件。

```
$bin/plugin install
$bin/plugin update
```

Logstash从1.5版本开始集成Kafka，Logstash 1.5及以上版本中所有插件的目录和命名都发生了改变，插件发布地址为Logstash-plugins。

2. 配置插件。
   - **Input配置示例**

     以下配置可以实现对Kafka读取端（consumer）的基本使用。

```
input {
    kafka {
        zk_connect => "localhost:2181"
        group_id => "Logstash"
        topic_id => "test"
        codec => plain
        reset_beginning => false # boolean (optional), default: false
        consumer_threads => 5  # number (optional), default: 1
        decorate_events => true # boolean (optional), default: false
    }
}
```

     参数说明：

     - `group_id`：消费者分组，可以通过组ID来指定，不同组之间的消费互不影响，相互隔离。
     - `topic_id`：指定消费话题（Topic），也可以理解为先订阅某个话题，然后消费。
     - `reset_beginning`：指定Logstash启动后从哪个位置开始读取数据，默认是结束位置，即Logstash进程会从上次读取结束时的偏移量开始继续读取数据；如果之前没有消费过，则从头读取数据。

       如果您要导入原数据，需将`reset_beginning`值改为`true`，Logstash进程将从头开始读取数据，作用类似于cat，但是Logstash读到最后一行时不会终止，而是变成`tail -F`，继续监听相应数据。
     - `decorate_events`：指定输出消息时会输出自身信息，包括消费消息的大小、Topic来源以及consumer的group信息。
     - `rebalance_max_retries`：当有新的consumer（Logstash）加入到同一个group时，将会Reblance，此后将会有Partitions的消费端迁移到新的consumer上。如果一个consumer获得了某个Partition的消费权限，那么它将会向Zookeeper注册Partition Owner registry节点信息，但是有可能此时旧的consumer尚没有释放此节点，此值用于控制注册节点的重试次数。
     - `consumer_timeout_ms`：在指定时间内没有消息到达将抛出异常，该参数一般无需修改。

     更多Input参数配置请参见Input。

     > **说明** 如果需要多个Logstash端协同消费同一个Topic，需要先把相应的Topic分多个Partitions（区），此时多个消费者消费将无法保证消息的消费顺序性，然后把两个或多个Logstash消费端配置成相同的`group_id`和`topic_id`。

   - **Output配置示例**

```
output {
    jdbc {
        driver_class => "com.mysql.jdbc.Driver"
        connection_string => "jdbc:mysql://HOSTNAME/DATABASE?user=USER&password=PASSWORD"
        statement => [ "INSERT INTO log (host, timestamp, message) VALUES(?, ?, ?)", "host", "@timestamp", "message" ]
    }
}
```

     参数说明：

     - `connection_string`：AnalyticDB for MySQL的连接地址。
     - `statement`：INSERT SQL的声明数组。

     更多Output参数配置请参见Output。

3. 在Logstash安装目录中执行`bin/Logstash -f config/xxxx.conf`命令启动任务，将Kafka数据写入AnalyticDB for MySQL。

2.9. 日志数据

2.9.1. 使用Logstash实时采集日志数据

Logstash是一个开源的服务器端数据处理管道，起初用于将日志类数据写入ES中。随着开源社区的不断发展，Logstash可以同时从多个数据源获取数据，并对其进行转换，然后将其发送到您需要的“存储端”。

以日志数据为例，由于AnalyticDB for MySQL支持原生JDBC方式访问，您可以通过开源logstash output插件logstash-output-jdbc将日志数据导入AnalyticDB for MySQL中进行进一步分析。但经过测试发现，在日志量非常大的情况下，通过JDBC方式将数据写入AnalyticDB for MySQL的性能较低，并且非常消耗CPU的资源（JDBC是单条记录写入）。为此，AnalyticDB for MySQL优化了一个基于JDBC的Logstash output plugin插件——logstash-ouput-analyticdb，专门用于以聚合方式向AnalyticDB for MySQL中写入日志数据。

通过logstash-output-analyticdb将数据写入AnalyticDB for MySQL时的性能，相较于logstash-output-jdbc有5倍提升，并且对CPU的消耗也明显降低。

安装

Logstash的安装流程请参见Installing Logstash，以下介绍如何安装logstash-output-analyticdb。

1. 进入logstash根目录：`cd logstash`。

2. 安装logstash-output-analyticdb：`bin/logstash-plugin install logstash-output-analyticdb`。
3. 在logstash目录下创建`vendor/jar/jdbc`目录：`mkdir -p vendor/jar/jdbc`。
4. 将jdbc jar拷贝到`vendor/jar/jdbc`中：`cd vendor/jar/jdbc; wget http://central.maven.org/maven2/mysql/mysql-connector-java/5.1.36/mysql-connector-java-5.1.36.jar`。

至此，已成功安装logstash-output-analyticdb。

使用方式

在config目录下创建一个`logstash-analyticdb.conf`（名字可以自定义）配置文件，`logstash-analyticdb.conf`文件的内容如下所示。

```
input
{
    stdin { }
}
output {
    analyticdb {
        driver_class => "com.mysql.jdbc.Driver"
        connection_string => "jdbc:mysql://HOSTNAME:PORT/DATABASE?user=USER&password=PASSWORD"
        statement => [ "INSERT INTO log (host, timestamp, message) VALUES(?, ?, ?)", "host", "@timestamp", "message" ]
        commit_size => 4194304
    }
}
```

- `connection_string`：连接AnalyticDB for MySQL的JDBC URL。
- `statement`：INSERT SQL的声明数组。

更多参数配置：

- `max_flush_exceptions`：当写入数据出现异常时，设置最大重试次数，默认值100。
- `skip_exception`：设置是否跳过异常，默认为FALSE，表示出现异常时将重试直到到达最大重试次数`max_flush_exceptions`，如果仍然失败，则同步程序抛异常终止。设置为TRUE时，如果达到重试次数后仍是失败，则跳过异常，将异常写入日志。
- `flush_size`：一次最多攒批数量，和`commit_size`参数搭配使用。
- `commit_size`：一次最多攒批数据量大小，和`flush_size`参数搭配使用，达到限定值即提交写入任务。

上述配置文件只是一个示例，您需要根据实际业务配置`logstash-analyticdb.conf`文件。与AnalyticDB for MySQL相关的其他配置请参见README。更多logstash配置和使用规则，请参见logstash文档。

至此，配置任务已全部完成，接下来将启动任务。

启动任务

在logstash的安装目录执行`bin/logstash -f config/logstash-analyticdb.conf`启动任务。

注意事项

建议您通过以下命令将logstash升级至最新版本后，再进行数据写入。

```
bin/logstash-plugin update logstash-output-analyticdb
```

2.9.2. 将日志服务SLS数据投递到ADB

日志作为一种特殊的数据，对处理历史数据、诊断问题等非常重要。对数据分析人员、开发人员或者运维人员而言，日志都是其工作过程中必不可缺的数据来源。阿里云从用户角度出发，支持通过日志服务将Nginx访问日志、Log4j日志、Apache日志以及结构化文本等日志实时同步至ADB，从而使用ADB进行实时日志分析。

前提条件

- 首次使用日志服务时，您需要通过阿里云账号登录日志服务 LOG产品详情页，单击**立即购买**，进入购买页面，然后单击**立即开通**即可购买日志服务，系统自动跳转到日志服务控制台。

  > **说明** 如果您之前已经开通过日志服务，可直接从创建Project开始。

- 在AnalyticDB for MySQL中完成以下准备工作：
  i. 创建集群。
  ii. 创建数据库账号。
  iii. 创建数据库。
  iv. 如果您需要通过外网地址连接AnalyticDB for MySQL集群，请申请外网地址。
  v. 通过CREATE TABLE创建表，存储投递过来的日志数据。

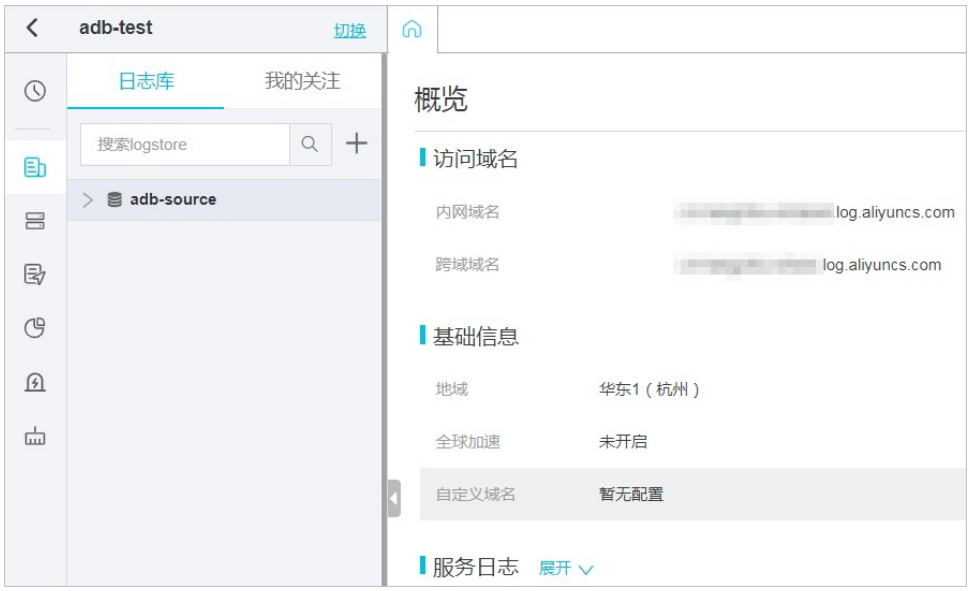
创建投递任务

您可以通过日志服务数据处理模块中的导出功能，将Logstore中采集到的日志投递到AnalyticDB for MySQL。

1. 登录日志服务控制台。
2. 创建Project和Logstore（日志库）。

   本示例创建adb-test Project和adb-source Logstore，详情请参见创建Project和创建Logstore。

> **说明** 目前仅支持同地域投递日志，因此日志服务中的Project所属地域应与AnalyticDB for MySQL所属地域相同。

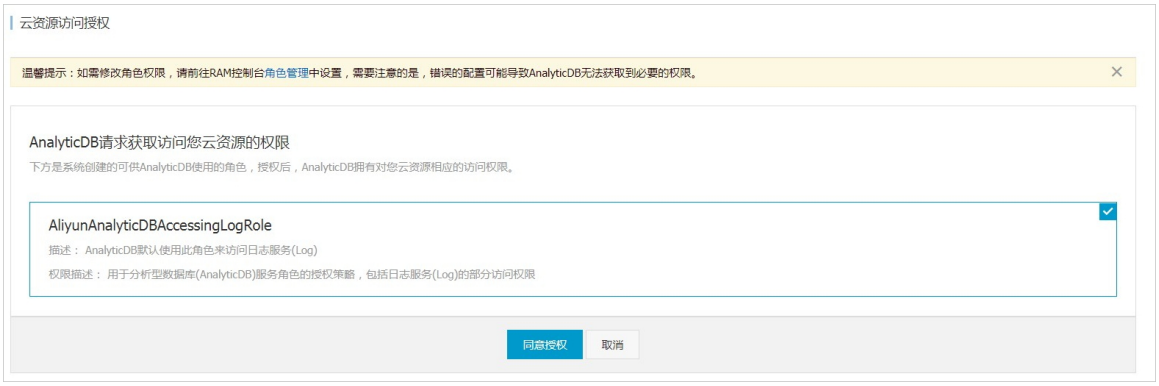


3. 在**日志库**列表中，单击目标Logstore名称前的**> > 数据处理 > 导出 > AnalyticDB**。

   如果您是首次将日志数据投递到AnalyticDB for MySQL，需要为AnalyticDB for MySQL授权，允许AnalyticDB for MySQL访问日志服务。

   i. 单击AnalyticDB后的**+**，系统自动提示您进行授权操作，单击**权限**。

   ii. 在云资源访问授权页面，单击**同意授权**，将角色**AliyunAnalyticDBAccessingLogRole**授予AnalyticDB for MySQL。



4. 完成授权后，单击AnalyticDB后的**+**，您可以选择**直接投递**日志数据或者**前往数据加工**对数据进行处理后再投递。

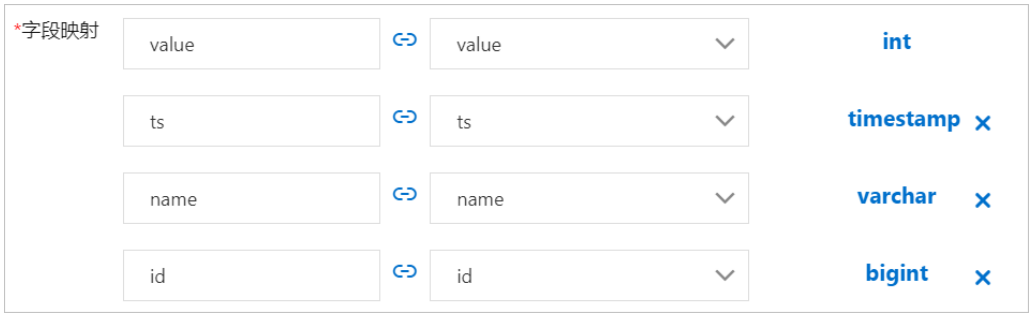
> **说明** 本示例选择直接投递数据。

5. 在**LogHub —— 数据投递**页面，按照页面提示进行参数配置。

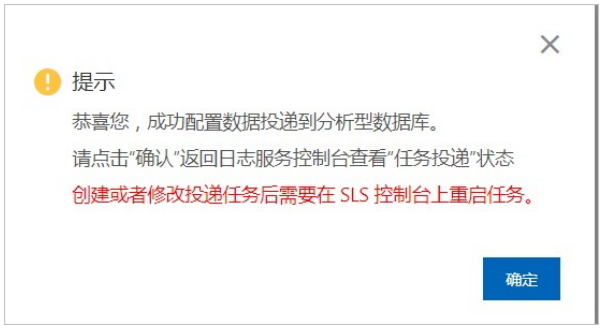
   数据投递参数说明

参数	说明
投递名称	为投递任务取一个名字，便于后续管理。
投递描述	为投递任务填写有意义的描述，便于后续管理。
集群版本	设置AnalyticDB for MySQL集群版本，选择**3.0**。 如何将日志数据投递到AnalyticDB for MySQL 2.0，请参见通过日志服务同步ECS日志数据到AnalyticDB for MySQL。 。
集群名称	设置目标AnalyticDB for MySQL集群。
数据库名称	填写目标AnalyticDB for MySQL中的数据库名。
表名	填写目标AnalyticDB for MySQL中的表名。

参数	说明
账号名称	目标AnalyticDB for MySQL中创建的数据库账号： ○ 高权限账号 ○ 普通账号
账号密码	填写账号名称对应的密码。
字段映射	系统自动提取日志服务中最近10分钟的日志字段，同时自动映射对应的AnalyticDB for MySQL字段。
投递时间	设置投递开始时间。 当数据写入日志服务后，日志服务可以实时将数据投递到AnalyticDB for MySQL。
是否过滤脏数据	设置是否过滤脏数据。 ○ **否**：遇到脏数据时，投递任务自动中断。 ○ **是**：遇到脏数据时，过滤掉脏数据。 脏数据包括数据类型转化失败和非空字段为空等。



6. 完成上述参数配置后，单击**确定**，日志服务将在您设置的投递时间将数据实时投递到AnalyticDB for MySQL表中。

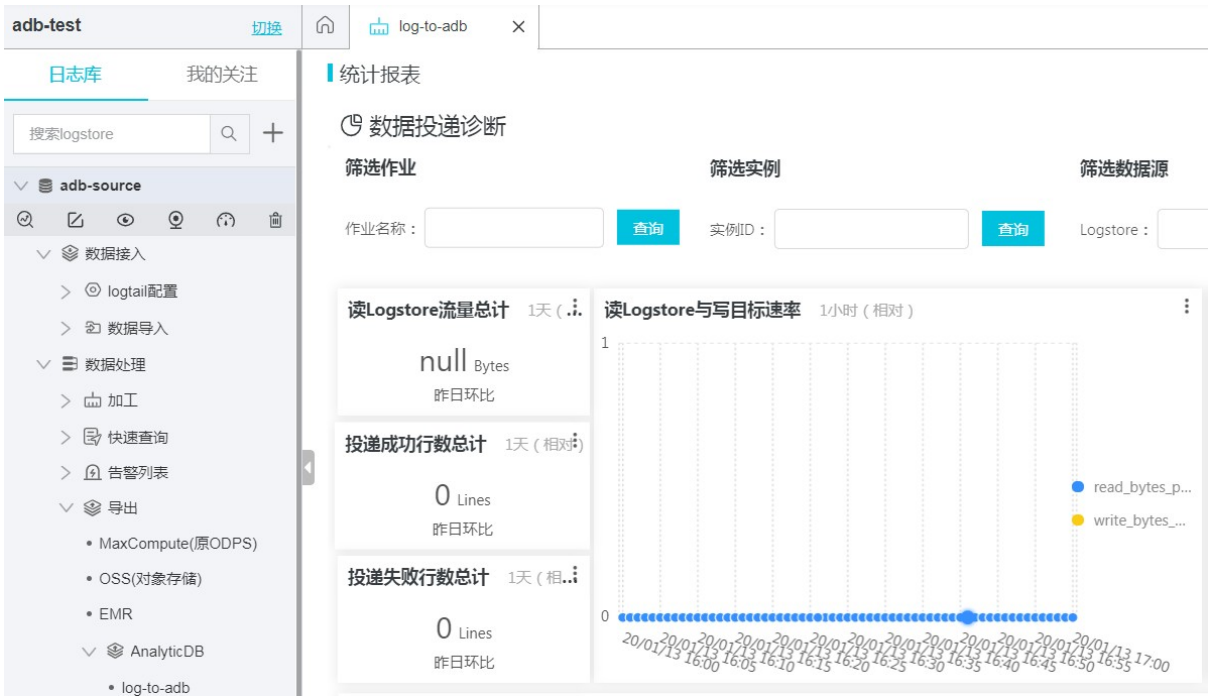


查看日志数据

将日志投递到AnalyticDB for MySQL后，您就可以通过SELECT对日志数据进行自由灵活地分析。

管理投递任务

- 监控投递任务：单击投递名称，可以查看投递任务的执行情况，包括投递成功数据量、投递失败数据量以及投递延迟等。



- 修改投递任务：单击**修改投递配置**修改投递任务。

- 重启投递任务：单击**停止**，待投递任务停止后，单击**启动**重启投递任务。

2.10. 本地数据

2.10.1. 使用LOAD DATA导入本地数据

本文介绍如何通过LOAD DATA将本地数据导入AnalyticDB for MySQL。

语法

```
LOAD DATA LOCAL '通过该命令导入本地文件'
    INFILE 'file_name' '本地文件的路径，包含文件地址和文件名。'
    [REPLACE | IGNORE] '导入数据遇到主键重复时用当前数据覆盖已有数据或者自动忽略失败行'
    INTO TABLE table_name 'AnalyticDB for MySQL中的目标表名'
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string'] '定义每行数据的列分隔符，默认为\t，与MySQL一致。'
        [[OPTIONALLY] ENCLOSED BY 'char'] '定义每列数据的enclosed符号'
    ]
    [LINES
        [TERMINATED BY 'string'] '定义行分隔符，默认为\n。'
    ]
    [IGNORE number {LINES | ROWS}] '设置导入数据时忽略开始的某几行'
    [(column_name_or_user_var
    [, column_name_or_user_var] ...)] '设置导入的列，如果不设置，默认按照列的顺序来导入数据'
```

参数

- `LOAD DATA LOCAL INFILE`：表示要进行本地文件导入操作。
- `file_name`：要导入AnalyticDB for MySQL的本地文件的路径，包含文件地址和文件名。

  > **说明** 如果`file_name`使用相对路径，则是相对于客户端程序启动时的路径。

- `table_name`：AnalyticDB for MySQL中的目标表名。

  > **说明** 表名前无需携带数据库名。

- `REPLACE`：导入数据时，遇到主键重复则强制用当前数据覆盖已有数据。
- `IGNORE`：导入数据时，遇到主键重复或者数据问题时则自动忽略失败行，部分行导入失败但不影响其他行导入。
- `[FIELDS] TERMINATED BY 'string'`：定义每行数据的列分隔符，默认为`\t`，与MySQL一致。
- `[FIELDS] ENCLOSED BY 'char'`：定义每列数据的enclosed符号。

  例如，某一列数据为`"a"`，定义`enclosed by '"'`后，导入数据时先将`"a"`前后的`"`移除，然后导入数据。
- `[LINES] TERMINATED BY 'string'`：定义行分隔符，默认为`\n`。
- `IGNORE number LINES`：设置导入数据时忽略开始的某几行。例如`IGNORE 1 LINES`，导入数据时忽略第一行数据，即第一行数据不会导入AnalyticDB for MySQL。
- `(column_name_or_user_var,...)`：设置导入的列，如果不设置，默认按照列的顺序来导入数据。
  - 如果导入列数 > 目标表的列数，则系统自动忽略多余的列。
  - 如果导入列数 < 目标表的列数，则后续缺少的列将自动填充默认值。

注意事项

- 客户端开启`local-file`。

  以mysql-client为例，您需要在`my.cnf`文件中增加以下配置开启`local-file`功能。

  ```
  cat ~/.my.cnf
  [mysqld]
  local-infile
  [mysql]
  local-infile
  ```

  更多`my.cnf`文件信息，请参见MySQL官方文档。
- 导入数据时无法保证操作的原子性。
  - 在`IGNORE`模式下，忽略导入失败的数据行。
  - 在`REPLACE`模式下，一旦有数据行导入失败，系统将中止后续`INSERT`操作，因此可能存在部分行数据导入，部分行数据未导入的情况。
- 支持通过`SHOW WARNINGS`命令，查看失败行的错误信息。

示例

本示例将本地文件`out.bak`中的数据导入AnalyticDB for MySQL的`test`表中。`out.bak`文件中共有5000行数据，列分隔符为`\t`，行分隔符`\n`，其中第10行数据存在问题，如下所示。

```
1 bb
2 bb
3 bb
4 bb
5 bb
6 bb
7 bb
8 bb
9 bb
bb   10
...
```

1. 连接AnalyticDB for MySQL集群，通过CREATE DATABASE和CREATE TABLE，在 `adb_demo` 数据库下创建表 `test` 表，从本地文件导入的数据将存储在 `test` 表中。

```
CREATE TABLE test (
a int(11) NOT NULL DEFAULT '0',
b varchar(8) NOT NULL,
PRIMARY KEY (a)
) DISTRIBUTED by hash(a);
```

2. 在mysql-client中执行LOAD DATA命令将本地文件 `out.bak` 中的数据导入AnalyticDB for MySQL的 `test` 表中。

```
#执行LOAD DATA命令， ignore模式下，部分行导入失败但不影响其他行导入。
mysql>  load data local infile '~/out.bak' ignore into table test FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' ;
Query OK, 4999 rows affected, 1 warning (0.69 sec)
mysql> show warnings;
+---------+-------+--------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| Level   | Code  | Message
|
+---------+-------+--------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| Warning | 13000 | com.alibaba.cloud.analyticdb.common.sql.hardcode.HardParserException: syntax error :syntax error => IDENTIFIER is n
ot value type pos:33 row: 0 and ceil:0 |
+---------+-------+--------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
1 row in set (0.01 sec)
mysql> select count(1) from test;
+----------+
| count(1) |
+----------+
|     4999 |
+----------+
1 row in set (0.14 sec)
#执行LOAD DATA命令，replace模式下，部分行导入失败后立即终止后续导入操作。
mysql> load data local infile '~/out.bak' replace into table test FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' ;
ERROR 1064 (42000): [13000, 2019061210070703000511314203303000266] syntax error :syntax error => IDENTIFIER is not value type pos:34 ro
w: 0 and ceil:0
#执行LOAD DATA命令，导入数据时会跳过前10条数据。
mysql> load data local infile '~/out.bak' replace into table test FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 10 LINES;
Query OK, 4990 rows affected (0.37 sec)
mysql> show warnings;
Empty set (0.00 sec)
```

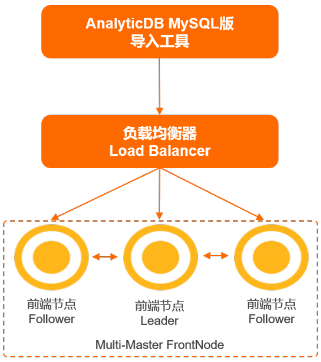
2.10.2. 使用AnalyticDB MySQL版导入工具导入本地数据

本文介绍如何使用AnalyticDB MySQL版导入工具将本地数据导入至AnalyticDB MySQL版集群。

功能介绍

- **工作原理**

  AnalyticDB MySQL版导入工具通过JDBC协议接入负载均衡器（Load Balancer），负载均衡器下可连接多个前端节点（FrontNode），前端节点主要负责MySQL协议解析、SQL解析、数据写入、查询调度等，数据经由前端节点转发给存储节点进行导入。



- **功能特点**

  相较于MySQL Load Data工具，AnalyticDB MySQL版导入工具有如下特点：

  - 支持通过配置 `batchSize` 、并发数等来控制导入速度，实现以最大化的吞吐量进行数据导入。参数配置详情，请参见步骤三：脚本准备。
  - 支持单个文件、多个文件或文件夹的导入，无需启动多个MySQL Load Data进程并行导入。
  - 能够利用并行、Batch、池化、流水线执行（读写非串行）、GC-less programming、大块文件顺序IO读等技术实现更佳的导入性能，导入工具如果配置合理，可以最大化AnalyticDB MySQL版集群的写入吞吐（Throughput）。

导入流程介绍

步骤	说明
步骤一：下载并安装导入工具	下载AnalyticDB MySQL版导入工具并完成解压和安装。
步骤二：导入数据准备	准备需要导入的源数据。
步骤三：脚本准备	修改导入脚本模板中的参数，准备数据导入脚本。
步骤四：执行导入	执行导入脚本将本地数据导入至AnalyticDB MySQL版集群。

步骤一：下载并安装导入工具

1. 下载导入工具压缩包。
2. 执行如下命令新建一个目录（本文示例中为*/u01/loadata*）：

```
# cd /u01/loadata && tar -xvfz adb-import-tool.tar.gz
```

解压后将会产生如下文件：

```
adb-import.sh.template
adb-import.sh.template.md5
adb-import-tool.jar
adb-import-tool.jar.md5
```

> **说明** 您可以通过执行 `java -version` 命令来确认是否已安装Java，以及Java版本是否为1.8或以上。

步骤二：导入数据准备

> **说明** 您还可以使用Linux的 `split` 命令对大文件进行切分（建议切分后的文件大小为1 GB~2 GB），文件切分后会形成更多的分片，更利于AnalyticDB MySQL版导入工具执行并行导入，从而提升导入速度，缩短导入时间。
>
> 例如，您可以将一个128 GB的文件 `filename.txt` 均匀地切分为64份，每个文件2 GB，那么AnalyticDB MySQL版导入工具将会以64的并行度来读取文件。切分命令如下：
>
> ```
> # split -l$((`wc -l < filename.txt`/64 + 1)) filename.txt filename.txt.split -da 2;
> ```

1. 确认需要导入的文件或文件夹的绝对路径。
2. 确认导入文件的行分隔符和列分隔符。更多关于分隔符的说明，请参见步骤三：脚本准备。
3. 确认导入文件的列顺序必须与建表DDL定义顺序一致，可以在数据库中执行SHOW CREATE TABLE确保列顺序。列数必须大于1。

   例如，可以使用如下命令定义表结构：

   ```
   CREATE TABLE `product_info` (
   `id` bigint NOT NULL,
   `name` varchar,
   `price` decimal(15, 2) NOT NULL,
   )
   DISTRIBUTE BY HASH(`id`)
   INDEX_ALL='Y';
   ```

   合法文件内容如下：

   ```
   1|tv|1000.0
   2|computer|2000.0
   3|cup|15.8
   ```

4. 确认导入文件最后一列不包含多余的列分隔符。

   例如 `1|abc|3.0` 是合法的，而 `1|abc|3.0|` 是非法的。

   > **说明**
   > - 对于空字符串，导入工具默认按照null处理，例如导入的文件为 `4||5.0` ，则 `name` 列会插入null值，而不是字符串 `''` 。
   > - 如果列中存在自增列，文件中无需特殊体现，导入工具可以兼容。

步骤三：脚本准备

`adb-import.sh.template` 是一个模板脚本，支持自定义脚本名称，例如要导入的表名为 `product_info` ，您可以将脚本命名为 `adb-import-product_info.sh` 。您可以复制一份模板脚本，并根据数据导入场景编辑导入脚本参数。

```
# 脚本中的参数说明 #
#-------------------------------- #
#           下面是必填参数
# -------------------------------- #
####################################
# java命令路径
# 注:如果在控制台下可直接执行java命令，则无需设置。
```

```
####################################
java_cmd=java
####################################
# 导入程序jar包地址
# 如果在脚本所在目录执行，则无需设置，否则需要设置绝对路径。
####################################
jar_path=adb-import-tool.jar
####################################
# 配置数据库连接参数
# 注:确保database已经在ADB中创建
#     如果encryptPassword=true需要填写base64加密后的密码。
####################################
host=host
port=3306
user=adbuser
password=pwd
database=dbname
encryptPassword=false
####################################
# 导入表名
####################################
tableName=please_set_table_name
####################################
# 导入的文件或文件夹的绝对路径，支持:
# 1）单个文件或单个文件夹
# 或者
# 2）同时导入多个文件，多个文件的路径间用英文逗号（,）分隔。
####################################
dataPath=please_set_data_file_path_or_dir_path
####################################
# 导入并行度
# 注:越大的并行度越有利于发挥ADB的性能
#     建议值>=16，<=96。
####################################
concurrency=64
####################################
# 导入写入VALUES的数量
# 注:越大的批次越有利于发挥ADB的性能
#     但也要结合单行的长度，不宜过大
#     建议值>=1024，<=4096。
####################################
batchSize=4096
####################################
# 导入文件编码，UTF-8或者GBK。
####################################
encoding=UTF-8
####################################
# 行分隔符
# 支持使用可见符（例如"\\n"）和不可见符作为分隔符。
# 如需使用不可见符作为分隔符，需使用16进制来表示。
* 例如\x0d\x06\x08\x0a需使用十六进制表示为"hex0d06080a"。
####################################
lineSeparator="\\n"
####################################
# 列分隔符
# 支持使用可见符（例如"\\|"）或不可见符作为分隔符。
# 如需使用不可见符作为分隔符，需使用16进制来表示。
# 例如\x07\x07需使用十六进制表示为"hex0707"。
####################################
delimiter="\\|"
# ------------------------------- #
#            下面是选填参数
# ------------------------------- #
####################################
# jvm参数
####################################
jvmopts="-Xmx12G -Xms12G"
####################################
# 当dataFile是一个文件夹时，
# 并行读取文件的数量。
####################################
maxConcurrentNumOfFilesToImport=64
####################################
# 选填，默认值：false，空字符串会变成null;
# 若设置为true,则空字符串会变成''。
# 建议设置为默认值false。
####################################
nullAsQuotes=false
####################################
# 每个文件导入完毕后是否打印目标表实际行数。
# 选填，默认值：false。
```

```
###################################
printRowCount=false
###################################
# SQL执行失败时候会打印SQL，
# 设置错误SQL的打印截断长度。
# 选填，默认值：1000。
###################################
failureSqlPrintLengthLimit=1000
###################################
# 导入数据时是否不执行INSERT，仅打印INSERT SQL命令。
# 选填，默认false。
###################################
disableInsertOnlyPrintSql=false
###################################
# 跳过表头。选填，默认false。
###################################
skipHeader=false
###################################
# INSERT SQL的缓冲数量。
# 便于发送给ADB的时候做到
# IO和计算分离，提高客户端性能。
###################################
windowSize=128
###################################
# 是否转义列中的\以及'符号。选填，默认true，表示需要转义。
# 转义对于客户端有一定字符串解析造成的性能损失，
# 特殊情况下保证没有转义字符的情况下，可以置false。
###################################
escapeSlashAndSingleQuote=true
###################################
# 导入数据遇到错误，是否忽略失败的批次。
###################################
ignoreErrors=false
###################################
# 导入数据遇到错误，是否打印出错的SQL。
###################################
printErrorSql=true
###################################
# 当导入数据遇到错误，且printErrorSql=true时，
# 是否打印出错的栈信息。
###################################
printErrorStackTrace=true
```

步骤四：执行导入

1. 执行如下命令导入脚本：

   ```
   sh adb-import-product_info.sh;
   ```

   若打印出如下日志表示执行正常：

   ```
   [2021-03-13 17:50:24.730] add consumer consumer-01
   ```

2. 导入期间，导入工具不会进行过多的日志滚动，您可以查询数据库获取导入进度，例如查询目标表的总行数，命令如下：

   ```
   mysql > select count(*) from dbname.product_info;
   ```

   > **说明** 数据导入期间若执行SQL出错，导入工具会立即停止导入，并打印出错误SQL的详细信息。此时导入的数据是不完整的，可通过执行 `TRUNCATE TABLE table_name` 清空表后重新导入；或者执行 `DROP TABLE table_name` 删除表后再新建表来重新导入。

3. 导入结束后会打印每个文件的读取行数、耗时，以及总体的耗时，最后会提示是否全部执行成功。 如果全部执行成功，则打印 `all import finished succesfully` ，否则打印 `all import finished with ERROR!` 。详细的导入行数，请查询数据库进行校验。

常见问题

- Q：如何查验客户端或其所在服务器负载是否存在瓶颈？

  A：若客户端存在瓶颈，将无法最大化压测数据库，此时您可以通过查看以下常用命令来查验客户端自身以及所在服务器负载是否存在瓶颈。

命令	说明
top	查看CPU使用率。
free	查看内存占用。
vmstat 1 1000	查看综合负载。
dstat -all --disk-util或iostat 1 1000	查看磁盘的读带宽和使用率。

命令	说明
jstat -gc <pid> 1000	查看导入工具Java进程的垃圾回收（Garbage Collection，简称GC）详情，如果GC频繁，可以尝试适当扩大JVM参数 `jvmopts` 中的堆内存大小，例如将其扩大到 `-Xmx16G -Xms16G` 。

- Q：如何将导入脚本参数化？

  A：如果确保导入文件的行列分隔符一致，可修改导入脚本中的 `tableName` 和 `dataPath` 参数，通过传入不同的表名和文件路径参数，实现一个脚本导入多个表的需求。

  示例如下：

  ```
  tableName=$1
  dataPath=$2
  ```

  使用参数化的方式执行导入。

  ```
  # sh adb-import.sh table_name001 /path/table_001
  # sh adb-import.sh table_name002 /path/table_002
  # sh adb-import.sh table_name003 /path/table_003
  ```

- Q：如何将导入程序放在后台运行？

  A：您可以执行如下命令在后台运行导入程序：

  ```
  # nohup sh adb-import.sh &
  ```

  导入程序在后台开始运行后，您可以执行以下命令查看检查日志，如果打印异常信息栈则说明导入存在错误，需要根据异常信息进行问题排查。命令如下：

  ```
  # tail -f nohup.out
  ```

  您还可以使用如下命令查看导入进程是否仍正常执行：

  ```
  # ps -ef|grep import
  ```

- Q：如何忽略导入程序中的错误行？

  A：导入程序中的错误行可以分为如下两类：

  - 执行SQL出错。

    针对此类错误，您可以通过设置参数 `ignoreErrors=true` 来忽略错误行。此时会在执行结果中打印详细的出错文件、起始行号（因设置了 `batchSize` ，错误行会在起始行号后的 `batchSize` 行内）以及执行出错的SQL。

  - 文件列数不符合预期。

    当文件列数不符合预期时，系统会立即停止导入该文件并打印出错误信息，但由于该错误是由于非法文件导致的，因此并不会被忽略，您需要手动排查文件的正确性。此类错误会打印如下错误信息：

    ```
    [ERROR] 2021-03-22 00:46:40,444 [producer- /test2/data/lineitem.csv.split00.100-41] analyticdb.tool.ImportTool
    (ImportTool.java:591) -bad line found and stop import! 16, file = /test2/data/tpch100g/lineitem.csv.split00.100, rowCount = 7, current
    row = 3|123|179698|145|73200.15|0.06|0.00|R|F|1994-02-02|1994-01-04|1994-02-
    23|NONE|AIR|ongside of the furiously brave acco|
    ```

- Q：如何缩小导入失败原因的排查范围？

  A：为帮助更快的定位导入失败原因，您可以从如下几个方面来缩小失败原因的排查范围：

  - 当导入失败时，AnalyticDB MySQL版导入工具会打印错误日志以及详细的错误原因，默认会截断SQL语句（最长支持1000个字符），若需要打印更全的SQL信息，您可以使用如下命令将 `failureSqlPrintLengthLimit` 参数扩大至一个合理值（例如1500）：

    ```
    printErrorSql=true
    failureSqlPrintLengthLimit=1500;
    ```

  - 由于SQL设置了 `batchSize` ，通常是上千行批量执行的SQL，不利于分辨错误行，您可以缩小 `batchSize` 参数（例如设置为 *10*）以便于定位错误的行。参数修改命令如下：

    ```
    batchSize=10;
    ```

  - 如果文件已进行了切分且已知错误的行所在的文件分片，为了复现问题，可通过修改 `dataPath` 参数来导入存在错误行的单个文件，查看错误信息。语句如下：

    ```
    dataPath=/u01/this/is/the/directory/where/product_info/stores/file007;
    ```

- Q：如何在Windows环境下运行导入程序？

  A：Windows环境暂未提供bat批处理脚本，您可以直接使用如下方法调用JAR文件来执行：

  ```
  usage: java -jar adb-import-tool.jar [-a <arg>] [-b <arg>] [-B <arg>] [-c <arg>]
        [-D <arg>] [-d <arg>] [-E <arg>] [-f <arg>] [-h <arg>] [-I <arg>]
        [-k <arg>] [-l <arg>] [-m <arg>] [-n <arg>] [-N <arg>] [-O <arg>]
        [-o <arg>] [-p <arg>] [-P <arg>] [-Q <arg>] [-s <arg>]  [-S <arg>]
        [-t <arg>] [-T <arg>] [-u <arg>] [-w <arg>][-x <arg>] [-y <arg>] [-z <arg>]
  ```

参数	是否必填	说明
-h,--ip <arg>	必填	AnalyticDB MySQL版集群的连接地址。
-u,--username <arg>		AnalyticDB MySQL版集群的数据库账号。
-p,--password <arg>		AnalyticDB MySQL版集群的数据库账号对应的密码。
-P,--port <arg>		AnalyticDB MySQL版集群使用的端口号。
-D,--databaseName <arg>		AnalyticDB MySQL版集群的数据库名称。
-f,--dataFile <arg>		需要导入的文件或文件夹的绝对路径，支持如下几种导入场景： • 仅导入单个文件或单个文件夹。 • 同时导入多个文件，多个文件的路径间用英文逗号（,）分隔。
-t,--tableName <arg>		需要导入的表名。
-a,--createEmptyFinishFilePath <arg>	选填	导入完毕后是否生成一个标志文件。默认为空字符串，表示不生成。若需要生成标志文件，直接输入文件名即可。例如您可以设置 -a file_a ，即可生成一个名为 file_a 的标志文件。
-b,--batchSize <arg>		设置 INSERT INTO tablename VALUES (..),(..) 中批量写入VALUES的数量。默认值：1。 说明 为更好地实现数据批量写入效果，建议将该值设置在1024~4096之间。
-B,--encryptPassword <arg>		数据库密码是否使用加密算法加密。默认值：false，表示不使用加密算法加密数据库密码。
-c,--printRowCount <arg>		每个文件导入完毕后是否打印目标表实际行数。默认值：false，表示不打印。
-d,--skipHeader <arg>		是否跳过表头。默认值：false，表示不跳过表头。
-E,--escapeSlashAndSingleQuote <arg>		是否转义列中的 \ 以及 ' 符号。默认值：true，表示需要转义。 说明 转义对于客户端字符串解析的性能有一定损失，若确保需要导入的文件中没有转义字符，可以设置该参数为false。
-I,--ignoreErrors <arg>		导入数据遇到错误，是否忽略失败批次。默认值：false，表示不忽略。
-k,--skipLineNum <arg>		跳过的行数，类似 IGNORE number {LINES | ROWS} 参数。默认值：0，表示不跳过。
-l,--delimiter <arg>		列分隔符。AnalyticDB MySQL版默认使用可见符 \\| 作为列分隔符。同时也支持使用不可见符作为分隔符，如需使用不可见符，需要使用十六进制来表示。例如， \x07\x07 需使用十六进制表示为 hex 0707 。
-m,--maxConcurrentNumOfFilesToImport <arg>		当dataFile是一个文件夹时，并行读取文件的数量。默认值： Integer.MAX_VALUE ，表示读所有文件。
-n,--nullAsQuotes <arg>		当需要导入的文件中存在 || 时，是否需要将其设置为 '' 。默认值：false，表示不将 || 设置为 '' ，而是设置为null。
-N,--printErrorSql <arg>		导入数据遇到错误，是否打印出错的SQL。默认值：true，表示打印出错误的SQL。
-O,--connectionPoolSize <arg>		AnalyticDB MySQL版数据库连接池大小。默认值：2。
-o,--encoding <arg>		文件编码方式。取值范围：GBK或UTF-8（默认值）。
-Q,--disableInsertOnlyPrintSql <arg>		导入数据库时是否不执行INSERT，仅打印INSERT的SQL命令。选填，默认值：false，表示执行INSERT。
-s,--lineSeparator <arg>		行分隔符。AnalyticDB MySQL版默认使用可见符 \\n 作为行分隔符。同时也支持使用不可见符作为分隔符，如需使用不可见符，需要使用十六进制来表示。例如， \x0d\x06\x08\x0a 需使用十六进制表示为 hex0d06080a 。
-S,--printErrorStackTrace <arg>		当导入数据遇到错误，且 printErrorSql=true 时，是否打印出错的栈信息。默认值：false，表示不打印。
-w,--windowSize <arg>		INSERT SQL的缓冲数量。便于将INSERT SQL命令发送至AnalyticDB MySQL版时，实现流水线加速以及IO和计算分离，从而提高客户端性能。默认值：128。

参数	是否必填	说明
-x,--insertWithColumnNames <arg>		执行 INSERT INTO 命令时是否带上列名，即是否执行 INSERT INTO tb(column1, column2) 命令进行数据导入。默认值：true，表示导入时需要带上列名。
-y,--failureSqlPrintLengthLimit <arg>		当执行INSERT 命令失败时需要打印错误SQL，使用该参数设置错误SQL的打印截断长度。默认值：1000。
-z,--connectionUrlParam <arg>		数据库连接参数。默认值： ?characterEncoding=utf-8 。 示例： ?characterEncoding=utf-8&autoReconnect=true 。

案例1：使用默认参数配置导入单个文件，命令如下：

```
java -Xmx8G -Xms8G -jar adb-import-tool.jar -hyourhost.ads.aliyuncs.com -uadbuser -ppassword -P3306 -Dtest --dataFile /data/lineitem.sample --tableName LINEITEM
```

案例2：修改相关参数实现最大化吞吐导入文件夹下所有文件，命令如下：

```
java -Xmx16G -Xms16G -jar adb-import-tool.jar -hyourhost.ads.aliyuncs.com -uadbuser -ppassword -P3306 -Dtest --dataFile /data/tpch100g --tableName LINEITEM --concurrency 64 --batchSize 2048
```

2.11. 异步提交导入导出任务

AnalyticDB MySQL版中支持通过异步方式提交数据导入导出任务。

异步提交任务

- **语法**

```
submit job INSERT overwrite INTO xxx SELECT ...FROM
```

执行上述命令后，将返回一个job_id。

- **示例**

```
mysql> submit job INSERT overwrite INTO test SELECT * FROM test_external_table;
+---------------------------------------+
| job_id                                |
+---------------------------------------+
| 20171121222029172031009082033030007l5 |
```

- 设置优先级调度

3.1.3.6及以上版本支持优先级调度。您可通过hint语法标识异步任务的优先级，默认优先级值为1，值越大表示优先级越高，会被系统优先调度。

```
mysql> /*+async_job_priority=10*/ submit job INSERT overwrite INTO test SELECT * FROM test_external_table;
```

查询异步任务状态

- **语法**

```
SHOW job STATUS WHERE job='job_id';
```

- **示例**

```
mysql> SHOW job STATUS WHERE job='20171121222029172031009082033030007l5';
+---------------------------------------+-------------+---------+----------+----------------------+----------------------+-------------------------+
| job_id                                | schema_name | status  | fail_msg | create_time          | update_time          | definition              |
+---------------------------------------+-------------+---------+----------+----------------------+----------------------+-------------------------+
| 20171121222029172031009082033030007l5 | test        | RUNNING | NULL     | 2017-11-21 22:20:31.0 | 2017-11-21 22:20:40.0 | insert into test select * from test |
```

- **任务状态说明**
  - INIT：任务进入队列
  - RUNNING：后台开始执行任务
  - FINISH / FAILED：任务成功或失败

终止任务

- **语法**

```
CANCEL job 'job_id'
```

- **示例**

```
mysql> CANCEL job '20171121222029172031009082033030007l5';
```

- **说明**

- 未调度起来的任务和已完成（失败或成功）的任务会被移除队列。
- 正在运行的任务会被终止，已导入的数据也会被回滚。

3.SQL手册

3.1. 基础数据类型

本文介绍云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）支持哪些数据类型以及与MySQL数据类型的差异对比。ADB不支持的数据类型请参见不支持的数据类型及运算符。

ADB支持的数据类型

类型	数据类型	说明	与MySQL数据类型差异
数值类型	`boolean` 布尔类型	值只能是 `0` 或 `1`，存储字节数1比特位。 • 取值 `0` 的逻辑意义为假。 • 取值 `1` 的逻辑意义为真。	一致。
	`tinyint` 微整数类型	取值范围 `-128` ~ `127`，存储字节数1字节。	一致。
	`smallint` 小整数类型	取值范围 `-32768` ~ `32767`，存储字节数2字节。	一致。
	`int` 或 `integer` 整数类型	取值范围 `-2147483648` ~ `2147483647`，存储字节数4字节。	ADB中的 `int` 对应MySQL中的 `int` 或者 `mediumint`。
	`bigint` 大整数类型	取值范围 `-9223372036854775808` ~ `9223372036854775807`，存储字节数8字节。	一致。
	`float` 单精度浮点数	取值范围 `-3.402823466E+38` ~ `-1.175494351E-38`，`0`，`1.175494351E-38` ~ `3.402823466E+38`，IEEE标准，存储字节数4字节。	一致。
	`double` 双精度浮点数	取值范围 `-1.7976931348623157E+308` ~ `-2.2250738585072014E-308`，`0`，`2.2250738585072014E-308` ~ `1.7976931348623157E+308`，IEEE标准，存储字节数8字节。	一致。
	`decimal(m,d)` 或 `numeric`	`m` 是数值的最大精度，取值范围为 `1` ~ `1000`；`d` 是小数点右侧数字的位数，要求 `d` ≤ `m`。	• MySQL支持的最大精度为 `65`。 • ADB支持的最大精度为 `1000`。
字符类型	`varchar` 变长字符串类型	存储字节数最大为16MB，使用时无需指定存储长度。	ADB中的 `varchar` 对应MySQL中的 `char`、`varchar`、`text`、`mediumtext` 或者 `longtext`。
	`binary` 二进制字符串类型	存储字符长度	ADB中的 `binary` 对应MySQL中的 `binary`、`varbinary` 或者 `blob`。
时间类型	`date` 日期类型	取值范围 `'0001-01-01'` ~ `'9999-12-31'`，支持的数据格式为 `'YYYY-MM-DD'`，存储字节数为4字节。	• MySQL支持 `0000-00-00`。 • ADB对时间类型的数值会进行合法性校验。如果开启参数 `ILLEGAL_DATE_CONVERT_TO_NULL_ENABLE=true`，在不合理的数值写入时，例如 `0000-00-00`，ADB会自动将其转化为 `NULL`。请确保写入的日期和时间有意义。
	`time` 时间类型	取值范围 `'00:00:00'` ~ `'23:59:59'`，支持的数据格式为 `'HH:MM:SS'`，存储字节数为8字节。	• MySQL支持自定义精度。 • ADB支持的精确为毫秒，即小数点后三位。

类型	数据类型	说明	与MySQL数据类型差异
	`datetime` 时间戳类型	取值范围 `'0001-01-01 00:00:00.000'` UTC~ `'9999-12-31 23:59:59.999'` UTC，支持的数据格式为 `'YYYY-MM-DD HH:MM:SS'` ，存储字节数为8字节。 说明 `datetime` 默认UTC时间，且不支持可更改。	• MySQL支持 `0000-00-00` ，支持自定义精度。 • ADB对时间类型的数值会进行合法性校验。如果开启参数 `ILLEGAL_DATE_CONVERT_TO_NULL_ENABLE=true` ，在不合理的数值写入时，例如 `0000-00-00` ，ADB会自动将其转化为 `NULL` 。请确保写入的日期和时间有意义。
	`timestamp` 时间戳类型	时间戳类型，取值范围 `'0001-01-01 00:00:00.000'` UTC~ `'9999-12-31 23:59:59.999'` UTC，支持的数据格式为 `'YYYY-MM-DD HH:MM:SS'` ，存储字节数为4字节。 说明 `timestamp` 默认为系统时区，可以在SESSION中设置时区。	• MySQL支持自定义精度。 • ADB支持的精确为毫秒，即小数点后三位。
空间类型	`point`	地理坐标	一致
	`json`	请参见JSON。	一致

3.2. 复杂数据类型

3.2.1. Array

3.1.1及以后版本的AnalyticDB MySQL版集群支持Array、Map数据类型。本文介绍了Array数据类型的定义、注意事项及使用示例。

Array类型定义

Array类型，存储数组，可重复，含义类似Java中LIST。Array中数据为相同类型。例如，列A定义 `array<int>` ，那么A中子元素均为int类型。支持嵌套，例如 `array<array<string>>` 。

注意事项

Array或Map列不支持构建索引，因此，在SQL查询语句中不建议直接过滤，需配合其他检索条件过滤。尽量减少扫描数据量。

使用示例

创建表

```
Create Table `array_test` (
 `a` int,
 `b` array<int>,
 `c` array<array<string>>,
 primary key (`a`)
) DISTRIBUTE BY HASH(`a`)
```

写入数据

比如插入一行数据，其中 `b=[1,2,3], c=[["a"],["b","c"]]` ：

```
insert into array_test values (1, '[1,2,3]', '[["a"],["b","c"]]');
```

查询数据

```
mysql> select * from array_test;
+------+---------+-------------------+
| a    | b       | c                 |
+------+---------+-------------------+
|    1 | [1,2,3] | [["a"],["b","c"]] |
+------+---------+-------------------+
1 row in set (0.08 sec)
```

注意

- Array类型下标是从1开始，而不是0。
- Array的取址操作： `b[1]` 等价于函数 `element_at(b, 1)` 。

```
mysql> select a,b[1],element_at(b,1),c[2],element_at(c,2) from array_test;
+------+------+-----------------+-----------+-----------------+
| a    | b[1] | element_at(b,1) | c[2]      | element_at(c,2) |
+------+------+-----------------+-----------+-----------------+
|    1 |    1 |               1 | ["b","c"] | ["b","c"]       |
+------+------+-----------------+-----------+-----------------+
1 row in set (0.11 sec)
```

查询结果返回类型为Array或Map的列，以JSON格式字符串输出。比如 `c[2]` 对应的是嵌套子列，类型为 `array<string>` ，那么以JSON格式返回。

支持的函数

函数	描述	返回类型
element_at	取值，下标从1开始，例如 `element_at(array[1,2], 1) ==> 1` 。	T
size	元素个数。	int
contains	是否包含子元素，例如 `contains(array[1,2], 2) ==> 1` 。	bool类型
array_max	取子元素最大值。	T
array_min	取子元素最小值。	T
array_position	取第一次出现的Index，例如 `array_position(array['a','b','b'],'b') ==>2` 。	int
array_remove	移除子元素，例如 `array_remove(array['a','b','b'],'b')==> ['a']` 。	array<T>
array_sort	排序， `array_sort(array[3,2,1]) ==> [1,2,3]` 。	array<T>
reverse	将数组中的子元素反转，例如 `(array[5,9,3]) ==> [3,9,5]` 。	array<T>
shuffle	把数组中的元素按随机顺序重新排列，例如 `shuffle(array[1,5,8])==> [5,1,8]` 。	array<T>
slice	截取子元素，例如 `array slice(array[1,2,3,4,5], 3,2) ==> [3,4]` 。	array<T>
concat	子元素合并且包含重复，例如 `concat(array[1], array[1,2]) ==>[1,1,2]` 。	array<T>
array_distinct	子元素去重，例如 `array_distinct(array[1,1,2]) ==>[1,2]` 。	array<T>
array_union	子元素合并且去重，例如 `array_union(array[1], array[1,2])==>[1,2]` 。	array<T>
array_intersect	求交集，例如 `array_intersect(array[1], array[1,2]) ==>[1]` 。	array<T>
array_join	类似Joiner拼接Array元素，例如 `array_join(array[1,2,3,4],'a') ==>1a2a3a4` 。	string
flatten	降维，例如 `flatten(array[array[1,2],array[3]])==> [1,2,3]` 。	array<X>

3.2.2. Map

3.1.1及以后版本的AnalyticDB MySQL版集群支持Array、Map数据类型。本文介绍了Map数据类型的定义、注意事项及使用示例。

Map类型定义

Map类型，存储 `k-v` 键值对，含义类似Java中的Map。其中key类型要求是原生类型（如 `tinyint` 、 `boolean` 、 `smallint` 、 `int` 、 `bigint` 、 `float` 、 `double` 、 `string` ），`value` 类型可以是原生类型，也可以是Map或Array类型。例如，列定义 `map<int, string>` 、 `map<int, map<int, string>>` 。

注意事项

- Array或Map列不支持构建索引，因此，在SQL查询语句中不建议直接过滤，需配合其他检索条件过滤。尽量减少scan数据量。
- 针对一条Map记录，key不能重复。
- 不保证key的写入顺序。比如写入时 `{"a":1,"b":2,"d":3}` ，查询时结果为 `{"d":3, "a":1, "b":2}` 。

使用示例

创建表

```
Create Table `map_test` (
 `a` int,
 `b` map<int, string>,
 `c` map<int, map<int, string>>,
 PRIMARY KEY (`a`)
) DISTRIBUTE BY HASH(`a`)
```

写入数据

比如插入一行数据，其中 `b={1:"a"}, c={1:{11:"a"},2:{22:"b"}}` :

```
INSERT INTO map_test VALUES(1, '{1:"a"}','{1:{11:"a"},2:{22:"b"}}');
```

查询数据

```
SELECT * FROM map_test;
+------+---------+--------------------------+
| a    | b       | c                        |
+------+---------+--------------------------+
|    1 | {1:"a"} | {1:{11:"a"},2:{22:"b"}} |
+------+---------+--------------------------+
1 row in set (0.07 sec)
```

> **注意**
>
> - Map类型的根据key取值操作： `element_at(b, 1)` ，其中1表示key值，不是下标。
> - Map类型，size函数返回的是key、value的个数总和。
> - `map_keys` 和 `map_values` 返回Array类型。

```
SELECT element_at(c,1), element_at(element_at(c,1),11) FROM map_test;
+-----------------+---------------------------------+
| element_at(c,1) | element_at(element_at(c,1),11) |
+-----------------+---------------------------------+
| {11:"a"}        | a                               |
+-----------------+---------------------------------+
1 row in set (0.07 sec)
SELECT map_keys(b),map_values(b),size(b),size(map_keys(b)),size(map_values(b)) FROM map_test;
+-------------+---------------+---------+------------------+--------------------+
| map_keys(b) | map_values(b) | size(b) | size(map_keys(b)) | size(map_values(b)) |
+-------------+---------------+---------+------------------+--------------------+
| [1]         | ["a"]         |       2 |                1 |                  1 |
+-------------+---------------+---------+------------------+--------------------+
1 row in set (0.08 sec)
SELECT map_keys(c),map_values(c),size(c),size(map_keys(c)),size(map_values(c)) FROM map_test;
+-------------+---------------------+---------+------------------+--------------------+
| map_keys(c) | map_values(c)       | size(c) | size(map_keys(c)) | size(map_values(c)) |
+-------------+---------------------+---------+------------------+--------------------+
| [1,2]       | [{11:"a"},{22:"b"}] |       4 |                2 |                  2 |
+-------------+---------------------+---------+------------------+--------------------+
1 row in set (0.08 sec)
```

支持的函数

函数	描述	返回类型
element_at	根据key值，获取value。例如 `element_at(map(array["a","b"],array[1,2]), a) ==> 1` 。	V
size	key个数与value个数总和。	int
map_keys	获取所有key列表。	array<K>
map_values	获取所有value列表。	array<V>

3.2.3. JSON

为赋能用户、降低用户处理半结构化数据的难度，AnalyticDB MySQL版提供了半结构化数据检索功能即JSON索引。

背景信息

大数据时代结构化数据检索已有多元化的、丰富的解决方案。但是，事实上大多数大数据都是半结构化，并且半结构化数据的数据量仍旧急剧增长。理解和分析半结构化数据的难度比结构化数据大很多，急需成熟的解决方案处理半结构化数据。为赋能用户、降低用户处理半结构化数据的难度，云原生数据仓库AnalyticDB MySQL版（简称AnalyticDB MySQL版）提供了半结构化数据检索功能，即JSON索引。

注意事项

- 如果在创建表时指定了某一列的数据类型为JSON，AnalyticDB MySQL版会自动创建JSON索引。

  JSON索引创建后，您可以通过ALTER语句来新增或删除索引。更多详情，请参见创建表。
- AnalyticDB MySQL版支持标准JSON格式，写入JSON串时必须严格符合标准JSON格式规范。
- JSON类型的数据列，不支持设置Default值。

创建表

例如，以下示例中的 `vj` 列的数据类型为JSON，建表成功后AnalyticDB MySQL版自动为 `vj` 列构建JSON索引。

```
CREATE TABLE json_test(
  id int,
  vj json COMMENT 'json类型，自动创建索引'
)
DISTRIBUTED BY HASH(id);
```

JSON索引创建后，您可以通过ALTER语句来删除或新增索引。语句如下：

- 删除索引

  ```
  ALTER TABLE db_name.table_name DROP KEY index_name;
  ```

  > **说明** 您可以使用SHOW INDEXES命令查看目标表中的所有索引信息，包括索引名称（即 `key_name` ）。更多详情，请参见SHOW。
- 新增索引

  ```
  ALTER TABLE db_name.table_name ADD KEY index_name(column_name);
  ```

JSON格式要求

写入数据时，AnalyticDB MySQL版对JSON数据中的属性键 `key` 和属性值 `value` 有如下要求：

- **属性键 `key`**

  必须使用双引号（ `""` ）将 `key` 引起来，例如 `{"addr":"xyz"}` 中的 `"addr"` 。
- **属性值 `value`**
  - 属性值 `value` 支持的数据类型为：BOOLEAN、NUMBER、VARCHAR、ARRAY、OBJECT、NULL。

    > **说明**
    > - NUMBER不能超过DOUBLE的取值范围，否则会写入报错。
    > - ARRAY类型可以为PLAIN ARRAY或嵌套ARRAY。例如， `{"hobby":["basketball", "football"]}` 和 `{"addr":[{"city":"beijing", "no":0}, {"city":"shenzhen", "no":0}]}` 。
  - 如果 `value` 是字符串类型，必须使用双引号（ `""` ）将 `value` 引起来。

    > **说明** 如果 `value` 是字符串类型，且 `value` 中包含双引号，需要做转义处理。例如， `{"addr":"xyz"ab"c"}` 中的 `value` ，即 `"xyz"ab"c"` 部分，需转义为 `"xyz\"ab\"c"` ，但由于写入过程中 `\` 会被转义，因此写入的数据应为 `{"addr":"xyz\\"ab\\"c"}` 。
  - 如果 `value` 是数值类型，直接写数据，不能使用双引号（ `""` ）将 `value` 引起来。
  - 如果 `value` 是 `Boolean` 类型，直接写 `TRUE` 或者 `FALSE` ，不能写成 `1` 或者 `0` 。
  - 如果 `value` 是 `Null` ，直接写 `Null` 。
  - 同一个 `key` 支持不同类型的 `value` ，查询时会返回指定类型的结果。

    例如，执行 `INSERT INTO test_tbl VALUES ({"id": 1})` 语句时，表示插入的 `id` 值是数字 `1` ；而执行 `INSERT INTO test_tbl VALUES ({"id": "1"})` 语句时，表示插入的 `id` 值是字符串 `"1"` 。

    如果执行 `SELECT id FROM test_tbl WHERE json_extract(col, '$.id')= 1;` 语句进行查询时，将返回数字 `"id": 1` ；而执行 `SELECT id FROM test_tbl WHERE json_extract(col, '$.id')= '1';` 语句进行查询时，将返回字符串 `"id": "1"` 。

写入数据

向表中写入数据时，JSON类型字段的写入方式与VARCHAR类型字段的写入方式相同，即在JSON串两端使用单引号引起来。

以下SQL示例包含多种JSON数据格式，供您参考使用。

```
INSERT INTO json_test VALUES(0, '{"id":0, "name":"abc", "age":0}');
INSERT INTO json_test VALUES(1, '{"id":1, "name":"abc", "age":10, "gender":"f"}');
INSERT INTO json_test VALUES(2, '{"id":3, "name":"xyz", "age":30, "company":{"name":"alibaba", "place":"hangzhou"}}');
INSERT INTO json_test VALUES(3, '{"id":5, "name":"a\\"b\\"c", "age":50, "company":{"name":"alibaba", "place":"america"}}');
INSERT INTO json_test VALUES(4, '{"a":1, "b":"abc-char", "c":true}');
INSERT INTO json_test VALUES(5, '{"uname":{"first":"lily", "last":"chen"}, "addr":[{"city":"beijing", "no":1}, {"city":"shenzhen", "no":0}]
, "age":10, "male":true, "like":"fish", "hobby":["basketball", "football"]}');
```

查询数据

查询数据时，AnalyticDB MySQL版支持使用函数 `json_extract` 。

- **语法**

```
json_extract(json, jsonpath)
```

- **命令说明**

  从JSON中返回 `jsonpath` 指定的值。

- **参数说明**
  - `json` 为JSON列的列名。
  - `jsonpath` ，通过点号（ `.` ）进行分割的JSON属性键 `key` 的路径，其中 `$` 表示最外层的路径。

  更多JSON函数用法请参见JSON函数。

- **示例**
  - **基本查询**

```
SELECT json_extract(vj,'$.name') FROM json_test WHERE id=1;
```

  - **等值查询**

```
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.name') = 'abc';
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.c') = true;
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.age') = 30;
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.company.name') = 'alibaba';
```

  - **范围查询**

```
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.age') > 0;
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.age') < 100;
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.name') > 'a' and json_extract(vj, '$.name') < 'z';
```

  - **IS NULL或IS NOT NULL查询**

```
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.remark') is null;
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.name') is not null;
```

  - **IN 查询**

```
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.name') in ('abc','xyz');
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.age') in (10,20);
```

  - **LIKE查询**

```
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.name') like 'ab%';
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.name') like '%bc%';
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.name') like '%bc';
```

  - **ARRAY查询**

```
SELECT id, vj FROM json_test WHERE json_extract(vj, '$.addr[0].city') = 'beijing' and json_extract(vj, '$.addr[1].no') = 0;
```

    说明 查询ARRAY数据时，支持下标取值，暂不支持遍历整个数组。

3.3. 保留字

本文为您介绍AnalyticDB MySQL版 3.0中的所有保留字。

注意 保留字不区分大小写。

字母序	保留字
A	ALL, ALTER, AND, ANY, ARRAY, AS, ASC
B	BEGIN, BETWEEN, BIGINT, BINARY, BLOB, BOOLEAN, BY, BYTES
C	CASE, CAST, CHAR, CHECK, COLUMN, COMPUTE, CONDITION, CONSTRAINT, CONTINUE, CREATE, CURRENT, CURSOR

字母序	保留字
D	DATABASE，DATE，DATETIME，DECIMAL，DECLARE，DEFAULT，DELETE，DESC，DESCRIBE，DISTINCT，DIV，DOUBLE，DROP，DUAL，DUPLICATE
E	ELSE，ESCAPE，EXCEPT，EXISTS，EXPLAIN，EXTRACT
F	FALSE，FETCH，FLOAT，FOR，FOREIGN，FROM，FULL，FULLTEXT
G	GEO2D，GRANT，GROUP，GROUPING，GROUP_CONCAT
H	HAVING
I	IDENTIFIED，IF，IGNORE，IN，INDEX，INNER，INSERT，INT，INTEGER，INTERSECT，INTERVAL，INTO，IS，ITERATE
J	JOIN
K	KEY，KILL
L	LEAVE，LEFT，LIKE，LIMIT，LOCK，LONGTEXT，LOOP
M	MATCH，MEDIUMINT，MEDIUMTEXT，MERGE，MINUS，MULTIVALUE
N	NATURAL，NOT，NULL
O	ON，OPEN，OR，ORDER，OUT，OUTER，OVER，OVERWRITE
P	PRIMARY，PROCEDURE
Q	无
R	RANGE，REAL，REFERENCES，REPEAT，REPLACE，REVOKE，RIGHT，RLIKE，ROW
S	SELECT，SEPARATOR，SET，SHOW，SMALLINT，SOME，SQLSTATE，STRING，SYSTEM
T	TABLE，TEXT，THEN，TIME，TIMESTAMP，TINYINT，TINYTEXT，TO，TRIGGER，TRUE，TRUNCATE
U	UNDO，UNION，UNIQUE，UNTIL，UPDATE，USE，USING
V	VALUE，VALUES，VARCHAR，VIEW
W	WHEN，WHERE，WHILE，WITH，WITHIN
X	XOR
Y	无
Z	无
_	_BINARY

3.4. SQL偏移表

3.4.1. DDL差异

MySQL	云原生数据仓库AnalyticDB MySQL	定义
ALTER DATABASE	不支持	修改数据库属性。
ALTER EVENT	不支持	修改现有事件的一个或多个特征。
ALTER FUNCTION	不支持	修改函数的定义。
ALTER LOGFILE GROUP	不支持	修改日志文件组。
ALTER PROCEDURE	不支持	修改存储过程特征。
ALTER SERVER	不支持	修改服务器信息。

MySQL	云原生数据仓库AnalyticDB MySQL	定义
ALTER TABLE	支持，详情请参见ALTER TABLE。 **说明** 暂不支持： • 修改表/列option • 多列alter • 新增约束	修改表的定义。
ALTER TABLE Partition	支持	修改表的分区。
ALTER TABLESPACE	不支持	修改表空间，添加新数据文件或从表空间删除数据文件。
ALTER VIEW	支持	修改视图的定义，该视图必须存在。
CREATE DATABASE	支持，详情请参见CREATE DATABASE。 **说明** 云原生数据仓库AnalyticDB MySQL中会忽略CHARACTER SET及COLLATE的值。	创建一个新的数据库。
CREATE EVENT	不支持	定义一个新的事件。
CREATE FUNCTION	不支持	定义一个新的函数。
CREATE INDEX	支持，详情请参见新增索引。 **说明** 云原生数据仓库AnalyticDB MySQL建表时默认创建全列索引 `index_all='Y'` ，不支持创建多列索引、唯一索引和空间索引，同时会忽略[index_option][algorithm_option | lock_option]。	定义一个新的索引。
CREATE LOGFILE GROUP	不支持	定义一个新的日志文件组或将文件添加到现有的日志文件组。
CREATE PROCEDURE	不支持	定义存储过程。
CREATE SERVER	不支持	定义一个用于存储引擎的服务器。
CREATE TABLE	支持，详情请参见CREATE TABLE。	创建表。
CREATE TEMPORARY TABLE	不支持	创建临时表
CREATE TABLE ... LIKE	支持	使用LIKE语法创建表。
CREATE TABLE ... SELECT	支持，详情请参见CTAS。	使用SELECT语法创建表。
FOREIGN KEY Constraints	不支持	外键约束。
CREATE TABLESPACE	不支持	定义表空间。
CREATE TRIGGER	不支持	定义触发器。
CREATE VIEW	支持，详情请参见CREATE VIEW。	定义新的视图。
DROP DATABASE	支持，详情请参见DROP DATABASE。 **说明** 为了保护数据，在云原生数据仓库AnalyticDB MySQL中删除数据库前，您需要先删除目标库下的表。	删除数据库。
DROP EVENT	不支持	删除事件。
DROP FUNCTION	不支持	删除函数。
DROP INDEX	不支持	删除索引。
DROP LOGFILE GROUP	不支持	删除日志文件组。
DROP PROCEDURE	不支持	删除存储过程。

MySQL	云原生数据仓库AnalyticDB MySQL	定义
DROP SERVER	不支持	删除服务器。
DROP TABLE	支持	删除一个或多个表。
DROP TABLESPACE	不支持	删除表空间。
DROP TRIGGER	不支持	删除触发器。
DROP VIEW	支持，详情请参见DROP VIEW。	删除一个或多个视图。
RENAME TABLE	支持	重命名一个或多个表。
TRUNCATE TABLE	支持	清空表的所有行。

3.4.2. DML差异

MySQL		云原生数据仓库AnalyticDB MySQL	定义
CALL		不支持	调用存储过程。
DELETE		支持，详情请参见DELETE。	从表中删除行。
DO		不支持	执行表达式，但不返回任何结果。
HANDLER		不支持	直接访问表存储引擎接口。
INSERT		支持，详情请参见INSERT INTO。	将新行插入到现有表中。
INSERT ... SELECT		支持，详情请参见INSERT SELECT FROM。	根据SELECT的结果将许多行快速插入到表中，可以从一个或多个表中进行选择。
INSERT ... ON DUPLICATE KEY UPDATE		支持，详情请参见INSERT ON DUPLICATE KEY UPDATE。 说明 • 目前仅3.1.3.5及之后版本的集群支持。 • 不支持更新主键列。	如果指定ON DUPLICATE KEY UPDATE子句，并且要插入的行将导致唯一索引或主键中的值重复，则会发生旧行的UPDATE。
INSERT DELAYED		不支持	DELAYED子句是标准SQL的MySQL扩展。
LOAD DATA		支持	以非常高的速度将文本文件中的行读取到表中。
LOAD XML		不支持	将数据从XML文件读取到表中。
REPLACE		支持，详情请参见REPLACE INTO。	REPLACE的运行与INSERT完全相同，不同之处在于，如果表中的旧行与PRIMARY KEY或UNIQUE索引的新行具有相同的值，则在插入新行之前删除该旧行。
SELECT，详情请参见语法。	SELECT ... INTO	不支持	使查询结果可以存储在变量中或写入文件。
	JOIN	支持，详情请参见JOIN。	连接两个子查询。
	UNION	支持，详情请参见UNION、INTERSECT和EXCEPT。	将来自多个SELECT语句的结果合并为一个结果集。
子查询，详情请参见子查询。	The Subquery as Scalar Operand	支持	等号操作行符的标量子查询。
	Comparisons Using Subqueries	支持	带有比较运算符的子查询。
	Subqueries with ANY, IN, or SOME	支持	带有ANY，IN或SOME的子查询。
	Subqueries with ALL	支持	带有ALL的子查询。
	Row Subqueries	支持	行子查询。
	Subqueries with EXISTS or NOT EXISTS	支持	带有EXISTS或NOT EXISTS的子查询。
	Correlated Subqueries	支持	子查询中包含对外层查询表的引用。
	Derived Tables	支持	在FROM子句中的子查询。

MySQL		云原生数据仓库AnalyticDB MySQL	定义
UPDATE		支持，详情请参见UPDATE。 说明 • 暂不支持更新主键列。 • 暂不支持批量更新多条SQL语句。	修改表中的数据。

3.4.3. 其他SQL功能差异

AnalyticDB MySQL不支持的MySQL 5.6功能

- 数据库管理（Database Administration）
  - 插件和自定义函数（Plugin and User-Defined Function）
  - SET
- 复合查询（Compound）
- 数据备份（Replication）
- 存储对象（Stored Objects）
- 交易和锁定（Transactional and Locking）
- 用户自定义函数（User-Defined Functions）

字符集与字符序

云原生数据仓库AnalyticDB MySQL暂时只支持一种utf8：

```
mysql> SHOW CHARACTER SET;
+---------+---------------+-------------------+--------+
| Charset | Description   | Default collation | Maxlen |
+---------+---------------+-------------------+--------+
| utf8    | UTF-8 Unicode | utf8_general_ci   | 3      |
+---------+---------------+-------------------+--------+
1 row in set (0.00 sec)
```

```
mysql> SHOW COLLATION;
+-----------------+---------+------+---------+----------+---------+
| Collation       | Charset | Id   | Default | Compiled | Sortlen |
+-----------------+---------+------+---------+----------+---------+
| utf8_general_ci | utf8    |   33 | Yes     | Yes      | 1       |
| binary          | binary  |   63 | Yes     | Yes      | 1       |
+-----------------+---------+------+---------+----------+---------+
2 rows in set (0.00 sec)
```

大小写

云原生数据仓库AnalyticDB MySQL的数据（Data）严格区分大小写，MySQL不区分大小写。

数据库管理

分类	MySQL	云原生数据仓库AnalyticDB MySQL	定义
账号管理	ALTER USER	不支持	修改用户。
	CREATE USER	支持，详情请参见CREATE USER。	创建新的用户。
	DROP USER	支持，详情请参见DROP USER。	删除一个或多个用户及其权限。
	GRANT	支持，详情请参见GRANT。	将某项权限授权给用户。
	RENAME USER	支持，详情请参见RENAME USER。	重命名现有的用户。
	REVOKE	支持，详情请参见REVOKE。	使系统管理员可以撤消用户的权限。
表维护	ANALYZE TABLE	支持，收集的统计信息不同，用户不感知差别。	键分布分析，并存储命名表的分布。
	CHECK TABLE	不支持	检查一个或多个表是否有错误。
	CHECKSUM TABLE	不支持	返回表内容的校验和。
	OPTIMIZE TABLE	支持	重新组织表数据和关联索引数据的物理存储，以减少存储空间并提高访问表时的I/O效率。

分类	MySQL	云原生数据仓库AnalyticDB MySQL	定义
	REPAIR TABLE	不支持	修复一个可能损坏的表，仅针对特定的存储引擎。
SHOW	SHOW AUTHORS	不支持	显示开发者信息。
	SHOW BINARY LOGS	不支持	显示服务器上的二进制日志文件。
	SHOW BINLOG EVENTS	不支持	显示二进制日志中的事件。
	SHOW CHARACTER	支持	显示所有可用的字符集。
	SHOW COLLATION	支持	显示服务器支持的排序规则。
	SHOW COLUMNS FROM	支持 说明 AnalyticDB MySQL 3.0不支持筛选条件，例如show columns from xxx，这个语句的执行结果会显示所有列。	显示有关给定表中列的信息。 它也适用于视图。
	SHOW CONTRIBUTORS	不支持	显示参与者。
	SHOW CREATE DATABASE	不支持	显示创建数据库语句。
	SHOW CREATE EVENT	不支持	显示创建事件语句。
	SHOW CREATE FUNCTION	不支持	显示创建函数语句。
	SHOW CREATE PROCEDURE	不支持	显示创建存储过程语句。
	SHOW CREATE TABLE	支持	显示创建表语句。
	SHOW CREATE TRIGGER	不支持	显示创建触发器语句。
	SHOW CREATE VIEW	支持	显示创建视图语句。
	SHOW DATABASES	支持	显示服务器上的数据库。
	SHOW ENGINE	不支持	显示存储引擎的日志或状态信息。
	SHOW ENGINES	不支持	显示服务器当前支持使用的存储引擎。
	SHOW ERRORS	不支持	显示最后一个执行语句所产生的错误信息。
	SHOW EVENTS	不支持	显示关于默认数据库里的事件信息。
	SHOW FUNCTION CODE	不支持	服务器内部调试，显示一个指定存储的内部实现的表示形式过程。
	SHOW FUNCTION STATUS	不支持	显示存储函数信息（需要先创建存储函数）。
	SHOW GRANTS	支持	显示指定用户拥有的访问权限。
	SHOW INDEX	支持	显示指定数据库的索引信息。
	SHOW MASTER STATUS	不支持	显示master当前正在使用的二进制信息。
	SHOW OPEN TABLES	不支持	显示一份已在数据表缓存里注册并处于打开状态的非临时数据表清单。
	SHOW PLUGINS	不支持	显示插件信息。
	SHOW PRIVILEGES	不支持	显示可以授权的权限以及定义。
	SHOW PROCEDURE CODE	不支持	服务器内部调试，显示一个指定存储的内部实现的表示形式过程。
	SHOW PROCEDURE STATUS	不支持	显示存储过程信息（需要先创建存储过程）。
	SHOW PROCESSLIST	支持，详情请参见SHOW PROCESSLIST。	显示当前正在执行的服务器活动的信息。
	SHOW PROFILE	不支持	显示当前会话执行语句资源使用情况。
	SHOW PROFILES	不支持	显示当前会话执行语句资源使用情况。

分类	MySQL	云原生数据仓库AnalyticDB MySQL	定义
	SHOW RELAYLOG EVENTS	不支持	显示relaylog事件信息（需要先做主从复制）。
	SHOW SLAVE HOSTS	不支持	显示master主机上已注册的复制主机列表（需要先做主从复制）。
	SHOW SLAVE STATUS	不支持	显示slave主机状态信息（需要先做主从复制）。
	SHOW STATUS	不支持	显示MySQL状态信息。
	SHOW TABLE STATUS	不支持	显示表属性信息。
	SHOW TABLES	支持	显示当前数据库中所有表的名称。
	SHOW TRIGGERS	不支持	显示触发器信息（需要先创建触发器）。
	SHOW VARIABLES	不支持	显示变量信息。
	SHOW WARNINGS	不支持	显示最后一个执行语句所产生的警告信息。
其他	BINLOG	不支持	BINLOG是内部使用的语句。
	CACHE INDEX	不支持	将表索引分配给特定的键高速缓存。
	FLUSH	支持	FLUSH语句具有多种变体形式，可以清除或重新加载各种内部缓存，刷新表或获取锁。
	KILL	支持，详情请参见KILL PROCESS。	终止正在执行的进程。
	LOAD INDEX INTO CACHE	不支持	LOAD INDEX INTO CACHE语句将表索引预加载到显式CACHE INDEX语句已为其分配的键高速缓存中，否则将其预加载到默认键高速缓存中。
	RESET	不支持	用于清除各种服务器操作的状态。

公用

MySQL	云原生数据仓库AnalyticDB MySQL	定义
DESCRIBE/EXPLAIN	支持EXPLAIN	DESCRIBE和EXPLAIN语句定义相同，用于获取有关表结构的信息或查询执行计划。
HELP	不支持	从MySQL参考手册返回在线帮助信息。
USE	支持	将命名数据库用作后续语句的默认（当前）数据库。 该语句要求对数据库或其中的某些对象具有某些权限。

3.4.4. 不支持的数据类型及运算符

不支持的数据类型

分类	数据类型
数值类型	BIT
	DEC
	UNSIGNED 说明 不支持无符号数。
字符类型	TINYBLOB
	MEDIUMBLOB
	LONGBLOB
	GEOMETRY
	LINESTRING
	POLYGON

分类	数据类型
空间类型	MULTIPOINT
	MULTILINESTRING
	MULTIPOLYGON
	GEOMETRYCOLLECTION
复杂数据类型	ENUM
	SET

不支持的运算符

MySQL	定义
:=	赋值
~	按位取反

3.5. DDL

3.5.1. CREATE DATABASE

`CREATE DATABASE` 用于创建数据库。

创建数据库

> **说明** 每个集群最多可以创建256个数据库。

- **语法**

```
CREATE DATABASE [IF NOT EXISTS] db_name
```

- **参数**

`db_name`：数据库名。以小写字符开头，可包含字母、数字以及下划线（_），但不能包含连续两个及以上的下划线（_），长度不超过64个字符。

> **说明** 数据库名不能是analyticdb，analyticdb是内置数据库。

- **示例**

```
CREATE DATABASE adb_demo;
```

使用数据库

数据库创建成功后，您可以通过 `USE db_name` 命令使用数据库。

- **语法**

```
USE db_name
```

- **示例**

```
use adb_demo;
show tables;
+------------------------------+
| Tables_in_adb_demo           |
+------------------------------+
| customer                     |
| test_table                   |
```

3.5.2. CREATE TABLE

云原生数据仓库AnalyticDB MySQL版支持通过 `CREATE TABLE` 创建表，也支持通过 `CTAS` 将查询到的数据写入新表中。

语法

```
CREATE TABLE [IF NOT EXISTS] table_name
  ({column_name column_type [column_attributes] [ column_constraints ] [COMMENT 'string']
  | table_constraints}
   [, ... ]  )
   table_attribute
   [partition_options]
   [storage_policy]
   [block_size]
[AS] query_expression
   COMMENT 'string'
column_attributes:
   [DEFAULT default_expr]
   [AUTO_INCREMENT]
column_constraints:
   [{NOT NULL|NULL} ]
   [PRIMARY KEY]
table_constraints:
   [{INDEX|KEY} [index_name] (column_name,...)]
   [PRIMARY KEY [index_name] (column_name,...)]
   [CLUSTERED KEY [index_name] (column_name,...)]
table_attribute:
   DISTRIBUTED BY HASH(column_name,...) | DISTRIBUTED BY BROADCAST
partition_options:
  PARTITION BY
       {VALUE(column_name) | VALUE(date_format(column_name, ?))}
  LIFECYCLE N
storage_policy:
  STORAGE_POLICY= 'HOT'|'COLD'|'MIXED' [hot_partition_count=N]
block_size:
  BLOCK_SIZE= VALUE
```

参数

参数	说明
`table_name`	表名。 表名以字母或下划线（_）开头，可包含字母、数字以及下划线（_），长度为1到127个字符。 支持 `db_name.table_name` 格式，区分不同数据库下相同名字的表。
`column_name`	列名。 列名以字母或下划线（_）开头，可包含字母、数字以及下划线（_），长度为1到127个字符。
`column_type`	要添加的列的数据类型。 AnalyticDB MySQL版支持的数据类型，请参见基础数据类型。
`column_attributes`	• `DEFAULT default_expr`：设置列的默认值，`DEFAULT` 为无变量表达式，例如 `current_timestamp`。 如果未指定默认值，则列的默认值为 `NULL`。 • `AUTO_INCREMENT`：定义自增列，可选项。 自增列的数据类型必须是 `BIGINT` 类型，AnalyticDB MySQL版为自增列提供唯一值，但自增列的值不是顺序递增。
`column_constraints`	• `NOT NULL` | `NULL`：定义了 `NOT NULL` 的列不允许值为 `NULL`；定义了 `NULL`（默认值）的列允许值为 `NULL`。 • `PRIMARY KEY`：定义主键。 如果有多个主键，语法为 `PRIMARY KEY(column_name [, ... ])`。

参数	说明
table_constraints	说明 云原生数据仓库AnalyticDB MySQL版不支持创建唯一索引。 • INDEX|KEY：定义倒排索引。 AnalyticDB MySQL版默认为表创建全索引，一般情况下无须手动创建索引。 • PRIMARY KEY：定义主键索引。 ○ 只有定义过主键的表支持DELETE和UPDATE操作。 ○ 主键中必须包含分布键和分区键，建议将分区键和分布键放在组合主键的前部。 • CLUSTERED KEY：聚集索引，定义表中的排序列，聚集索引中键值的逻辑顺序决定了表中相应行的物理顺序。 例如，clustered key col5_col6_cls_index(col5,col6) 定义了 col5 col6 的聚集索引，col5 col6 和 col6 col5 是不同的聚集索引。 聚集索引会将该列或者多列进行排序，保证与该列相同或者相近的数据存储在磁盘的相同或相近位置。当以聚集列做为查询条件时，查询结果存储在磁盘的相同位置，这样可以减少磁盘的IO，提高查询性能。 如何判断是否需要聚集索引：查询一定会携带的字段可以作为聚集索引。例如，SAAS类应用中，用户通常只访问自己的数据，用户ID可以定义为聚集索引，保证数据的局部性，提升数据查询性能。 聚集列有以下限制： ○ 每张表中只支持创建一个聚集列索引。 ○ 由于聚集索引会进行全表排序，导致数据写入性能下降、CPU占用较高，因此一般不建议使用聚集索引。
table_attribute	• DISTRIBUTED BY HASH(column_name,...)：在普通表中定义表的分布键，按照 column_name 的HASH值进行分片。 AnalyticDB MySQL版支持将多个字段作为分布键。 AnalyticDB MySQL版不支持修改分布键。如果需要修改分布键，请参见更改分区键/分布键。 • DISTRIBUTED BY BROADCAST：定义维度表，维度表会在集群的每个节点存储一份数据，因此建议维度表的数据量不宜太大。
partition_options	PARTITION BY：普通表中定义分区。 通过 LIFECYCLE N 方式实现表生命周期管理，即对分区进行排序，超出 N 的分区将被过滤掉。 例如，PARTITION BY VALUE(column_name) 表示使用 column_name 的值来做分区，PARTITION BY VALUE(DATE_FORMAT(column_name, '%Y%m%d')) 表示将 column_name 格式化为类似 20190101 的日期格式做分区。LIFECYCLE 365 表示每个节点最多保留的分区个数为365，即如果数据保存天数为365天，则第366天写入数据后，系统会自动删除第1天写入的数据。 AnalyticDB MySQL版不支持修改分区键。如果需要修改分区键，请参见更改分区键/分布键。 说明 • 二级分区不是整张表级，而是每个Shard级别。如果数据分布不均，则会导致整张表保留的二级保存分区数大于N。 • 二级分区不是实时清理的，是后台异步任务清理的。 • 当您使用 PARTITION BY 指定二级分区时，必须定义 LIFECYCLE N，否则会报错。若不指定二级分区，数据不会被清理。
storage_policy	说明 目前仅弹性模式集群版（新版）实例支持冷热数据分层存储功能。 STORAGE_POLICY：指定热（HOT）、冷（COLD）、或混合（MIXED）的存储策略。默认值为HOT。 不同存储策略下数据读写性能不同，存储成本不同。为了降低数据存储成本，同时还要保证查询性能，您可以选择将查询频度高的数据（称为热数据）存储在SSD介质；将查询频度低的数据（称为冷数据）存储在HDD介质。 根据业务需求，您还可以按表粒度、表的二级分区粒度独立选择冷、热存储介质。例如，指定这个表数据全部存储在SSD，或者全部存储在HDD，或者指定这个表的一部分二级分区存储在SSD，另一部分二级分区存储在HDD。 • HOT、COLD、MIXED大小写兼容。 • HOT：所有分区都在SSD。 • COLD：所有分区都在HDD。 • MIXED：部分分区在SSD，部分分区在HDD，需要通过 hot_partition_count 指定存在SSD上的分区的数量。
hot_partition_count=N	指定MIXED存储策略时热分区的个数。表示按分区键的值的大小倒序排列，最大N个分区为热分区，其他分区为冷分区。 • N为非零正整数。 • 指定MIXED策略时，必须同时指定热分区的个数；其他策略禁止指定 hot_partition_count=N 。

参数	说明
block_size	指定列式存储中每个block存储的Value的个数，也是最小的IO单元，默认值取值说明： • 弹性模式集群版（新版）且计算资源为32核以下（不包括32核）的集群block_size默认值为8192。 • 维度表的block_size默认值为4096。 • 其它情况下block_size默认值为32760。当block_size为32760，在 `SHOW CREATE TABLE <table_name>;` 时，block_size不做显示。 调大或调小会使得每次IO读取的Value个数变大或变小，具体产生的影响需要结合查询特征，例如点查询时，若block_size较大，存储读block的效率会降低，此时可以适当调小block_size。 **注意** 若不熟悉列式存储原理，建议不要进行更改。

注意事项

- 创建表时，AnalyticDB MySQL版集群默认编码格式为utf-8，相当于MySQL中的utf8mb4编码，暂不支持其他编码格式。
- 目前AnalyticDB MySQL版集群支持创建的最大表数目如下所示：
  - 集群版：min（节点组数量*256,10000）。
  - 基础版：
    - T8，500。
    - T16和T32，1500。
    - T52，2500。

示例

- 新建TEST表。

```
CREATE TABLE test (
       id bigint auto_increment,
       name varchar,
       value int,
       ts timestamp
)
DISTRIBUTED BY HASH(id);
```

TEST为普通表，`id`为自增列，分布键为`id`，按照`id`值进行HASH分区。

- 新建CUSTOMER表。

```
CREATE TABLE customer (
customer_id bigint NOT NULL COMMENT '顾客ID',
customer_name varchar NOT NULL COMMENT '顾客姓名',
phone_num bigint NOT NULL COMMENT '电话',
city_name varchar NOT NULL COMMENT '所属城市',
sex int NOT NULL COMMENT '性别',
id_number varchar NOT NULL COMMENT '身份证号码',
home_address varchar NOT NULL COMMENT '家庭住址',
office_address varchar NOT NULL COMMENT '办公地址',
age int NOT NULL COMMENT '年龄',
login_time timestamp NOT NULL COMMENT '登录时间',
PRIMARY KEY (login_time,customer_id, phone_num)
 )
DISTRIBUTED BY HASH(customer_id)
PARTITION BY VALUE(DATE_FORMAT(login_time, '%Y%m%d')) LIFECYCLE 30
COMMENT '客户信息表';
```

CUSTOMER表为普通表，`customer_id`为分布键，`login_time`为分区键，`login_time`、`customer_id`、`phone_num`为组合主键。

MySQL语法兼容性说明

AnalyticDB MySQL版标准建表语法中必须包含`DISTRIBUTED BY ...`，而MySQL建表语法中没有`DISTRIBUTED BY ...`。AnalyticDB MySQL版默认兼容MySQL建表语法，您可以根据实际情况通过以下两种方式处理`DISTRIBUTED BY ...`不兼容问题。

- 如果MySQL表含有主键，AnalyticDB MySQL版默认将主键作为`DISTRIBUTED BY COLUMN`。

```
CREATE TABLE t (c1 bigint, c2 int, c3 varchar, PRIMARY KEY(c1,c2));
Query OK, 0 rows affected (2.37 sec)
SHOW CREATE TABLE t;
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table | Create Table                                                                                                                                              |
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| t     | Create Table `t` (
 `c1` bigint,
 `c2` int,
 `c3` varchar,
 primary key (c1,c2)
) DISTRIBUTED BY HASH(`c1`,`c2`) INDEX_ALL='Y' |
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.04 sec)
```

- 如果MySQL表不含主键，AnalyticDB MySQL版将添加一个 `__adb_auto_id__` 字段作为主键和 `DISTRIBUTED BY COLUMN` 。

```
CREATE TABLE t (c1 bigint, c2 int, c3 varchar);
Query OK, 0 rows affected (0.50 sec)
SHOW CREATE TABLE t;
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| t     | Create Table `t` (
 `c1` bigint,
 `c2` int,
 `c3` varchar,
 `__adb_auto_id__` bigint AUTO_INCREMENT,
 primary key (__adb_auto_id__)
) DISTRIBUTED BY HASH(`__adb_auto_id__`) INDEX_ALL='Y' |
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
1 row in set (0.04 sec)
```

创建表时指定冷热数据存储策略

> **说明** 目前仅弹性模式集群版（新版）实例支持冷热数据分层存储功能。

您可以在创建表时指定冷热存储策略。

```
CREATE TABLE [IF NOT EXISTS] table_name
  ({column_name column_type [column_attributes] [ column_constraints ] [COMMENT 'string']
  | table_constraints}
   [, ... ]  )
   table_attribute
   [partition_options]
   [storage_policy]
   [AS] query_expression
   COMMENT 'string'
storage_policy:
   STORAGE_POLICY= 'HOT'|'COLD'|'MIXED' [hot_partition_count=N]
```

示例：创建表时指定冷（COLD）存储策略

```
CREATE TABLE test_table (
 L_ORDERKEY bigint NOT NULL,
 L_LINENUMBER int NOT NULL,
 L_SHIPDATE date NOT NULL,
 dummy varchar,
 primary key (l_orderkey,l_linenumber,l_shipdate)
) DISTRIBUTE BY HASH(l_orderkey)
PARTITION BY VALUE(date_format(l_shipdate, '%Y%m')) LIFECYCLE 200 INDEX_ALL='Y'
STORAGE_POLICY='COLD';
```

示例：创建表时指定热（HOT）存储策略

```
CREATE TABLE test_table (
 L_ORDERKEY bigint NOT NULL,
 L_LINENUMBER int NOT NULL,
 L_SHIPDATE date NOT NULL,
 dummy varchar,
 primary key (l_orderkey,l_linenumber,l_shipdate)
) DISTRIBUTE BY HASH(l_orderkey)
PARTITION BY VALUE(date_format(l_shipdate, '%Y%m')) LIFECYCLE 200 INDEX_ALL='Y'
STORAGE_POLICY='HOT';
```

示例：创建表时指定混合（MIXED）存储策略，同时指定热分区数量为16个

```
CREATE TABLE test_table (
 L_ORDERKEY bigint NOT NULL,
 L_LINENUMBER int NOT NULL,
 L_SHIPDATE date NOT NULL,
 dummy varchar,
 primary key (l_orderkey,l_linenumber,l_shipdate)
) DISTRIBUTE BY HASH(l_orderkey)
PARTITION BY VALUE(date_format(l_shipdate, '%Y%m')) LIFECYCLE 200 INDEX_ALL='Y'
STORAGE_POLICY='MIXED' HOT_PARTITION_COUNT=16;
```

创建带向量索引的表

您可以在创建表时同步创建ann索引。定义为：

```
ann index [index_name] (col_name,...)] [algorithm=HNSW_PQ ] [dis_function=SquaredL2]
```

示例：

```
CREATE TABLE fact_tb (
xid bigint not null,
cid bigint not null,
uid varchar not null,
vid varchar not null,
wid varchar not null,
short_feature array < smallint >(32),
float_feature array < float >(32),
ann index short_feature_index(short_feature),
ann index float_feature_index(float_feature),
PRIMARY KEY (xid, cid, vid)
) DISTRIBUTE BY HASH(xid) PARTITION BY VALUE(cid) LIFECYCLE 4;
```

参数说明：

- `short_feature/float_feature`：向量列的名称，用户自定义。
- `array<float>(32)`：向量列的数据类型和向量的维数，用户自定义。必须要指定向量的维度，支持的类型包括： float、byte、short三种数组类型。表示对feature_data的列类型定义为512，示例如下：

  ```
  feature_data` array<float>(512)
  ```

- `ann`：系统关键字。
- `index`：系统关键字。
- `short_feature_index/float_feature_index`：索引名，用户自定义。对feature_data这一列创建向量索引：

  ```
  ann index ecnn_index(`FEATURE_DATA`) algorithm=HNSW_PQ dis_function=SquaredL2
  ```

  algorithm、dis_function定义可以省略，默认值分别为HNSW_PQ、SquaredL2。
- `algorithm`：向量距离计算公式使用的算法。AnalyticDB MySQL版支持的向量距离计算公式算法如下表所示。

算法	适用场景	适用数据类型
HNSW_PQ	适用于单表数据量在百万级别到千万级别之间，对向量维度敏感的中等规模数据量场景。	short[]、byte[]、float[]

- `dis_function`：向量距离计算公式，默认值为SquaredL2。AnalyticDB MySQL版支持的向量距离计算公式如下表所示。

距离计算公式	计算公式	适用数据类型
SquaredEuclidean（简称SquaredL2）	(x1-y1)^2+(x2-y2)^2+...	byte[]、short[]或者float[]

使用向量检索查询

1. 查询与向量 `'[1,1,1,1]'` 距离最近的5条记录：

   ```
   SELECT id, l2_distance(short_feature, '[1,1,1,1]') FROM fact_tb ORDER BY 2 LIMIT 5;
   ```

2. 带向量索引查询与向量 `'[1,1,1,1]'` 距离最近的5条记录，按距离排序：

   ```
   SELECT id, l2_distance(short_feature, '[1,1,1,1]') FROM fact_tb WHERE xid = 1 AND cid = 0 ORDER BY 2 LIMIT 5;
   ```

3. 带向量索引查询与向量 `'[1,1,1,1]'` 距离最近的5条记录，按距离排序，并控制距离最大范围：

   ```
   SELECT id, l2_distance(float_feature, '[1.2,1.5,2,3.0]') FROM fact_tb WHERE l2_distance(float_feature, '[1.2,1.5,2,3.0]') < 50.0 AND xi
   d = 2 ORDER BY 2 LIMIT 5;
   ```

3.5.3. CREATE RESOURCE GROUP

ADB弹性模式集群版（新版）支持通过 `CREATE RESOURCE GROUP` 创建资源池，资源池的资源粒度为计算节点个数。

语法

```
CREATE RESOURCE GROUP resource_group_name
    [QUERY_TYPE = {interactive, batch, default_type}]
    [NODE_NUM = N]
```

- 资源池名 `resource_group_name` 不可以与默认资源池 `user_default` 同名。
- 资源池名大小写不敏感，例如test_group和Test_Group是同名资源池，资源池名展示为大写TEST_GROUP。

参数

参数	说明
QUERY_TYPE	查询类型。可选值为： • interactive • batch • default_type 默认值为default_type。详情请参见查询执行模式（数仓版）。
NODE_NUM	节点个数。指定资源池的节点个数，默认值为0。

示例

1. 新建资源池。
   - 创建单个计算节点的资源池。
     ```
     CREATE RESOURCE GROUP 'BATCH_RG' NODE_NUM=1;
     ```
   - 创建查询执行模式为batch的资源池。
     ```
     CREATE RESOURCE GROUP 'BATCH_RG' QUERY_TYPE=batch NODE_NUM=1;
     ```
2. 查看当前资源池列表。
   ```
   SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS;
   ```

3.5.4. CTAS

云原生数据仓库AnalyticDB MySQL版支持通过 `CREATE TABLE` 创建表，也支持通过 `CTAS` 将查询到的数据写入新表中。

语法

```
CREATE TABLE [IF NOT EXISTS] <table_name> [table_definition]
[IGNORE|REPLACE] [AS] <query_statement>
```

> **说明** 该建表方式默认与CREATE TABLE一致，支持语法也相同，例如默认为表创建全索引等。

参数

参数	说明
`table_name`	表名。 表名以字母或下划线（_）开头，可包含字母、数字以及下划线（_），长度为1~127个字符。 支持 `db_name.table_name` 格式，区分不同数据库下相同名字的表。
`IF NOT EXISTS`	判断 `table_name` 指定的表是否存在，若存在，则不执行建表语句。
`IGNORE`	可选参数，若表中已有相同主键的记录，新记录不会被写入。
`REPLACE`	可选参数，若表中已有相同主键的记录，新记录将替换已有相同主键的记录。

示例

本示例根据 `customer` 表，创建一个新的表 `new_customer` 。 `customer` 表结构如下：

```
CREATE TABLE customer (
customer_id bigint NOT NULL COMMENT '顾客ID',
customer_name varchar NOT NULL COMMENT '顾客姓名',
phone_num bigint NOT NULL COMMENT '电话',
city_name varchar NOT NULL COMMENT '所属城市',
sex int NOT NULL COMMENT '性别',
id_number varchar NOT NULL COMMENT '身份证号码',
home_address varchar NOT NULL COMMENT '家庭住址',
office_address varchar NOT NULL COMMENT '办公地址',
age int NOT NULL COMMENT '年龄',
login_time timestamp NOT NULL COMMENT '登录时间',
PRIMARY KEY (login_time,customer_id,phone_num)
 )
DISTRIBUTED BY HASH(customer_id)
PARTITION BY VALUE(DATE_FORMAT(login_time, '%Y%m%d')) LIFECYCLE 30
COMMENT '客户信息表';
```

示例1

从 `customer` 表读取所有列数据，并将数据写入新表 `new_customer` 。新表 `new_customer` 的主键、分布键、分区键、列数据类型、默认值与 `customer` 表相同。示例语句如下：

```
CREATE TABLE new_customer
AS
SELECT * FROM customer;
```

示例2

从 `customer` 表读取 `customer_id` 、 `customer_name` 列的数据，并将数据写入新表 `new_customer` 。示例语句如下：

```
CREATE TABLE new_customer
AS
SELECT customer_id, customer_name
FROM customer;
```

示例3

从 `customer` 表读取 `customer_id` 、 `login_time` 列的数据，并将数据写入新表 `new_customer` 。定义新表 `customer_id` 、 `login_time` 为主键， `customer_id` 为分布键。示例语句如下：

```
CREATE TABLE new_customer (
  PRIMARY KEY (customer_id,login_time))
  DISTRIBUTE BY HASH (customer_id)
AS
SELECT customer_id, login_time
FROM customer;
```

示例4

从 `customer` 表读取 `customer_id` 、 `login_time` 列的数据，并将数据写入新表 `new_customer` 。定义新表 `customer_id` 、 `login_time` 为主键， `customer_id` 为分布键，重新定义 `login_time` 列类型。示例语句如下：

```
CREATE TABLE new_customer (
  login_time date,
  PRIMARY KEY (customer_id,login_time))
  DISTRIBUTE BY HASH (customer_id)
AS
SELECT customer_id, login_time
FROM customer;
```

示例5

从 `customer` 表读取 `customer_id` 、 `customer_name` 、 `login_time` 列的数据，并将数据写入新表 `new_customer` 。定义新表倒排索引为 `customer_id` 。示例语句如下：

```
CREATE TABLE new_customer (
  INDEX a_idx (customer_id))
AS
SELECT customer_id, customer_name, login_time
FROM customer;
```

3.5.5. ALTER TABLE

云原生数据仓库AnalyticDB MySQL版支持通过 `ALTER TABLE` 修改表。本文介绍 `ALTER TABLE` 语法。

语法

```
ALTER TABLE table_name
  {
    ADD [COLUMN] (column_name column_definition,...)
  | ADD {INDEX|KEY} [index_name] (column_name,...)
  | ADD CLUSTERED [INDEX|KEY] [index_name] (column_name,...)
  | DROP [COLUMN] column_name
  | DROP {INDEX|KEY} index_name
  | DROP CLUSTERED [INDEX|KEY] index_name
  | MODIFY [COLUMN] column_name column_definition
  | RENAME COLUMN column_name to new_column_name
  | RENAME new_table_name
  | TRUNCATE PARTITION {partition_names | ALL}
  | STORAGE_POLICY= {'HOT'|'COLD'|'MIXED' [hot_partition_count=N]}}
```

增加列

- **语法**

```
ALTER TABLE db_name.table_name ADD [COLUMN] column_name data_type;
```

- **示例**

在CUSTOMER表中增加一列province，数据类型为VARCHAR。

```
ALTER TABLE adb_demo.customer ADD COLUMN province varchar comment '省份';
```

删除列

- **语法**

```
ALTER TABLE db_name.table_name DROP [COLUMN] column_name data_type;
```

- **示例**

在CUSTOMER表中删除类型为VARCHAR的province列。

```
ALTER TABLE adb_demo.customer DROP COLUMN province;
```

更改COMMENT

- **语法**

```
ALTER TABLE db_name.table_name MODIFY [COLUMN] column_name data_type comment 'new_comment';
```

- **示例**

将CUSTOMER表中province列的COMMENT更改为顾客所属省份。

```
ALTER TABLE adb_demo.customer MODIFY COLUMN province varchar comment '顾客所属省份';
```

设置NULL

> **说明** 仅支持将NOT NULL变更为NULL。

- **语法**

```
ALTER TABLE db_name.table_name MODIFY [COLUMN] column_name data_type {NULL};
```

- **示例**

将CUSTOMER表中province列的值更改为可空（NULL）。

```
ALTER TABLE adb_demo.customer MODIFY COLUMN province varchar NULL;
```

更改DEFAULT值

- **语法**

```
ALTER TABLE db_name.table_name MODIFY [COLUMN] column_name data_type DEFAULT 'default';
```

- **示例**

将CUSTOMER表中性别sex的默认值设置为0（性别为男）。

```
ALTER TABLE adb_demo.customer MODIFY COLUMN sex int(11) NOT NULL DEFAULT 0;
```

更改列类型

- **语法**

```
ALTER TABLE db_name.table_name MODIFY [COLUMN] column_name new_data_type;
```

- **注意事项**

仅支持整型数据类型之间，以及浮点数据类型之间的列类型更改，并且只能将取值范围小的数据类型更改为取值范围大的数据类型，或者将单精度数据类型更改为双精度数据类型。

- 整型数据类型：支持TINYINT、SMALLINT、INT、BIGINT间，小类型到大类型的更改，例如支持将TINYINT更改为BIGINT，不支持将BIGINT更改为TINYINT。
- 浮点数据类型：支持将FLOAT更改为DOUBLE类型，不支持将DOUBLE更改为FLOAT类型。

- **示例**

以TEST表为例，order_number列原本为INT类型，建表语句如下。

```
CREATE TABLE adb_demo.test(id int, order_number int NOT NULL DEFAULT 100, name varchar) DISTRIBUTE BY HASH(id);
```

将TEST表中order_number列由INT类型更改为BIGINT类型。

```
ALTER TABLE adb_demo.test MODIFY COLUMN order_number BIGINT NOT NULL DEFAULT 100;
```

新增索引

AnalyticDB MySQL建表时默认创建全列索引 `index_all='Y'` 。若建表时未创建全列索引，可以通过以下方式新增索引。变更索引后需要Build，Build的详细信息，请参见Build任务。

> **说明** AnalyticDB MySQL不支持创建唯一索引。

- **语法**

```
ALTER TABLE db_name.table_name ADD KEY index_name(column_name);
```

- **示例**

在CUSTOMER表中为age列新增索引。

```
ALTER TABLE adb_demo.customer ADD KEY age_idx(age);
```

删除索引

- **语法**

```
ALTER TABLE db_name.table_name DROP KEY index_name;
```

- **参数说明**

可以通过以下命令获取 `index_name` 。

```
SHOW INDEXES FROM db_name.table_name;
```

- **示例**

删除CUSTOMER表中age列的索引。

```
ALTER TABLE adb_demo.customer DROP KEY age_idx;
```

更改表名

- **语法**

```
ALTER TABLE db_name.table_name RENAME new_table_name;
```

- **示例**

将customer表更名为new_customer。

```
ALTER TABLE customer RENAME new_customer;
```

更改列名

> **说明** 不支持更改主键列的列名。

- **语法**

```
ALTER TABLE db_name.table_name RENAME COLUMN column_name to new_column_name;
```

- **示例**

将customer表中的age列更名为new_age。

```
ALTER TABLE customer RENAME COLUMN age to new_age;
```

更改表的生命周期

- **语法**

```
ALTER TABLE db_name.table_name PARTITIONS N;
```

- **示例**

以customer表为例，原本的生命周期为30，建表语句如下。

```
CREATE TABLE customer (
customer_id bigint NOT NULL COMMENT '顾客ID',
customer_name varchar NOT NULL COMMENT '顾客姓名',
phone_num bigint NOT NULL COMMENT '电话',
city_name varchar NOT NULL COMMENT '所属城市',
sex int NOT NULL COMMENT '性别',
id_number varchar NOT NULL COMMENT '身份证号码',
home_address varchar NOT NULL COMMENT '家庭住址',
office_address varchar NOT NULL COMMENT '办公地址',
age int NOT NULL COMMENT '年龄',
login_time timestamp NOT NULL COMMENT '登录时间',
PRIMARY KEY (login_time,customer_id, phone_num)
 )
DISTRIBUTED BY HASH(customer_id)
PARTITION BY VALUE(DATE_FORMAT(login_time, '%Y%m%d')) LIFECYCLE 30
COMMENT '客户信息表';
```

将customer表的生命周期由30改为40。语句如下。

```
ALTER TABLE customer PARTITIONS 40;
```

更改表的冷热数据存储策略

> **说明** 目前仅弹性模式集群版（新版）支持冷热数据分层存储功能。

您可以执行ALTER TABLE语句更改表的冷热数据存储属性。

```
ALTER TABLE table_name storage_policy;
storage_policy:
   STORAGE_POLICY= {'HOT'|'COLD'|'MIXED' [hot_partition_count=N]}
```

COLD、HOT、MIXED三种策略之间可以任意转换。

执行ALTER TABLE语句后，存储策略不会立即变更。如需立即变更，可以手动执行 `BUILD TABLE db_name.table_name` 。

在创建表时指定冷热存储策略的方法，请参见CREATE TABLE。

示例：更改表的存储策略为COLD

```
ALTER TABLE test_table storage_policy = 'COLD';
```

示例：更改表的存储策略为HOT

```
ALTER TABLE test_table storage_policy = 'HOT';
```

示例：更改表的存储策略为MIXED，其中热分区的个数为10个

```
ALTER TABLE test_table storage_policy = 'MIXED' hot_partition_count = 10;
```

更改分区键/分布键

AnalyticDB MySQL集群不支持更改分区键和分布键。如果您的业务必须更改分区键或分布键，可通过以下方案解决。

假设您有一个表order需要将现有分布键order_id更改为customer_id，操作如下：

1. 使用分布键customer_id创建一个临时表order_auto_opt_v1。

```
CREATE TABLE order_auto_opt_v1 (
  order_id bigint NOT NULL COMMENT '订单ID',
  customer_id bigint NOT NULL COMMENT '顾客ID',
  customer_name varchar NOT NULL COMMENT '顾客姓名',
  order_time timestamp NOT NULL COMMENT '订单时间',
  --省略其他字段
  PRIMARY KEY (order_id, customer_id) --分布键customer_id需要添加到主键中
)
DISTRIBUTED BY HASH(customer_id) --修改order_id为customer_id
PARTITION BY VALUE(DATE_FORMAT(order_time, '%Y%m%d')) LIFECYCLE 90 --二级分区保持不变
COMMENT '订单信息表';
```

2. 使用 `INSERT OVERWRITE INTO SELECT` 将源表的数据导入到临时表，详情请参见INSERT OVERWRITE INTO SELECT。

```
INSERT OVERWRITE INTO order_auto_opt_v1
SELECT * FROM order;
```

3. 判断分布键是否合理。数据导入后，需要判断新的分布键是否有数据倾斜问题，详情请参见分布字段合理性诊断。
4. 使用 `RENAME TABLE <源表表名> to <new_源表表名>;` 更改源表表名。

```
RENAME TABLE order to order_backup; --数据导入完成后，重命名源表做为备份
```

5. 使用 `RENAME TABLE <临时表表名> to <源表表名>;` 将临时表表名更改为源表表名。

```
RENAME TABLE order_auto_opt_v1 to order;
```

3.5.6. CREATE VIEW

本文介绍如何使用 `CREATE VIEW` 语法创建视图。

语法

```
CREATE
[OR REPLACE]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name
AS select_statement;
```

参数	是否必填	说明
`OR REPLACE`	选填	根据是否存在重名视图，选择对应的规则来创建视图。具体规则如下： • 若不存在重名视图，AnalyticDB MySQL版会直接创建一个新视图。 • 若存在重名视图，AnalyticDB MySQL版会先删除原有的重名视图，再重新创建。 **说明** 若未设置该参数，当新建的视图名称与已存在视图重名时，会创建失败。
`[SQL SECURITY]`		定义查询视图的数据时的安全验证方式，支持如下取值： • `INVOKER`：表示按照 `INVOKER`（调用者）的身份来执行查询SQL。 该安全验证方式下，系统会在用户查询视图数据时，验证用户是否拥有如下权限： ○ 查询视图的权限。 ○ 查询视图所引用的对象的权限。 仅当用户同时拥有上述权限时，才能查询视图的数据。 • `DEFINER`：表示按照 `DEFINER`（定义者）的身份来执行查询SQL。 该安全验证方式下，系统会在用户查询视图数据时，验证用户是否拥有如下权限： ○ 查询者只需拥有查询视图的权限。 ○ 定义者需要拥有查询视图所引用对象的权限。 如果定义者被撤权后，即使查询者仍拥有视图查询权限，仍然会出现无法查询视图的问题。 **说明** • 若未设置该参数，AnalyticDB MySQL版默认使用 `INVOKER` 安全验证方式，即用户查询视图数据时，需要同时拥有查询视图的权限和查询视图所引用对象的权限。 • 需为仅V3.1.4.0或以上版本的AnalyticDB MySQL版集群支持该配置。 如何查看集群版本，请参见查看版本。 如需升级版本，请提交工单联系技术支持。
`view_name`	必填	视图的名字。 **说明** 您也可以在视图名字前加上数据库名称来定义该视图所属的数据库，例如 `adb_demo.view`。若不添加，默认该视图属于当前数据库。
`select_statement`		视图中的数据来源。

示例

- **数据准备**

  通过AnalyticDB MySQL版高权限账号执行如下操作：

  i. 创建账号 `user1`，语句如下：

  ```
  CREATE USER user1 IDENTIFIED BY 'user1_pwd';
  ```

  ii. 已创建数据库 `adb_demo`，并在库中创建表 `t1`，建表语句如下：

  ```
  Create Table `t1` (
   `id` bigint AUTO_INCREMENT,
   `id_province` bigint NOT NULL,
   `user_info` varchar,
   primary key (`id`)
  ) DISTRIBUTE BY HASH(`id`);
  ```

往表 `t1` 中插入测试数据，语句如下：

```
INSERT INTO t1(id_province,user_info) VALUES (1,'Tom'),(1,'Jerry'),(2,'Jerry'),(3,'Mark');
```

- **创建视图**

  > **说明** 本文示例以在创建视图（视图数据来源于表 `t1` ）时设置不同的安全验证方式为例，介绍DEFINER、INVOKER的不同权限效果。

  - 创建视图 `v1` 时，设置 `SQL SECURITY` 为 `INVOKER` ，语句如下：

    ```
    CREATE SQL SECURITY INVOKER VIEW v1
      AS SELECT id_province,user_info FROM t1 WHERE id_province=1;
    ```

  - 创建视图 `v2` 时，设置 `SQL SECURITY` 为 `DEFINER` ，语句如下：

    ```
    CREATE SQL SECURITY DEFINER VIEW v2
      AS SELECT id_province,user_info FROM t1 WHERE id_province=1;
    ```

  - 创建视图 `v3` 时，不设置 `SQL SECURITY` （即系统默认使用 `INVOKER` ），语句如下：

    ```
    CREATE VIEW v3
      AS SELECT id_province,user_info FROM t1 WHERE id_province=1;
    ```

- **权限对比**

  - 仅通过高权限账号授予 `user1` 查询3个视图的权限，语句如下：

    ```
    GRANT SELECT ON adb_demo.v1 TO 'user1'@'%';
    GRANT SELECT ON adb_demo.v2 TO 'user1'@'%';
    GRANT SELECT ON adb_demo.v3 TO 'user1'@'%';
    ```

    此时，使用 `user1` 账号连接AnalyticDB MySQL版集群的 `adb_demo` 数据库后， `user1` 仅能查询视图 `v2` 数据。查询语句如下：

    ```
    SELECT * FROM v2
    ```

    查询结果如下：

    ```
    +-------------+-----------+
    | ID_PROVINCE | USER_INFO |
    +-------------+-----------+
    |           1 | Tom       |
    |           1 | Jerry     |
    +-------------+-----------+
    ```

    而通过SELECT语句查询 `v1` 或 `v3` 视图数据时则会报错。查询语句如下：

    ```
    SELECT * FROM v1
    ```

    或

    ```
    SELECT * FROM v3
    ```

    执行上述语句进行查询时，均会返回如下错误：

    ```
    ERROR 1815 (HY000): [20049, 2021083110261019216818804803453927668] : Failed analyzing stored view
    ```

  - 在授予 `user1` 查询3个视图的权限基础上，再通过高权限账号授予 `user1` 查询 `t1` 表的权限，语句如下：

    ```
    GRANT SELECT ON adb_demo.t1 to user1@'%';
    ```

    此时，使用 `user1` 账号连接AnalyticDB MySQL版集群的 `adb_demo` 数据库后， `user1` 账号能够查询全部视图 `v1` 、 `v2` 和 `v3` 的数据，查询语句如下：

    ```
    SELECT * FROM v1
    ```

    或

    ```
    SELECT * FROM v2
    ```

    或

    ```
    SELECT * FROM v3
    ```

    执行上述3条查询语句均会返回相同的结果，结果如下：

    ```
    +-------------+-----------+
    | ID_PROVINCE | USER_INFO |
    +-------------+-----------+
    |           1 | Tom       |
    |           1 | Jerry     |
    +-------------+-----------+
    ```

最佳实践

更多详情，请参见通过视图管控数据权限。

3.5.7. DROP DATABASE

`DROP DATABASE` 用于删除数据库。

语法

```
DROP DATABASE db_name;
```

> **说明** 删除数据库之前，必须先删除数据库中的表。

示例

```
use adb_demo2;
+------------------------------+
show tables;
+------------------------------+
| Tables_in_adb_demo2          |
+------------------------------+
| test2                        |
+------------------------------+
drop table test2;
drop database adb_demo2;
```

3.5.8. DROP TABLE

`DROP TABLE` 用于删除表。

语法

```
DROP TABLE db_name.table_name;
```

> **说明** 执行该命令会同时删除表数据和表结构。

示例

```
DROP TABLE adb_demo.customer;
```

3.5.9. DROP VIEW

`DROP VIEW` 用于删除视图。

语法

```
DROP VIEW [IF EXISTS] view_name, [, view_name] ...
```

参数

`view_name`：视图名字，视图名前可以加上数据库名，区分不同数据库中的同名视图。

示例

删除视图v。

```
DROP VIEW v;
```

3.5.10. TRUNCATE TABLE

`TRUNCATE TABLE` 用于清空表数据或者表分区数据。

语法

- 清空表数据

  ```
  TRUNCATE TABLE db_name.table_name;
  ```

- 清空表中的指定分区

  ```
  TRUNCATE TABLE db_name.table_name PARTITION partition_name;
  ```

  分区名的数据类型为bigint，您可以通过以下SQL获取某个表的所有分区名。

```
select partition_name from information_schema.partitions where table_name = 'your_table_name' order by partition_name desc limit 100;
```

注意事项

> 说明
> - 数据库备份期间无法执行，执行 `TRUNCATE TABLE` 将会报错。
> - 执行 `TRUNCATE TABLE` 命令清空表中的数据，表结构不会被删除。

示例

- 清空CUSTOMER表中的数据。

```
TRUNCATE TABLE adb_demo.customer;
```

- 清空表中的指定分区。

```
TRUNCATE TABLE adb_demo.customer partition 20170103,20170104,20170108;
```

3.5.11. SHOW

您可以通过 `SHOW` 语句查看数据库相关信息，例如数据库列表、数据库中的表以及表中的列或索引等。

SHOW DATABASES

查看当前集群中的数据库。

- 语法

```
SHOW DATABASES;
```

- 示例

```
SHOW DATABASES;
```

返回结果如下：

```
+--------------------+
| Database           |
+--------------------+
| adb_test           |
| MYSQL              |
| adb_demo           |
| INFORMATION_SCHEMA |
+--------------------+
```

SHOW TABLES

查看用户当前数据库中的表。

- 语法

```
SHOW TABLES [IN db_name];
```

- 示例

```
SHOW TABLES IN adb_demo;
```

返回结果如下：

```
+--------------------+
| Tables_in_adb_demo |
+--------------------+
| customer           |
| json_test          |
+--------------------+
```

SHOW COLUMNS

查看表的列信息。

- 语法

```
SHOW COLUMNS IN db_name.table_name;
```

- 示例

```
SHOW COLUMNS IN adb_demo.customer;
```

返回结果如下：

```
+---------+---------+------+------+---------+-------+
| Field   | Type    | Null | Key  | Default | Extra |
+---------+---------+------+------+---------+-------+
| id      | int     | NO   | PRI  | NULL    |       |
| name    | varchar | YES  |      | NULL    |       |
| address | varchar | YES  |      | NULL    |       |
| gender  | boolean | YES  |      | NULL    |       |
+---------+---------+------+------+---------+-------+
```

SHOW CREATE TABLE

查看表的建表语句。

- 语法

```
SHOW CREATE TABLE db_name.table_name;
```

- 示例

```
SHOW CREATE TABLE adb_demo.customer;
```

返回结果如下：

```
+----------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table    | Create Table                                                                                                                                                                                       |
+----------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| customer | Create Table `customer` (
 `id` int NOT NULL,
 `name` varchar(50),
 `address` varchar(80),
 `gender` boolean,
 primary key (`id`)
) DISTRIBUTE BY HASH(`id`) INDEX_ALL='Y' STORAGE_POLICY='HOT' BLOCK_SIZE=8192 |
+----------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

SHOW GRANTS

查看当前登录用户的权限。

- 语法

```
SHOW GRANTS;
```

- 示例

```
SHOW GRANTS;
```

返回结果如下：

```
+----------------------------------------------------------+
| Grants for adb_acc@%                                     |
+----------------------------------------------------------+
| GRANT ALL ON `*`.`*` TO 'adb_acc'@'%' WITH GRANT OPTION |
+----------------------------------------------------------+
```

SHOW INDEXES

查看表的索引信息。

- 语法

```
SHOW INDEXES FROM db_name.table_name;
```

- 示例

```
SHOW INDEXES FROM adb_demo.json_test;
```

返回结果如下，其中 `Key_name` 即为索引名：

```
+-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+-----
----+---------------+
| Table     | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comm
ent | Index_comment |
+-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+-----
----+---------------+
| json_test |          1 | id_0_idx |            1 | id          | A         |           0 |     NULL | NULL   |      | BTREE      |
|               |
| json_test |          1 | vj_idx   |            1 | vj          | A         |           0 |     NULL | NULL   |      | BTREE      |
|               |
+-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+-----
----+---------------+
```

3.6. DML

3.6.1. INSERT INTO

`INSERT INTO` 用于向表中插入数据，遇到主键重复时会自动忽略当前写入数据，不做更新，作用等同于 `INSERT IGNORE INTO` 。

语法

```
INSERT [IGNORE]
    INTO table_name
    [( column_name [, …] )]
    [VALUES]
    [(value_list[, …])]
    [query];
```

参数

- `IGNORE` ：可选参数，若系统中已有相同主键的记录，新记录不会被写入。
- `column_name` ：可选参数，列名。
- `query` ：通过定义查询，将一行或多行数据插入表中。

注意事项

如果插入数据时不指定列名，则要插入的数据必须和CREATE TABLE语句中声明的列的顺序一致。

示例

创建CUSTOMER和COURSES表。

```
CREATE TABLE customer (
  customer_id bigint NOT NULL COMMENT '顾客ID',
  customer_name varchar NOT NULL COMMENT '顾客姓名',
  phone_num bigint NOT NULL COMMENT '电话',
  city_name varchar NOT NULL COMMENT '所属城市',
  sex int NOT NULL COMMENT '性别',
  id_number varchar NOT NULL COMMENT '身份证号码',
  home_address varchar NOT NULL COMMENT '家庭住址',
  office_address varchar NOT NULL COMMENT '办公地址',
  age int NOT NULL COMMENT '年龄',
  login_time timestamp NOT NULL COMMENT '登录时间',
  PRIMARY KEY (login_time, customer_id, phone_num)
)
DISTRIBUTED BY HASH(customer_id)
PARTITION BY VALUE(DATE_FORMAT(login_time, '%Y%m%d')) LIFECYCLE 30
COMMENT '客户信息表';
```

```
CREATE TABLE courses(
id bigint AUTO_INCREMENT PRIMARY KEY,
name varchar(20) NOT NULL,
grade varchar(20) default '三年级',
submission_date timestamp
)
DISTRIBUTED BY HASH(id)
```

- 向CUSTOMER表中插入一条数据。

  ```
  INSERT INTO customer(customer_id,customer_name,phone_num,city_name,sex,id_number,home_address,office_address,age,login_time)
  values
  (002367,'杨过','13900001234','杭州',0,'987300','西湖','转塘云栖小镇',23,'2018-03-02 10:00:00');
  ```

- 向CUSTOMER表中插入多条数据。

```
INSERT INTO  customer(customer_id,customer_name,phone_num,city_name,sex,id_number,home_address,office_address,age,login_time)
values
(002367,'李四','13900001234','杭州',0,'987300','西湖','转塘云栖小镇',23,'2018-03-02 10:00:00'),(002368,'张三','13900001111','杭州',0,'987300'
,'西湖','转塘云栖小镇',28,'2018-08-01 11:00:00'),(002369,'王五','13900002222','杭州',1,'987300','西湖','转塘云栖小镇',35,'2018-09-12 08:11:00'
);
```

- 向CUSTOMER表中插入多条数据时，可以省略列名。

```
INSERT INTO
customer values
(002367,'李四','13900001234','杭州',0,'987300','西湖','转塘云栖小镇',23,'2018-03-02 10:00:00'),(002368,'张三','13900001111','杭州',0,'987300'
,'西湖','转塘云栖小镇',28,'2018-08-01 11:00:00'),(002369,'王五','13900002222','杭州',1,'987300','西湖','转塘云栖小镇',35,'2018-09-12 08:11:00'
);
```

- 向COURSES表中插入一条数据。

```
insert into courses (name,submission_date) values("Jams",NOW());
```

- `INSERT query` 示例请参见INSERT SELECT FROM。

3.6.2. REPLACE INTO

本文介绍如何使用 `REPLACE INTO` 语句。

语法

`REPLACE INTO` 用于实时覆盖写入数据。写入数据时，会先根据主键判断待写入的数据是否已经存在于表中，并根据判断结果选择不同的方式写入数据：

- 如果待写入数据已经存在，则先删除该行数据，然后插入新的数据。
- 如果待写入数据不存在，则直接插入新数据。

> **说明** 本文所指的主键既包括单个列的主键，也包括由多列组成的复合主键（Composite Primary Key）。

语法如下：

```
REPLACE INTO table_name [(column_name,...)] VALUES ({常量|NULL|DEFAULT},...),(...),...
```

示例

- 通过 `REPLACE INTO` 向CUSTOMER表中插入一条数据。

```
REPLACE INTO customer(customer_id,customer_name,phone_num,city_name,sex,id_number,home_address,office_address,age,login_time)
values
(002367,'Bob','13900001234','Hangzhou',0,'987300','WestLake','CloudTown',23,'2018-03-02 10:00:00');
```

- 向CUSTOMER表中插入多条数据时，可以省略列名。

```
REPLACE INTO
customer values
(002367,'John','13900001111','Hangzhou',0,'987300','WestLake','CloudTown',23,'2018-03-02 10:00:00'),(002368,'Adam','13900002222','Hangzho
u',0,'987300','WestLake','CloudTown',28,'2018-08-01 11:00:00'), (002369,'Brook','13900003333','Hangzhou',1,'987300','WestLake','CloudTown
',35,'2018-09-12 08:11:00');
```

3.6.3. ALTER RESOURCE GROUP

ADB弹性模式集群版（新版）支持通过 `ALTER RESOURCE GROUP` 修改资源池的资源量、查询类型、用户绑定关系属性。

语法

```
ALTER RESOURCE GROUP resource_group_name
    [QUERY_TYPE = {interactive, batch, default_type}]
    [NODE_NUM = N]
    [ADD_USER=user_name]
    [DROP_USER=user_name]
```

- 用户与资源池绑定的操作 `ADD_USER` 和 `DROP_USER` 必须单独操作，不可以与其他属性搭配使用。
- 资源池名大小写不敏感，例如test_group和Test_Group是同名资源池，资源池名展示为TEST_GROUP。

参数

参数	说明
QUERY_TYPE	查询类型。可选值为： • interactive • batch • default_type 默认值为default_type。详情请参见查询执行模式（数仓版）。

参数	说明
NODE_NUM	节点个数。指定资源池的节点个数，默认值为0。
ADD_USER	绑定用户。添加用户与资源池的绑定关系。
DROP_USER	解绑用户。解除用户与资源池的绑定关系。

示例

- 修改资源池的资源量。

  修改资源池的NODE_NUM为3个计算节点：

  ```
  ALTER RESOURCE GROUP 'BATCH_RG' NODE_NUM=3
  ```

- 修改资源池的查询类型。

  修改资源池的QUERY_TYPE为默认值default_type：

  ```
  ALTER RESOURCE GROUP 'BATCH_RG' QUERY_TYPE=default_type
  ```

- 修改资源池的查询类型和资源量。

  ```
  ALTER RESOURCE GROUP 'BATCH_RG' QUERY_TYPE=default_type NODE_NUM=3
  ```

- 绑定用户与资源池

  ```
  ALTER RESOURCE GROUP 'BATCH_RG' ADD_USER=batch_user
  ```

  注意 不可以与其他属性一起使用，下面的语句是错误的。

  ```
  ALTER RESOURCE GROUP 'BATCH_RG' ADD_USER=batch_user QUERY_TYPE=default_type
  ```

- 解绑用户与资源池

  ```
  ALTER RESOURCE GROUP 'BATCH_RG' DROP_USER=batch_user
  ```

- 修改默认资源池的QUERY_TYPE

  ```
  -- 修改默认资源池为 batch 查询类型
  ALTER RESOURCE GROUP user_default QUERY_TYPE=batch
  -- 恢复为默认值
  ALTER RESOURCE GROUP user_default QUERY_TYPE=default_type
  ```

- 查看当前资源池列表

  ```
  SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS;
  ```

3.6.4. DROP RESOURCE GROUP

AnalyticDB MySQL版的弹性模式集群版（新版）支持通过 `DROP RESOURCE GROUP` 删除资源池。

语法

```
DROP RESOURCE GROUP resource_group_name;
```

说明

- 删除资源池的同时会解除所有数据库账号与该资源池绑定的关系。
- 资源池名大小写不敏感，例如 `test_group` 和 `Test_Group` 是同名资源池，资源池名展示为 `TEST_GROUP` 。

示例

1. 删除资源池，命令如下：

   ```
   DROP RESOURCE GROUP 'BATCH_RG';
   ```

2. 查看当前资源池列表，命令如下：

   ```
   SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS;
   ```

3.6.5. INSERT SELECT FROM

如果您的数据在其他表中已经存在，可以通过 `INSERT SELECT FROM` 将数据复制到另外一张表。

语法

```
INSERT INTO table_name
[( column_name [, …] )]
query;
```

参数

- `column_name`：列名，如果需要将源表中的部分列数据插入到目标表中，SELECT子句中的列必须与INSERT子句中列的顺序、数据类型一致。
- `query`：可以是`SELECT FROM TABLE`或者`SELECT FROM VIEW`。

示例

- 以指定列名的方式，从CUSTOMER表中复制某几列数据到NEW_CUSTOMER表中。

```
INSERT INTO new_customer (customer_id, customer_name, phone_num)
SELECT customer_id, customer_name, phone_num FROM customer
WHERE customer.customer_name = '杨过';
```

- 不指定列名，从CUSTOMER表中复制所有列数据到NEW_CUSTOMER表中。

```
INSERT INTO new_customer
SELECT (customer_id,customer_name,phone_num,city_name,sex,id_number,home_address,office_address,age,login_time) FROM customer
WHERE customer.customer_name = '杨过';
```

3.6.6. REPLACE SELECT FROM

`REPLACE SELECT FROM`用于将其他表中的数据实时覆盖写入目标表中。写入数据时，根据主键判断待写入的数据是否已经存在于表中，如果已经存在，则先删除该行数据，然后插入新的数据；如果不存在，则直接插入新数据。

语法

```
REPLACE INTO table_name
[(column_name,...)]
query;
```

参数

- `query`：可以是`SELECT FROM TABLE`或者`SELECT FROM VIEW`。
- `column_name`：列名，如果需要将源表中的部分列数据插入到目标表中，`SELECT`子句中的列必须与`REPLACE`子句中列的顺序、数据类型一致。

注意事项

执行`REPLACE SELECT FROM`命令时，需先创建待写入数据的目标表。

示例

以指定列名的方式，从CUSTOMER表中复制某几列数据到NEW_CUSTOMER表中。

```
REPLACE INTO new_customer (customer_id, customer_name, phone_num)
  SELECT customer_id, customer_name, phone_num FROM customer
  WHERE customer.customer_name = '杨过';
```

3.6.7. INSERT OVERWRITE INTO SELECT

本文介绍云原生数据仓库AnalyticDB MySQL版表数据的高性能写入方式`INSERT OVERWRITE INTO SELECT`，包括应用场景、功能原理、注意事项、语法和异步写入应用。

应用场景

`INSERT OVERWRITE INTO SELECT`常规的使用场景包括：

- 进行分区级数据写入。
- 进行数据初始化（全量写入）。
- 进行大批量数据写入操作，不建议用于少量数据的写入。

功能原理

写入任务是通过外表方式将外部数据批量写入到AnalyticDB MySQL内部的。需要在AnalyticDB MySQL中定义对应数据源的外表，然后通过`INSERT OVERWRITE INTO SELECT`语句将外表数据写入AnalyticDB MySQL表。

每个表的写入任务串行执行，即单表写入并发为1，无法调整。为保证单任务写入性能，防止集群负载过高，集群写入任务并发默认为2，不建议调整。

> **说明** 如有需要调整写入并发数，请提交工单联系技术支持，由技术支持评估调整。

`INSERT OVERWRITE INTO SELECT`的基本特性如下。

- 资源消耗大：该写入模式在进行高性能写入时会消耗大量集群资源，建议在业务低峰期使用。
- 批量可见：写入任务完成前数据不可见，任务完成后该任务写入的数据批量可见。如果目标表中已存在数据，`INSERT OVERWRITE INTO SELECT`命令执行结束

之前，目标表中的数据不会发生任何变化； `INSERT OVERWRITE INTO SELECT` 命令执行结束后，系统自动将数据写入目标表中，目标表的原数据将被清空。

- 分区覆盖：通过 `INSERT OVERWRITE INTO SELECT` 写入的分区数据会覆盖目标表同一分区的数据。
- 自动构建索引：写入时同步构建索引，写入任务完成，目标表则具备索引，可提升查询性能。

注意事项

不能同时通过 `INSERT OVERWRITE INTO SELECT` 和实时写入方式（INSERT INTO、REPLACE INTO、DELETE、UPDATE）向同一个表中写入数据，否则实时写入的数据会被丢弃。

语法

```
INSERT OVERWRITE INTO table_name [(column_name,...)]
select_statement
```

参数说明

- table_name：目标表的表名。
- column_name：目标表的列名。
- select_statement：SELECT查询语句。

  SELECT语句中每一列的数据类型需要与[(column_name,...)]中的数据类型相匹配。

  如果SELECT语句中的列数比[(column_name,...)]中的列数多，会写入失败；如果SELECT语句中的列数比[(column_name,...)]中的列数少，写入数据时，[(column_name,...)]中最后多出的列会自动填充默认值，无默认值时值为NULL。

示例

从顾客表Customers中选择cust_id列的数据写入到订单表Vendors中的vend_id列，示例语句如下。

```
INSERT OVERWRITE INTO Vendors (vend_id)
SELECT cust_id FROM Customers;
```

异步写入

提交任务

通常使用 `SUBMIT JOB` 提交异步任务，由后台调度执行。示例语句如下。

```
SUBMIT JOB
INSERT OVERWRITE adb_table
SELECT * FROM adb_external_table
```

写入调优

在写入任务前增加Hint（ `/* direct_batch_load=true*/` ）可以加速写入任务。该Hint可以在节约大量资源的情况下进一步提高写入性能。示例语句如下。

```
/* direct_batch_load=true*/
SUBMIT JOB
INSERT OVERWRITE adb_table
SELECT * FROM adb_external_table
```

> **说明** 仅3.1.5及以上内核版本支持 `/* direct_batch_load=true*/` 。若使用后性能无明显优化，可提交工单进行升级与优化。查看内核版本，请参见如何查看实例版本信息。

进度查询

通过 `SUBMIT JOB` 提交写入任务后会返回job_id。以该job_id为条件查询写入任务的状态，示例语句如下。

```
show job status where job='<job_id>'
```

返回结果status列为FINISH，则写入任务完成。

3.6.8. INSERT ON DUPLICATE KEY UPDATE

本文介绍如何使用 `INSERT ON DUPLICATE KEY UPDATE` 。

功能说明

执行 `INSERT ON DUPLICATE KEY UPDATE` 语句时，AnalyticDB MySQL版会首先尝试在表中插入新行，但如果新的数据与已有数据的主键重复，将使用 `INSERT ON DUPLICATE KEY UPDATE` 子句中指定的值更新现有行。AnalyticDB MySQL版会根据待写入行是否存在选择对应的执行语句，规则如下：

- 待写入行不存在，则执行INSERT来插入新行，受影响的行数为1。
- 待写入行存在，则执行UPDATE来更新现有行，受影响的行数也为1。

语法

```
INSERT INTO table_name[(column_name[, …])]
[VALUES]
[(value_list[, …])]
ON DUPLICATE KEY UPDATE
   c1 = v1,
   c2 = v2,
   ...;
```

示例

本文所有示例均基于 `student_course` 表，建表语句如下：

```
CREATE TABLE student_course(
    id bigint,
    user_id bigint,
    nc_id varchar,
    nc_user_id varchar,
    nc_commodity_id varchar,
    course_no varchar,
    course_name varchar,
    business_id varchar,
    PRIMARY KEY(user_id)
) DISTRIBUTED BY HASH(user_id);
```

使用如下语句插入一行数据：

```
INSERT INTO student_course (`id`, `user_id`, `nc_id`, `nc_user_id`, `nc_commodity_id`, `course_no`, `course_name`, `business_id`)
VALUES(277941, 11056941, '1001EE1000000043G2T5', '1001EE1000000043G2TO', '1001A5100000003YABO2', 'kckm303', '工业会计实战V9.0--55', 'kuaiji')
ON DUPLICATE KEY UPDATE
course_name = '工业会计实战V9.0--55',
business_id = 'kuaiji';
```

执行 `SELECT * FROM student_course;` 语句，返回如下结果说明数据插入成功：

```
+-------+----------+--------------------+--------------------+--------------------+-----------+--------------------+------------+
| id    | user_id  | nc_id              | nc_user_id         | nc_commodity_id    | course_no | course_name        |business_id |
+-------+----------+--------------------+--------------------+--------------------+-----------+--------------------+------------+
|277941 | 11056941 |1001EE1000000043G2T5 |1001EE1000000043G2TO |1001A5100000003YABO2 |kckm303    |工业会计实战V9.0--55 |kuaiji      |
```

此时，需要再往 `student_course` 表中入一行新数据：

```
INSERT INTO student_course(`id`, `user_id`, `nc_id`, `nc_user_id`, `nc_commodity_id`, `course_no`, `course_name`, `business_id`)
VALUES(277942, 11056941, '1001EE1000000043G2T5', '1001EE1000000043G2TO', '1001A5100000003YABO2', 'kckm303', '工业会计实战V9.0--66', 'kuaiji')
ON DUPLICATE KEY UPDATE
course_name = '工业会计实战V9.0--66',
business_id = 'kuaiji';
```

但由于新插入的数据中存在重复主键（即 `user_id` 与第一次插入的数据重复，均为 `11056941` ），因此执行上述语句后只会更新 `ON DUPLICATE KEY UPDATE` 子句中的 `course_name = '工业会计实战V9.0--66',business_id = 'kuaiji'` 值，您可以执行 `SELECT * FROM student_course;` 语句来查看更新后的数据，返回结果如下：

```
+-------+----------+--------------------+--------------------+--------------------+-----------+--------------------+------------+
| id    | user_id  | nc_id              | nc_user_id         | nc_commodity_id    | course_no | course_name        |business_id |
+-------+----------+--------------------+--------------------+--------------------+-----------+--------------------+------------+
|277941 | 11056941 |1001EE1000000043G2T5 |1001EE1000000043G2TO |1001A5100000003YABO2 |kckm303    |工业会计实战V9.0--66 |kuaiji      |
```

常见问题

Q：是否支持通过Logstash插件使用 `INSERT ON DUPLICATE KEY UPDATE` 语句批量插入数据？

A：支持。使用 `INSERT ON DUPLICATE KEY UPDATE` 语句批量插入数据时，您无需在每个 `VALUES()` 语句后都添加 `ON DUPLICATE KEY UPDATE` ，仅需在最后一个 `VALUES()` 后加上即可。

例如，需要在 `student_course` 表中批量插入3条数据，您可以执行如下语句：

```
INSERT INTO student_course(`id`, `user_id`, `nc_id`, `nc_user_id`, `nc_commodity_id`, `course_no`, `course_name`, `business_id`)
VALUES(277943, 11056941, '1001EE1000000043G2T5', '1001EE1000000043G2TO', '1001A5100000003YABO2', 'kckm303', '工业会计实战V9.0--77', 'kuaiji')
,
(277944, 11056943, '1001EE1000000043G2T5', '1001EE1000000043G2TO', '1001A5100000003YABO2', 'kckm303', '工业会计实战V9.0--88', 'kuaiji'),
(277945, 11056944, '1001EE1000000043G2T5', '1001EE1000000043G2TO', '1001A5100000003YABO2', 'kckm303', '工业会计实战V9.0--99', 'kuaiji')
ON DUPLICATE KEY UPDATE
course_name = '工业会计实战V9.0--77',
business_id = 'kuaiji';
```

更多关于Logstash插件详情，请参见Logstash概述。

3.6.9. UPDATE

`UPDATE` 用于更新数据。

语法

```
UPDATE  table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
```

注意事项

- 执行 `UPDATE` 命令时，要求表中存在主键。
- 暂不支持更新主键列。
- 暂不支持批量更新多条SQL语句。

示例

将CUSTOMER表中 `customer_id ='2369'` 客户的姓名更改为黄女士。

```
update customer set customer_name ='黄女士' where customer_id ='2369';
```

3.6.10. DELETE

`DELETE` 用于删除表中的记录。

语法

```
DELETE FROM table_name
[ WHERE condition ]
```

注意事项

- 执行 `DELETE` 命令时，表中必须存在主键。
- `DELETE` 暂不支持使用表的别名。
- 不建议通过 `DELETE` 命令做全表、全分区删除，建议使用 `TRUNCATE TABLE` 、 `TRUNCATE PARTITION` 。

示例

- 删除CUSTOMER表中 `name` 为 `张三` 的数据。

  ```
  DELETE FROM customer WHERE customer_name='张三';
  ```

- 删除CUSTOMER表的中多行。

  ```
  DELETE FROM customer WHERE age<18;
  ```

3.6.11. KILL PROCESS

`KILL PROCESS` 用于终止正在运行的PROCESS。

语法

```
KILL PROCESS process_id
```

参数

process_id：来源于SHOW PROCESSLIST返回结果中的ProcessId字段。

权限

- 默认您可以通过 `KILL PROCESS` 终止您当前账号下正在运行的PROCESS。
- 高权限账号通过GRANT语句授予普通账号PROCESS权限，普通账号可以终止集群下所有用户正在运行的PROCESS。

  ```
  GRANT process on *.* to account_name;
  ```

3.6.12. SHOW PROCESSLIST

`SHOW PROCESSLIST` 用于查看正在运行的PROCESS。

> 说明 您也可以通过 `INFORMATION_SCHEMA PROCESSLIST` 表查看正在运行的PROCESS。

语法

```
SHOW [FULL] PROCESSLIST
```

返回参数

执行 `SHOW FULL PROCESSLIST` 或者 `SHOW PROCESSLIST` 后，返回结果中包含以下参数。

- Id：PROCESS的Id。
- ProcessId：任务的唯一标识，执行 `KILL PROCESS` 时需要使用ProcessId。
- User：当前用户。
- Host：显示发出这个语句的客户端的主机名，包含IP和端口号。
- DB：显示该PROCESS目前连接的是哪个数据库。
- Command：显示当前连接所执行的命令，即休眠（sleep）、查询（query）以及连接（connect）三种类型的命令。
- Time：显示Command执行的时间，单位为秒。
- State：显示当前连接下SQL语句的执行状态。
- Info：显示SQL语句。

> **说明** 如果不使用 `FULL` 关键字，只能查看每个记录中Info字段的前100个字符。

权限

- 默认您可以通过 `SHOW PROCESSLIST` ，查看您当前账号下正在运行的PROCESS。
- 高权限账号通过GRANT语句授予普通账号PROCESS权限，普通账号可以查看集群下所有用户正在运行的PROCESS。

```
GRANT process on *.* to account_name;
```

3.7. DQL

3.7.1. SELECT

3.7.1.1. 语法

`SELECT` 语句用于从一个或多个表中查询数据，具体语法如下所示。

```
[ WITH with_query [, ...] ]
SELECT
[ ALL | DISTINCT ] select_expr [, ...]
[ FROM table_reference [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition]
[ WINDOW window_name AS (window_spec) [, window_name AS (window_spec)] ...]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY {column_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

- `table_reference` ：查询的数据源，可以是表、视图、关联表或者子查询。
- 表名和列名不区分大小写。
- 表名和列名中如果含有关键字或者空格等字符，可以使用反引号（``）将其引起来。

WHERE

`WHERE` 关键字后跟 `BOOLEAN` 表达式，用于从表中查询满足条件的数据。例如，在CUSTOMER表中查询 `customer_id` 为 `2368` 的顾客信息。

```
SELECT * FROM CUSTOMER where customer_id=2368;
```

ALL和DISTINCT

ALL和DISTINCT关键字用于指定查询结果是否返回重复的行，默认值为ALL，即返回所有匹配的行，DISTINCT将从结果集中删除重复的行。

```
SELECT col1, col2 FROM t1;
SELECT DISTINCT col1, col2 FROM t1;
```

以下为SELECT中的其他关键字用法。

- LIMIT
- UNION、INTERSECT和EXCEPT
- WITH
- GROUP BY
- HAVING
- JOIN
- ORDER BY

- 子查询

3.7.1.2. LIMIT

`LIMIT` 子句用于限制最终结果集的行数， `LIMIT` 子句中通常会携带一个或两个数字参数，第一个参数指定要返回数据行的第一行的偏移量，第二个参数指定要返回的最大行数。

以下示例查询ORDERS表，通过LIMIT限制返回结果，仅返回5行数据。

```
SELECT orderdate FROM orders LIMIT 5;
-------------
o_orderdate
-------------
 1996-04-14
 1992-01-15
 1995-02-01
 1995-11-12
 1992-04-26
```

以下示例查询CUSTOMER表，按照创建时间排序，返回第3个到第7个客户的信息。

```
SELECT * FROM customer ORDER BY create_date LIMIT 2,5
```

3.7.1.3. UNION、INTERSECT和EXCEPT

`UNION` 、 `INTERSECT` 和 `EXCEPT` 用于将多个查询结果集进行组合，从而得到一个最终结果。

语法

```
query
{ UNION [ ALL ] | INTERSECT | EXCEPT }
query
```

参数

- `UNION` ：返回两个查询表达式的集合运算。
- `UNION ALL` ： `ALL` 关键字用于保留 `UNION` 中产生的重复行。
- `INTERSECT` ：返回只有在两个集合中同时出现的行，返回结果将删除两个集合中的重复行。
- `EXCEPT` ：先删除两个集合中重复的数据行，返回只在第一个集合中出现且不在第二个集合中出现的所有行。

计算顺序

- `UNION` 和 `EXCEPT` 集合运算符为左关联，如果未使用圆括号来改变计算顺序，则按照从左到右的顺序进行集合运算。

  例如，以下查询中，首先计算 `T1` 和 `T2` 的 `UNION` ，然后对 `UNION` 结果执行 `EXCEPT` 操作。

  ```
  select * from t1
  union
  select * from t2
  except
  select * from t3
  order by c1;
  ```

- 在同一查询中，组合使用集合运算符时， `INTERSECT` 运算符优先于 `UNION` 和 `EXCEPT` 运算符。

  例如，以下查询先计算 `T2` 和 `T3` 的交集，然后将计算得到的结果与 `T1` 进行并集。

  ```
  select * from t1
  union
  select * from t2
  intersect
  select * from t3
  order by c1;
  ```

- 可以使用圆括号改变集合运算符的计算顺序。

  以下示例中，将 `T1` 和 `T2` 的并集结果与 `T3` 执行交集运算。

  ```
  (select * from t1
  union
  select * from t2)
  intersect
  (select * from t3)
  order by c1;
  ```

3.7.1.4. WITH

本文介绍如何在 `SELECT` 语句中使用 `WITH` 子句。

查询中可以使用 `WITH` 子句来创建通用表达式（Common Table Express，简称CTE），`WITH` 子句定义的子查询，供 `SELECT` 查询引用。`WITH` 子句可以扁平化嵌套查询或者简化子查询，`SELECT` 只需执行一遍子查询，提高查询性能。

> 说明
> - CTE是一个命名的临时结果集，仅在单个SQL语句（例如SELECT、INSERT或DELETE）的执行范围内存在。
> - CTE仅在查询执行期间持续。

注意事项

- CTE之后可以接SQL语句（例如 `SELECT`、`INSERT` 或 `UPDATE` 等）或者其他的CTE（只能使用一个 `WITH`），多个CTE中间用逗号（,）分隔，否则CTE将失效。
- CTE语句中暂不支持分页功能。

WITH使用方法

- **以下两个查询等价**

```
SELECT a, b
FROM (SELECT a, MAX(b) AS b FROM t GROUP BY a) AS x;
```

```
WITH x AS (SELECT a, MAX(b) AS b FROM t GROUP BY a)
SELECT a, b FROM x;
```

- **WITH子句可用于多子查询：**

```
WITH
t1 AS (SELECT a, MAX(b) AS b FROM x GROUP BY a),
t2 AS (SELECT a, AVG(d) AS d FROM y GROUP BY a)
SELECT t1.*, t2.*
FROM t1 JOIN t2 ON t1.a = t2.a;
```

- **WITH子句中定义的关系可以互相连接**

```
WITH
x AS (SELECT a FROM t),
y AS (SELECT a AS b FROM x),
z AS (SELECT b AS c FROM y)
SELECT c FROM z;
```

3.7.1.5. GROUP BY

`GROUP BY` 子句用于对查询结果进行分组，可以在 `GROUP BY` 中使用 `GROUPING SETS`、`CUBE`、`ROLLUP` 以不同的形式展示分组结果。

```
GROUP BY expression [, ...]
```

GROUPING SETS

`GROUPING SETS` 用于在同一结果集中指定多个 `GROUP BY` 选项，作用相当于多个 `GROUP BY` 查询的 `UNION` 组合形式。

```
SELECT origin_state, origin_zip, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state),
    (origin_state, origin_zip),
    (destination_state));
```

上述示例等同于：

```
SELECT origin_state, NULL, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state
UNION ALL
SELECT origin_state, origin_zip, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state, origin_zip
UNION ALL
SELECT NULL, NULL, destination_state, sum(package_weight)
FROM shipping GROUP BY destination_state;
```

CUBE

`CUBE` 用于列出所有可能的分组集。

```
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY origin_state, destination_state WITH CUBE
```

上述示例等同于：

```
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ())
```

ROLLUP

`ROLLUP` 可以以层级的方式列出分组集。

```
SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY ROLLUP (origin_state, origin_zip)
```

上述示例等同于：

```
SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS ((origin_state, origin_zip), (origin_state), ())
```

注意事项

- 查询中必须使用标准聚合函数（ `SUM` 、 `AVG` 或 `COUNT` ）声明非分组列，否则无法使用 `GROUP BY` 子句。
- `GROUP BY` 中的列或表达式列表必须与查询列表中的非聚合表达式的列相同。

示例

例如，以下查询列表中包含两个聚合表达式，第一个聚合表达式使用 `SUM` 函数，第二个聚合表达式使用 `COUNT` 函数，其余两列（ `LISTID` 、 `EVENTID` ）声明为分组列。

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by listid, eventid
order by 3, 4, 2, 1
limit 5;
listid | eventid | revenue | numtix
-------+---------+---------+--------
89397|      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
(5 rows)
```

`GROUP BY` 子句中的表达式也可以使用序号来引用所需的列。

例如，上述示例可改写为以下形式。

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by 1,2
order by 3, 4, 2, 1
limit 5;
listid | eventid | revenue | numtix
-------+---------+---------+--------
89397  |      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
```

3.7.1.6. HAVING

`HAVING` 子句与聚合函数以及 `GROUP BY` 子句一起使用，在分组和聚合计算完成后，HAVING子句对分组进行过滤，去掉不满足条件的分组。

```
[ HAVING condition ]
```

注意事项

- `HAVING` 条件中引用的列必须为分组列或引用了聚合函数结果的列。
- `HAVING` 子句必须与聚合函数以及 `GROUP BY` 子句一起使用，用于对 `GROUP BY` 分组进行过滤，去掉不满足条件的分组。

示例

在CUSTOMER表中，进行分组查询，查询账户余额大于指定值的记录。

```
SELECT count(*), mktsegment, nationkey,
       CAST(sum(acctbal) AS bigint) AS totalbal
FROM customer
GROUP BY mktsegment, nationkey
HAVING sum(acctbal) > 5700000
ORDER BY totalbal DESC;
 _col0 | mktsegment | nationkey | totalbal
-------+------------+-----------+----------
  1272 | AUTOMOBILE |        19 |  5856939
  1253 | FURNITURE  |        14 |  5794887
  1248 | FURNITURE  |         9 |  5784628
  1243 | FURNITURE  |        12 |  5757371
  1231 | HOUSEHOLD  |         3 |  5753216
  1251 | MACHINERY  |         2 |  5719140
  1247 | FURNITURE  |         8 |  5701952
```

3.7.1.7. JOIN

以下查询由 `FROM` 子句中的两个子查询联接组成，查询不同类别活动（音乐会和演出）的已售门票数和未售门票数。

```
join_table:
    table_reference [INNER] JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference join_condition
  | table_reference CROSS JOIN table_reference [join_condition])
table_reference:
    table_factor
  | join_table
table_factor:
    tbl_name [alias]
  | table_subquery alias
  | ( table_references )
join_condition:
    ON expression
```

示例

```
select catgroup1, sold, unsold
from
(select catgroup, sum(qtysold) as sold
from category c, event e, sales s
where c.catid = e.catid and e.eventid = s.eventid
group by catgroup) as a(catgroup1, sold)
join
(select catgroup, sum(numtickets)-sum(qtysold) as unsold
from category c, event e, sales s, listing l
where c.catid = e.catid and e.eventid = s.eventid
and s.listid = l.listid
group by catgroup) as b(catgroup2, unsold)
on a.catgroup1 = b.catgroup2
order by 1;
```

3.7.1.8. ORDER BY

本文介绍如何使用 `ORDER BY` 子句对查询结果进行排序。

语法

```
ORDER BY expression
[ ASC | DESC ]
[ LIMIT count]
```

参数	是否必填	说明
`expression`	必填	指定需要进行排序的字段，取值范围如下： • 输出列的名称。例如 `device` ，表示根据 `device` 列进行排序。 • 输出列的序号，即结果列按从左到右排列时的位置顺序，从1开始。例如 `4` ，表示根据返回结果的第4列进行排序。

参数	是否必填	说明
[ASC | DESC]	选填	指定查询结果根据目标列进行升序或降序排列，取值范围如下： • ASC：升序。 • DESC：降序。 说明 • 未指定该参数时，默认按升序排序。 • 若在 expression 参数中指定了多个字段，需要为每个字段单独指定升序或降序。例如 ORDER BY 2 ASC, 4 DESC，表示返回结果按照第2列升序，第四列降序进行排序。
[LIMIT count]		指定查询结果的返回行数。 未指定该参数时，默认返回所有结果行。

示例

- 统计在不同城市售出的设备数量，并按照设备名称和销售量排列，语句如下：

```
SELECT os ,device,city,COUNT(*) AS num FROM requests GROUP BY os,device,city ORDER BY num,device;
```

返回结果如下：

```
os      |device |city         |num
--------+-------+-------------+---
windows |PC     |Hangzhou     |1
windows |PC     |Shenzhen     |1
windows |PC     |Shanghai     |1
linux   |PC     |Shanghai     |1
windows |PC     |Shijiazhuang |2
linux   |PC     |Beijing      |2
windows |PC     |Beijing      |4
linux   |Phone  |Hangzhou     |1
ios     |Phone  |Zhangjiakou  |1
windows |Phone  |Shenzhen     |1
linux   |Phone  |Beijing      |2
ios     |Phone  |Shijiazhuang |2
windows |Phone  |Shijiazhuang |2
```

- 统计设备销售数量排名前5的城市，并按照设备名称降序，按照销售量升序排列，语句如下：

```
SELECT os,device,city,COUNT(*) AS num FROM requests GROUP BY os,device,city ORDER BY 2 DESC,4 ASC LIMIT 5;
```

返回结果如下：

```
os     |device |city        |num
-------+-------+------------+----
windows|Phone  |Shenzhen    |1
linux  |Phone  |Hangzhou    |1
ios    |Phone  |Zhangjiakou |1
linux  |Phone  |Beijing     |2
ios    |Phone  |Shijiazhuang|2
```

3.7.1.9. 子查询

以下示例查询门票销量排名前10的卖家，WHERE 子句中包含一个表子查询，子查询生成多个行，每行包含一列数据。

说明 表子查询可以包含多个列和行。

```
select firstname, lastname, cityname, max(qtysold) as maxsold
from users join sales on users.userid=sales.sellerid
where users.cityname not in(select venuecity from venue)
group by firstname, lastname, cityname
order by maxsold desc, cityname desc
limit 10;
firstname  | lastname |      cityname  | maxsold
-----------+----------+----------------+---------
Noah       | Guerrero | Worcester      |       8
Isadora    | Moss     | Winooski       |       8
Kieran     | Harrison | Westminster    |       8
Heidi      | Davis    | Warwick        |       8
Sara       | Anthony  | Waco           |       8
Bree       | Buck     | Valdez         |       8
Evangeline | Sampson  | Trenton        |       8
Kendall    | Keith    | Stillwater     |       8
Bertha     | Bishop   | Stevens Point  |       8
Patricia   | Anderson | South Portland |       8
```

3.7.2. 查询用户

AnalyticDB for MySQL兼容MySQL数据库，AnalyticDB for MySQL中也有一个名为MySQL的内置数据库，该数据库中存储的是AnalyticDB for MySQL中的用户、权限信息以及存储过程等。您可以通过SELECT语句查询AnalyticDB for MySQL中的用户信息。

注意事项

AnalyticDB for MySQL中，只有通过高权限账号查询用户信息。

> **说明** AnalyticDB for MySQL中的高权限账号相当于MySQL中的root账号。

示例

```
USE MYSQL;
SELECT User, Host, Password FROM mysql.user;
+---------+-----+-------------------------+
| User    | Host| Password                |
+---------+-----+-------------------------+
| account1|  %  | *61f3777f02386598cd***** |
| account2|  %  | *0fe79c07e168cabc99***** |
```

3.7.3. CROSS JOIN

通过cross join语句实现列转行功能。能将某一列中的多个元组转换成多行。

数组结构转多行

将某一列转成数组，然后转成多行，示例如下：

```
####建库
CREATE DATABASE mydb;
USE mydb;
###建表
CREATE TABLE test(
userid INT
,user_name VARCHAR
,product VARCHAR
) distributed by hash(userid);
###插入两行测试数据
INSERT INTO test VALUES
(1,'aaa','cat,mat,bat'),(2,'bbb','dog,pog,fog');
####查询数据，产品列转行，其中temp_table为临时表名可以更换，col为返回列名
SELECT userid, col
FROM (select userid, split(product,',') as numbers_array from test)
CROSS JOIN UNNEST(numbers_array) as temp_table(col);
###查询结果
userid col
1    cat
1    mat
1    bat
2    dog
2    pog
2    fog
```

3.8. DCL

3.8.1. CREATE USER

`CREATE USER` 用于创建账号。

```
CREATE USER
    [if not exists] user [auth_option] [, [if not exists] user [auth_option]] ...
```

注意事项

通过 `CREATE USER` 创建账号时，您需要拥有 `CREATE_USER` 权限。

示例

创建账号account2，密码为Account2。

```
CREATE USER if not exists 'account2' IDENTIFIED BY 'Account2';
```

3.8.2. DROP USER

`DROP USER` 用于删除用户。

语法

```
DROP USER [if exists] user [, [if exists] user] ...
```

注意事项

通过 `DROP USER` 删除用户时，您需要拥有 `CREATE_USER` 权限。

示例

```
DROP USER account_2;
```

3.8.3. GRANT

`GRANT` 用于为用户授权。

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON priv_level
    TO user [auth_option]
    [WITH {GRANT OPTION}]
```

参数

- `priv_type`：权限类型，详情请参见权限模型。
- `column_list`：可选参数，当 `priv_type` 为 `SELECT` 时，可以填写表中的列名，针对具体列授予 `SELECT` 授权。
- `priv_level`：被授权对象层级。
  - `*.*`：整个集群级别的权限。
  - `db_name.*`：数据库级别的权限。
  - `db_name.table_name` 或者 `table_name`：表级别的权限。

注意事项

通过 `GRANT` 授权用户时，您需要拥有 `GRANT OPTION` 权限。

示例

- 为账号account2授予集群级别的 `all` 权限。

  ```
  GRANT all ON *.* TO 'account2';
  ```

- 为账号account3授予数据库级别的 `all` 权限。

  ```
  GRANT all ON adb_demo.* TO 'account3';
  ```

- 可以通过 `GRANT` 创建并授权账号。

  例如，创建全局DML账号。

  ```
  GRANT insert,select,update,delete on *.* to 'test'@'%' identified by 'Testpassword1';
  ```

  创建数据库级别DML账号。

```
GRANT insert,select,update,delete on adb_demo.* to 'test123' identified by 'Testpassword123';
```

- 创建账号并授予列级别的 `SELECT` 权限。

```
GRANT select (customer_id, sex) ON customer TO 'test321' identified by 'Testpassword321';
```

3.8.4. RENAME USER

`RENAME USER` 用于更改用户名。

语法

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

示例

```
RENAME USER account2 TO account_2;
SELECT User, Host, Password FROM mysql.user;
+----------+-----+----------------------- -+
| User     | Host| Password                |
+----------+-----+-------------------------+
| account2 | %   | *61f3777f02386598cda****|
| account_2| %   | *0fe79c07e168cabc99b****|
```

3.8.5. REVOKE

`REVOKE` 用于撤销用户权限。

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user
```

参数

- `priv_type`：权限类型，详情请参见权限模型。
- `column_list`：可选参数，当 `priv_type` 为 `SELECT` 时，可以填写表中的列名，针对具体列撤销 `SELECT` 权限。
- `priv_level`：被撤销权限的对象层级。
  - `*.*`：整个集群级别的权限。
  - `db_name.*`：数据库级别的权限。
  - `db_name.table_name` 或者 `table_name`：表级别的权限。

注意事项

通过 `REVOKE` 撤销用户权限时，您需要拥有 `GRANT OPTION` 权限。

示例

撤销账号account3数据库级别的 `all` 权限。

```
REVOKE all ON adb_demo.* FROM 'account3';
```

3.9. Build任务

Build任务用于将实时写入的数据转换为历史数据格式。在该过程中会构建索引、清理冗余数据、执行异步DDL任务等，将数据从写优化转变为读优化。

功能说明

`build` 命令可以对实时写入的数据及其涉及到的历史分区，进行合并、构建索引、执行异步DDL等操作。如果用户通过INSERT/UPDATE/DELETE等修改分区的数据，那么该分区需要执行Build，与实时数据结合后，生成新的分区。如果用户没有修改分区的数据，则该分区不会被Build任务修改。

Build任务开始后，不能停止任务。

当前Build任务是以表为粒度进行调度的。Build任务开始执行后，表级的Build任务会切分为以Shard为粒度的Build子任务，每个Shard的3个副本都有一个Build子任务，当所有的子任务都执行完成后，Build任务完成。不同表的Build任务可以并行执行，并行度默认为32。

自动触发BUILD任务

自动触发Build任务需要满足以下其中一个条件：

- 达到最小Build时间间隔（预留模式：0.5小时，弹性模式：1.5小时），且单表满足新增数据达到5W条。
- 距离上次Build，时间已过去24小时且至少修改1行数据。

手动触发BUILD任务

可通过以下两种方式手动触发BUILD任务：

- **build**

```
build table <table_name>;
```

- **force build**

  Build只会修改变动的分区，而 `force build` 命令可以对所有分区进行强制Build。

```
build table <table_name> force = true;
```

> 注意 在表较大时，`force build` 的执行时间会很长，尤其聚集索引需要对分区排序，执行需要慎重。

3.10. 元数据库数据字典

云原生数据仓库AnalyticDB MySQL版的元数据库为INFORMATION_SCHEMA库，兼容MySQL的元数据库。查询元数据库可以直接在JDBC连接中使用SQL语句进行查询。查询test库下的所有表，示例如下：

```
select * from TABLES where table_schema='test'
```

SCHEMATA

SCHEMATA表提供了关于数据库的信息。

FIELD	TYPE	ALLOW_NULL	DEFAULT	COMMENT
CATALOG_NAME	varchar(16)	YES	NULL	CATALOG名称
SCHEMA_NAME	varchar(64)	NO	NULL	SCHEMA名称
DEFAULT_CHARACTER_SET_NAME	varchar(64)	YES	UTF-8	默认字符集
DEFAULT_COLLATION_NAME	varchar(64)	YES	OFF	默认排序规则
SQL_PATH	varchar(255)	YES	NULL	SQL路径

TABLES

TABLES表提供数据库表信息。该部分数据包括表的元数据与部分表对应数据的元数据，如分区信息等。

FIELD	TYPE	ALLOW_NULL	DEFAULT	COMMENT
TABLE_CATALOG	varchar(512)	NO	NULL	固定值'def'
TABLE_SCHEMA	varchar(64)	NO	NULL	所属SCHEMA的名称
TABLE_NAME	varchar(64)	NO	NULL	表名称
TABLE_TYPE	varchar(64)	YES	NULL	标记：分区表PARTITION_TABLE、维度表DIMENSION_TABLE
ENGINE	varchar(64)	YES	NULL	引擎类型
VERSION	bigint(21)	YES	NULL	tableId
ROW_FORMAT	varchar(20)	YES	NULL	固定值'Compact'
TABLE_ROWS	bigint(21)	YES	NULL	记录条数
AVG_ROW_LENGTH	bigint(21)	YES	NULL	未使用，为null
DATA_LENGTH	bigint(21)	YES	NULL	数据大小
MAX_DATA_LENGTH	bigint(21)	YES	NULL	未使用，为null
INDEX_LENGTH	bigint(21)	YES	NULL	索引大小
DATA_FREE	bigint(21)	YES	NULL	未使用，为null
AUTO_INCREMENT	bigint(21)	YES	NULL	表的自增值（未使用，为null）
CREATE_TIME	datetime	YES	NULL	创建时间
UPDATE_TIME	datetime	YES	NULL	更新时间
CHECK_TIME	datetime	YES	NULL	未使用，为null

FIELD	TYPE	ALLOW_NULL	DEFAULT	COMMENT
TABLE_COLLATION	varchar(32)	YES	NULL	固定值“utf8_bin”
CHECKSUM	bigint(21)	YES	NULL	未使用，为null
CREATE_OPTIONS	varchar(255)	YES	NULL	未使用，为null
TABLE_COMMENT	varchar(255)	YES	NULL	表注释

- 如果不知道具体的表名，只是希望了解某一个SCHEMA下有多少张不同类型的表，可以查询这张表。例如：

```
SELECT DISTINCT TABLE_GROUP, TABLE_NAME FROM information_schema.tables WHERE TABLE_SCHEMA = 'xxx';
```

- 如果希望了解某一个SCHEMA下面哪些是视图，可以查询这张表。例如：

```
SELECT TABLE_GROUP, TABLE_NAME FROM information_schema.tables WHERE TABLE_SCHEMA = 'xxx' AND TABLE_TYPE = 'VIEW';
```

COLUMNS

该表存储了所有的表中字段的详细信息。

FIELD	TYPE	ALLOW_NULL	DEFAULT	COMMENT
TABLE_CATALOG	varchar(8)	YES	NULL	CATALOG名称
TABLE_SCHEMA	varchar(64)	NO	NULL	所属SCHEMA
TABLE_NAME	varchar(64)	NO	NULL	所属表名
COLUMN_NAME	varchar(64)	NO	NULL	列名
ORDINAL_POSITION	bigint(21)	YES	NULL	在表中的位置
COLUMN_DEFAULT	varchar(255)	YES	NULL	默认列
IS_NULLABLE	tinyint(1)	YES	1	是否允许空
DATA_TYPE	bigint(21)	YES	NULL	数据类型名称
CHARACTER_MAXIMUM_LENGTH	bigint(21)	YES	NULL	字符最大长度
CHARACTER_OCTET_LENGTH	bigint(21)	YES	NULL	字符八进制长度
NUMERIC_PRECISION	int(11)	YES	NULL	数值精度
NUMERIC_SCALE	bigint(21)	YES	NULL	数值范围
DATETIME_PRECISION	bigint(21)	YES	NULL	时间精度
CHARACTER_SET_NAME	varchar(32)	YES	NULL	字符集名称
COLLATION_NAME	varchar(32)	YES	NULL	排序规则名称
COLUMN_TYPE	varchar(64)	YES	NULL	列类型
COLUMN_KEY	varchar(3)	NO	NULL	索引类型
EXTRA	varchar(30)	NO	NULL	是否on update
PRIVILEGES	varchar(80)	NO	NULL	固定值：select,insert,update,references
COLUMN_COMMENT	varchar(1024)	1024	NULL	列注释

如果希望了解某一个表包含的所有列信息，可以根据TABLE_SCHEMA和TABLE_NAME查询需要的列信息。

4.系统函数

4.1. 条件判断函数

本文介绍AnalyticDB for MySQL中的条件判断函数。

- CASE
- IF
- IFNULL
- NULLIF

本文中的条件判断函数以 `conditiontest` 表为测试数据。

```
create table conditiontest(a int) distributed by hash(a);
```

```
insert into conditiontest values (1),(2),(3);
```

```
SELECT * FROM conditiontest;
+---+
| a |
+---+
| 2 |
| 1 |
| 3 |
```

CASE

```
CASE expression
    WHEN value THEN result
    [ WHEN ... ]
    [ ELSE result ]
    END
```

- 命令说明：简单 `CASE` 表达式会从左到右依次查找 `value` ，直到找到和 `expression` 相等的 `value` ，并返回对应的 `result` 结果；如果没有找到相等的 `value` ，则返回 `ELSE` 语句后的 `result` 结果。
- 示例：

```
SELECT a,
      CASE a
      WHEN 1 THEN 'one'
      WHEN 2 THEN 'two'
      ELSE 'three'
      END as caseresult
FROM conditiontest;
+---+------------+
| a | caseresult |
+---+------------+
| 2 | two        |
| 1 | one        |
| 3 | three      |
```

```
CASE
    WHEN condition THEN result
    [ WHEN ... ]
    [ ELSE result ]
    END
```

- 命令说明：高级 `CASE` 表达式会从左到右依次计算 `condition` ，直到第一个为 `TRUE` 的 `condition` ，并返回对应的 `result` 结果；如果没有找到为 `TRUE` 的 `condition` ，则返回 `ELSE` 语句后的 `result` 结果。
- 示例：

```
SELECT a,
      CASE a
      WHEN a=1 THEN 'one1'
      WHEN a=2 THEN 'two2'
      ELSE 'three3'
      END as caseresult
FROM conditiontest;
+---+------------+
| a | caseresult |
+---+------------+
| 1 | one1       |
| 3 | three3     |
| 2 | three3     |
```

IF

```
if(condition, true_value)
```

- 命令说明：如果 `condition` 为 `true`，结果返回 `true_value`；否则返回 `null`。
- 示例：

```
SELECT IF((2+3)>4,5);
+--------+
| _col0 |
+--------+
|     5 |
```

```
if(condition, true_value, false_value
```

- 命令说明：如果 `condition` 为 `true`，结果返回 `true_value`；否则结果返回 `false_value`。
- 示例：

```
SELECT IF((2+3)<5,5, 6);
+--------+
| _col0 |
+--------+
|     6 |
```

IFNULL

```
IFNULL(expr1,expr2)
```

- 命令说明：如果 `expr1` 结果不为空，则返回 `expr1` 的值；否则返回 `expr2` 的值。
- 示例：

```
SELECT IFNULL(NULL,2);
+--------+
| _col0 |
+--------+
|     2 |
SELECT IFNULL(1,0);
+--------+
| _col0 |
+--------+
|     1 |
```

NULLIF

```
NULLIF(expr1,expr2)
```

- 命令说明：如果 `expr1` 与 `expr2` 值相等，结果返回 `null`；否则结果返回 `expr1` 的值。
- 示例：

```
SELECT NULLIF (2,1);
+--------+
| _col0 |
+--------+
|     2 |
SELECT NULLIF (2,2);
+--------+
| _col0 |
+--------+
| NULL  |
```

4.2. 数值函数和运算符

4.2.1. 算术运算符

AnalyticDB for MySQL支持以下算术运算符。

+	加
-	减
*	乘
/	除法
DIV	除法（整数视角）
%或MOD	取模
-	改变参数符号

+

- 命令说明：加法。
- 示例：

```
select 3+5;
+-------+
| _col0 |
+-------+
|     8 |
```

```
select 3+2.9875;
+---------+
| _col0   |
+---------+
| 5.9875 |
```

-

- 命令说明：减法。
- 示例：

```
select 3-5;
+-------+
| _col0 |
+-------+
|    -2 |
```

```
select 3-1.5;
+-----------+
| _col0     |
+-----------+
| 1.5       |
```

*****

- 命令说明：乘法。
- 示例：

```
select 3*pi();
+------------------+
| _col0            |
+------------------+
| 9.42477796076938 |
```

/

- 命令说明：除法。
- 示例：

```
select 3/pi();
+--------------------+
| _col0              |
+--------------------+
| 0.954929658551372 |
```

DIV

- 命令说明：除法，从除法结果中舍弃小数点右侧的小数部分。
- 示例：

```
select 3 div pi();
+--------+
| _col0 |
+--------+
|     0 |
```

```
select 33 div 2;
+--------+
| _col0 |
+--------+
|    16 |
```

%或MOD

- 命令说明：返回两个参数除法后的余数。
- 示例：

```
select 3 mod pi();
+--------+
| _col0 |
+--------+
|   3.0 |
```

```
select 33 % 2;
+--------+
| _col0 |
+--------+
|     1 |
```

-

- 命令说明：将正数变为负数或者将负数变为正数。
- 示例：

```
select - 2;
+--------+
| _col0 |
+--------+
|    -2 |
```

```
select - 2;
+--------+
| _col0 |
+--------+
|    -2 |
```

4.2.2. 数值函数

本文介绍AnalyticDB MySQL版集群支持的数值函数。

- ABS：返回参数的绝对值。
- ACOS：返回参数的反余弦值。
- ASIN：返回参数的反正弦值。
- ATAN：返回参数的反正切值。
- ATAN2：返回参数x除以参数y之后的反正切值。
- CEILING或CEIL：返回大于等于参数x，且最接近x的整数值。
- COS：返回参数的余弦值。
- COT：返回参数的余切值。
- CRC32：返回参数的循环冗余码。
- DEGREES：弧度转换为角度度。
- EXP：返回以e为底、x为幂的值。
- FLOOR：返回小于等于参数x，且最接近x的整数值。
- LN：返回参数的自然对数。
- LOG：对数函数。
- LOG2：返回以2为底的对数。
- LOG10：返回以10为底的对数。
- PI：返回圆周率pi的值。
- POWER/POW：返回x的y次幂。
- RADIANS：角度转换为弧度。
- RAND返回目标数字范围内的一个随机数。

- ROUND：返回参数四舍五入后的值。
- SIGN：返回参数的符号的值。
- SIN：返回参数的正弦值。
- SQRT：返回参数的平方根。
- TAN：返回参数的正切值。

ABS

```
abs(x)
```

- 命令说明：返回参数 `x` 的绝对值。
- 输入值类型：TINYINT、SMALLINT、INT、BIGINT、DOUBLE、FLOAT或DECIMAL。
- 返回值类型：
  - 当输入值类型为TINYINT、SMALLINT、INT或BIGINT时，返回值类型为BIGINT。
  - 当输入值类型为DOUBLE或FLOAT时，返回值类型为DOUBLE。
  - 当输入值类型为DECIMAL时，返回值类型为DECIMAL。
- 示例：
  - 语句如下：

    ```
    SELECT abs(4.5);
    ```

    返回结果如下：

    ```
    +----------+
    | abs(4.5) |
    +----------+
    |      4.5 |
    +----------+
    ```

  - 语句如下：

    ```
    SELECT abs(-4);
    ```

    返回结果如下：

    ```
    +--------+
    | abs(4) |
    +--------+
    |      4 |
    +--------+
    ```

ACOS

```
acos(x)
```

- 命令说明：返回参数 `x` 的反余弦值。

  如果 `x>1` 或者 `x<-1` ，返回结果为 `NULL` 。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

  ```
  SELECT acos(0.5);
  ```

  返回结果如下：

  ```
  +--------------------+
  | acos(0.5)          |
  +--------------------+
  | 1.0471975511965979 |
  +--------------------+
  ```

ASIN

```
asin(x)
```

- 命令说明：返回参数 `x` 的反正弦值。

  如果 `x>1` 或者 `x<-1` ，返回结果为 `NULL` 。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT asin(0.5);
```

返回结果如下：

```
+--------------------+
| asin(0.5)          |
+--------------------+
| 0.5235987755982989 |
+--------------------+
```

ATAN

```
atan(x)
```

- 命令说明：返回参数 `x` 的反正切值。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT atan(0.5);
```

返回结果如下：

```
+--------------------+
| atan(0.5)          |
+--------------------+
| 0.4636476090008061 |
+--------------------+
```

ATAN2

```
atan2(x, y)
atan(x, y)
```

- 命令说明：返回参数 `x` 除以参数 `y` 之后的反正切值。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT atan2(0.5,0.3);
```

```
+--------------------+
| atan2(0.5,0.3)     |
+--------------------+
| 1.0303768265243125 |
+--------------------+
```

CEILING或CEIL

```
ceiling(x)
ceil(x)
```

- 命令说明：返回大于或等于 `x` ，且最接近 `x` 的整数。
- 输入值类型：TINYINT、SMALLINT、INT、BIGINT、DOUBLE、FLOAT或DECIMAL。
- 返回值类型：
  - 当输入值类型为TINYINT、SMALLINT、INT或BIGINT时，返回值类型为BIGINT。
  - 当输入值类型为DOUBLE或FLOAT时，返回值类型为DOUBLE。
  - 当输入值类型为DECIMAL时，返回值类型为DECIMAL。
- 示例：
  - 语句如下：

    ```
    SELECT ceiling(4);
    ```

    返回结果如下：

    ```
    +------------+
    | ceiling(4) |
    +------------+
    |          4 |
    +------------+
    ```

- 语句如下：

```
SELECT ceil(-4.5);
```

返回结果如下：

```
+------------+
| ceil(-4.5) |
+------------+
|         -4 |
+------------+
```

COS

```
cos(x)
```

- 命令说明：返回 `x` 的余弦值。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT cos(1.3);
```

返回结果如下：

```
+---------------------+
| cos(1.3)            |
+---------------------+
| 0.26749882862458735 |
+---------------------+
```

COT

```
cot(x)
```

- 命令说明：返回参数 `x` 的余切值。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT cot(1.234);
```

返回结果如下：

```
+---------------------+
| cot(1.234)          |
+---------------------+
| 0.35013639786701445 |
+---------------------+
```

CRC32

```
crc32(x)
```

- 命令说明：返回参数 `x` 的循环冗余码。
- 输入值类型：VARBINARY。
- 返回值类型：BIGINT。
- 示例：

```
SELECT crc32('中国');
```

返回结果如下：

```
+-----------------+
| crc32('中国')   |
+-----------------+
|       737014929 |
+-----------------+
```

DEGREES

```
degrees(x)
```

- 命令说明：弧度转换为角度。
- 输入值类型：DOUBLE。

- 返回值类型：DOUBLE。
- 示例：

```
SELECT degrees(1.3);
```

返回结果如下：

```
+-------------------+
| degrees(1.3)      |
+-------------------+
| 74.48451336700703 |
+-------------------+
```

EXP

```
exp(x)
```

- 命令说明：返回以 `e` 为底， `x` 为幂的值。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT exp(4.5);
```

返回结果如下：

```
+-------------------+
| exp(4.5)          |
+-------------------+
| 90.01713130052181 |
+-------------------+
```

FLOOR

```
floor(x)
```

- 命令说明：返回小于或等于 `x` ，且最接近 `x` 的整数。
- 输入值类型：TINYINT、SMALLINT、INT、BIGINT、DOUBLE、FLOAT或DECIMAL。
- 返回值类型：
  - 当输入值类型为TINYINT、SMALLINT、INT或BIGINT时，返回值类型为BIGINT。
  - 当输入值类型为DOUBLE或FLOAT时，返回值类型为DOUBLE。
  - 当输入值类型为DECIMAL时，返回值类型为DECIMAL。
- 示例：
  - 语句如下：

    ```
    SELECT floor(7);
    ```

    返回结果如下：

    ```
    +----------+
    | floor(7) |
    +----------+
    |        7 |
    +----------+
    ```

  - 语句如下：

    ```
    SELECT floor(-3.5);
    ```

    返回结果如下：

    ```
    +-------------+
    | floor(-3.5) |
    +-------------+
    |          -4 |
    +-------------+
    ```

LN

```
ln(x)
```

- 命令说明：返回 `x` 的自然对数。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。

- 示例：

```
SELECT ln(2.718281828459045);
```

返回结果如下：

```
+------------------------+
| ln(2.718281828459045) |
+------------------------+
|                    1.0 |
+------------------------+
```

LOG

```
log(x)
log(x, y)
```

- 命令说明：
  - 输入一个参数时，返回 `x` 的自然对数。
  - 输入两个参数时，返回以 `x` 为底的 `y` 的对数。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：
  - 语句如下：

    ```
    SELECT log(16);
    ```

    返回结果如下：

    ```
    +--------------------+
    | log(16)            |
    +--------------------+
    | 2.772588722239781 |
    +--------------------+
    ```

  - 语句如下：

    ```
    SELECT log(10,100);
    ```

    返回结果如下：

    ```
    +-------------+
    | log(10,100) |
    +-------------+
    |         2.0 |
    +-------------+
    ```

LOG2

```
log2(x)
```

- 命令说明：返回以2为底的对数。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT log2(8.7654);
```

返回结果如下：

```
+--------------------+
| log2(8.7654)       |
+--------------------+
| 3.131819928389146 |
+--------------------+
```

LOG10

```
log10(x)
```

- 命令说明：返回以10位底的对数。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT log10(100.876);
```

返回结果如下：

```
+--------------------+
| log10(100.876)     |
+--------------------+
| 2.0037878529824615 |
+--------------------+
```

PI

```
pi()
```

- 命令说明：返回圆周率pi的值。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT pi();
```

返回结果如下：

```
+-------------------+
| pi()              |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

POWER/POW

```
power(x, y)
pow(x, y)
```

- 命令说明：返回 `x` 的 `y` 次幂。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：
  - 语句如下：

```
SELECT power(1.2,3.4);
```

返回结果如下：

```
+-------------------+
| power(1.2,3.4)    |
+-------------------+
| 1.858729691979481 |
+-------------------+
```

  - 语句如下：

```
SELECT pow(-2,-3);
```

返回结果如下：

```
+------------+
| pow(-2,-3) |
+------------+
|     -0.125 |
+------------+
```

RADIANS

```
radians(x)
```

- 命令说明：角度转换为弧度。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT radians(60.0);
```

返回结果如下：

```
+--------------------+
| radians(60.0)      |
+--------------------+
| 1.0471975511965976 |
+--------------------+
```

RAND

```
rand()
rand()*(y-x)+x
```

- 命令说明：随机函数 `rand()` 不支持设置入参，会随机返回0到1之间（包含0但不包含1）的一个数。

  您也可以使用 `rand()*(y-x)+x` 函数，随机返回 `x` 到 `y` 之间（包含 `x` 但不包含 `y` ）的一个数。
- 返回值类型：DOUBLE。
- 示例：
  - 语句如下：

    ```
    SELECT rand();
    ```

    返回结果如下：

    ```
    +--------------------+
    | rand()             |
    +--------------------+
    | 0.6613712594764614 |
    +--------------------+
    ```
  - 随机返回3到12之间的一个浮点数，语句如下：

    ```
    SELECT rand()*(12-3)+3;
    ```

    返回结果如下：

    ```
    +-------------------+
    | rand()*(12-3)+3   |
    +-------------------+
    | 9.073329270781976 |
    +-------------------+
    ```

ROUND

```
round(x)
round(x,d)
```

- 命令说明：将参数 `x` 四舍五入， `d` 是要保留的精度，默认 `d` 为 `0` ，舍入算法取决于 `x` 的数据类型。
  - 如果 `x` 为 `null` ，返回结果为 `NULL` 。
  - 如果 `d>0` ，则四舍五入到指定的小数位。
  - 如果 `d=0` ，则四舍五入到最接近的整数。
  - 如果 `d<0` ，则四舍五入到小数点左侧的指定位数。
- 输入值类型：TINYINT、SMALLINT、INT、BIGINT、DOUBLE、FLOAT或DECIMAL。
- 返回值类型：
  - 当输入值类型为TINYINT、SMALLINT、INT或BIGINT时，返回值类型为BIGINT。
  - 当输入值类型为DOUBLE或FLOAT时，返回值类型为DOUBLE。
  - 当输入值类型为DECIMAL时，返回值类型为DECIMAL。
- 示例：
  - 将345.983四舍五入到最近的整数，语句如下：

    ```
    SELECT round(345.983,0);
    ```

    返回结果如下：

    ```
    +------------------+
    | round(345.983,0) |
    +------------------+
    |          346.000 |
    +------------------+
    ```

- 将345.123四舍五入到小数点后一位，语句如下：

```
SELECT round(345.123,1);
```

返回结果如下：

```
+------------------+
| round(345.123,1) |
+------------------+
|          345.100 |
+------------------+
```

- 将345.984四舍五入到小数点前一位，语句如下：

```
SELECT round(344.984,-1);
```

返回结果如下：

```
+-------------------+
| round(344.984,-1) |
+-------------------+
|           340.000 |
+-------------------+
```

- 将345.984四舍五入到小数点前4位，语句如下：

```
SELECT round(345.984,-4);
```

返回结果如下：

```
+-------------------+
| round(345.984,-4) |
+-------------------+
|             0.000 |
+-------------------+
```

SIGN

```
sign(x)
```

- 命令说明：根据参数 `x` 的符号返回对应的数值：
  - 如果 `x>0` ，则返回1。
  - 如果 `x=0` ，则返回0。
  - 如果 `x<0` ，则返回-1。
- 输入值类型：TINYINT、SMALLINT、INT、BIGINT、DOUBLE、FLOAT或DECIMAL。
- 返回值类型：BIGINT。
- 示例：
  - 语句如下：

```
SELECT sign(12);
```

返回结果如下：

```
+----------+
| sign(12) |
+----------+
|        1 |
+----------+
```

  - 语句如下：

```
SELECT sign(-4.5);
```

返回结果如下：

```
+------------+
| sign(-4.5) |
+------------+
|         -1 |
+------------+
```

SIN

```
sin(x)
```

- 命令说明：返回参数 `x` 的正弦值。
- 输入值类型：DOUBLE。

- 返回值类型：DOUBLE。
- 示例：

```
SELECT sin(1.234);
```

返回结果如下：

```
+--------------------+
| sin(1.234)         |
+--------------------+
| 0.9438182093746337 |
+--------------------+
```

SQRT

```
sqrt(x)
```

- 命令说明：返回参数 `x` 的平方根。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT sqrt(4.222);
```

返回结果如下：

```
+-------------------+
| sqrt(4.222)       |
+-------------------+
| 2.054750593137766 |
+-------------------+
```

TAN

```
tan(x)
```

- 命令说明：返回参数 `x` 的正切值。
- 输入值类型：DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT tan(8);
```

返回结果如下：

```
+--------------------+
| tan(8)             |
+--------------------+
| -6.799711455220379 |
+--------------------+
```

4.3. 日期和时间函数

本文介绍AnalyticDB for MySQL中的日期和时间函数。

- ADDDATE：返回添加指定时间后的日期。
- ADDTIME：返回添加指定时间后的时间。
- CONVERT_TZ：转换时区，从from_tz转到to_tz给出的时区，并返回结果值。
- CURDATE：返回当前日期。
- CURTIME：返回当前时间。
- DATE：返回日期或日期时间表达式中的日期。
- DATE_FORMAT：按照Format指定的格式，将日期时间格式化成字符串。
- SUBDATE/DATE_SUB：返回Date减去指定INTERVAL间隔后的日期。
- DATEDIFF：返回Expr1减去Expr2后的天数。
- DAY/DAYOFMONTH：返回Date中的日，取值范围1~31。
- DAYNAME：返回日期对应的工作日的名称，例如星期一为Monday。
- DAYOFWEEK：返回日期对应的工作日索引值。
- DAYOFYEAR：返回指定日期是当年的哪一天。
- EXTRACT：返回日期或时间的单独部分，例如年、月、日、小时、分钟等。
- FROM_DAYS：根据指定的天数N，返回对应的DATE值。
- FROM_UNIXTIME：返回Unixtime时间戳。
- HOUR：返回时间中的小时。

- LAST_DAY：返回日期或者日期时间中对应月份的最后一天。
- LOCALTIME/LOCALTIMESTAMP/NOW：返回当前时间戳。
- MAKEDATE：按照参数Year和DayOfYear，返回一个日期。
- MAKETIME：按照参数Hour、Minute和Second，返回一个时间。
- MINUTE：返回时间中的分钟。
- MONTH：返回日期中的月份。
- MONTHNAME：返回日期中月份的全名。
- PERIOD_ADD：将日期格式的参数P增加N个月。
- PERIOD_DIFF：返回P1和P2之间相差的月数。
- QUARTER：返回日期在一年中的季度。
- SEC_TO_TIME：将Seconds转换为时间。
- SECOND：返回时间中的秒。
- STR_TO_DATE：按照指定日期或时间显示格式，将字符串转换为日期或日期时间类型。
- SUBTIME：返回Expr1减去Expr2后的时间。
- SYSDATE：获取系统时间。
- TIME：以字符串形式返回Expr中的时间。
- TIME_FORMAT：按照Format指定的格式，以字符串形式显示时间。
- TIME_TO_SEC：返回Time转换为秒的结果。
- TIMEDIFF：返回Expr1减去Expr2后的时间。
- TIMESTAMP：返回Expr表示的日期或日期时间。
- TIMESTAMPADD：将Interval添加到日期或日期时间表达式datetime_expr中。
- TIMESTAMPDIFF：返回日期或日期时间表达式datetime_expr1减去datetime_expr2后的结果。
- TO_DAYS：根据给定日期Date，返回自0年开始的天数。
- TO_SECONDS：根据给定的Expr，返回自0年开始的秒数。
- UNIX_TIMESTAMP：返回自1970-01-01 00:00:00 UTC以来秒数的Unix时间戳。
- UTC_DATE：返回UTC日期。
- UTC_TIME：返回UTC时间。
- UTC_TIMESTAMP：返回UTC时间戳。
- WEEK：返回日期对应的周数。
- WEEKDAY：返回日期对应的工作日。
- WEEKOFYEAR：返回日期对应的日历周。
- YEAR：返回日期中的年份。
- YEARWEEK：返回日期的年份和星期。

ADDDATE

```
ADDDATE(date,INTERVAL expr unit)
ADDDATE(expr,days)
```

- 参数类型：

```
adddate(date, INTERVAL expr unit)
adddate(timestamp, INTERVAL expr unit)
adddate(datetime, INTERVAL expr unit)
adddate(varchar, INTERVAL expr unit)
adddate(date, varchar)
adddate(date, bigint)
adddate(datetime, bigint)
adddate(datetime, varchar)
adddate(timestamp, varchar)
adddate(timestamp, bigint)
adddate(varchar, bigint)
adddate(varchar, varchar)
```

- 返回值类型：DATE。
- 命令说明：返回添加指定时间后的日期。
  - `unit` 可取值为：`second`、`minute`、`hour`、`day`、`month`、`year`、`minute_second`、`hour_second`、`hour_minute`、`day_second`、`day_minute`、`day_hour`、`year_month`。`unit` 默认值为 `day`。
  - `days`、`expr`：系统将返回 `expr` 加上 `days` 之后的结果。
- 示例：

  示例1

```
select adddate(date '2022-01-22',interval '3' day);
```

  返回结果如下。

```
+--------------------------------------------+
| adddate(DATE '2022-01-22', INTERVAL  '3' DAY)|
+--------------------------------------------+
| 2022-01-25                                 |
```

示例2

```
select adddate(timestamp '2022-01-22',interval '3' day);
```

返回结果如下。

```
+-------------------------------------------------+
| adddate(TIMESTAMP '2022-01-22', INTERVAL  '3' DAY)|
+-------------------------------------------------+
| 2022-01-25 00:00:00                             |
```

示例3

```
select adddate(datetime '2022-01-22',interval '3' day);
```

返回结果如下。

```
+-------------------------------------------------+
| adddate(DATETIME '2022-01-22', INTERVAL  '3' DAY |
+-------------------------------------------------+
| 2022-01-25 00:00:00                             |
```

示例4

```
select adddate('2022-01-22',interval '3' day);
```

返回结果如下。

```
+---------------------------------------+
| adddate('2022-01-22', INTERVAL  '3' DAY)|
+---------------------------------------+
| 2022-01-25                            |
```

示例5

```
select adddate(datetime '2022-01-22',interval '3' second);
```

返回结果如下。

```
+-----------------------------------------------------+
| adddate(DATETIME '2022-01-22', INTERVAL  '3' SECOND)|
+-----------------------------------------------------+
|                              2022-01-22 00:00:03 |
```

示例6

```
select adddate(datetime '2022-01-22',interval '3' minute);
```

返回结果如下。

```
+-----------------------------------------------------+
| adddate(DATETIME '2022-01-22', INTERVAL  '3' MINUTE)|
+-----------------------------------------------------+
|                              2022-01-22 00:03:00 |
```

示例7

```
select adddate(datetime '2022-01-22',interval '3' hour);
```

返回结果如下。

```
+-------------------------------------------------+
| adddate(DATETIME '2022-01-22', INTERVAL  '3' HOUR |
+-------------------------------------------------+
|                            2022-01-22 03:00:00 |
```

示例8

```
select adddate(datetime '2022-01-22',interval '3' month);
```

返回结果如下。

```
+-------------------------------------------------+
| adddate(DATETIME '2022-01-22', INTERVAL  '3' MONTH)|
+-------------------------------------------------+
|                              2022-04-22 00:00:00 |
```

示例9

```
select adddate(datetime '2022-01-22',interval '3' year);
```

返回结果如下。

```
+-----------------------------------------------------+
| adddate(DATETIME '2022-01-22', INTERVAL  '3' YEAR)|
+-----------------------------------------------------+
|                              2025-01-22 00:00:00 |
```

示例10

```
select adddate(datetime '2022-01-22',interval '01:01:10' hour_second) as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-22 01:01:10 |
```

示例11

```
select adddate(datetime '2022-01-22',interval '00:10' hour_minute) as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-22 00:10:00 |
```

示例12

```
select adddate(datetime '2022-01-22',interval '1 01:01:10' day_second) as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-23 01:01:10 |
```

示例13

```
select adddate(datetime '2022-01-22',interval '01:10' minute_second) as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-22 00:01:10 |
```

示例14

```
select adddate(datetime '2022-01-22',interval '1 01:01' day_minute) as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-23 01:01:00 |
```

示例15

```
select adddate(datetime '2022-01-22',interval '1 01' day_hour) as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-23 01:00:00 |
```

示例16

```
select adddate(datetime '2022-01-22 12:32:1',interval '2 2' year_month) as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2024-03-22 12:32:01 |
```

示例17

```
select adddate('2022-01-22','3');
```

返回结果如下。

```
+---------------------------+
| adddate('2022-01-22', '3')|
+---------------------------+
| 2022-01-25                |
```

示例18

```
select adddate('2022-01-22',3);
```

返回结果如下。

```
+-------------------------+
| adddate('2022-01-22', 3)|
+-------------------------+
| 2022-01-25              |
```

示例19

```
select adddate(datetime '2022-01-22 12:12:32',3);
```

返回结果如下。

```
+-----------------------------------------+
| adddate(DATETIME '2022-01-22 12:12:32', 3)|
+-----------------------------------------+
|                      2022-01-25 12:12:32 |
```

示例20

```
select adddate(datetime '2022-01-22 12:12:32','3');
```

返回结果如下。

```
+------------------------------------------+
| adddate(DATETIME '2022-01-22 12:12:32', '3')|
+------------------------------------------+
|                       2022-01-25 12:12:32 |
```

示例21

```
select adddate(timestamp '2022-01-22 12:12:32','3');
```

返回结果如下。

```
+------------------------------------------+
| adddate(TIMESTAMP '2022-01-22 12:12:32', '3')|
+------------------------------------------+
|                        2022-01-25 12:12:32 |
```

示例22

```
select adddate(timestamp '2022-01-22 12:12:32',3);
```

返回结果如下。

```
+-----------------------------------------+
| adddate(TIMESTAMP '2022-01-22 12:12:32', 3)|
+-----------------------------------------+
|                      2022-01-25 12:12:32 |
```

示例23

```
select adddate('2022-01-22 12:12:32',3);
```

返回结果如下。

```
+---------------------------------+
| adddate('2022-01-22 12:12:32', 3)|
+---------------------------------+
| 2022-01-25 12:12:32             |
```

示例24

```
select adddate('2022-01-22 12:12:32','3');
```

返回结果如下。

```
+-------------------------------------+
| adddate('2022-01-22 12:12:32', '3')|
+-------------------------------------+
| 2022-01-25 12:12:32                |
```

ADDTIME

```
ADDTIME(expr1,expr2)
```

- 命令说明：返回添加指定时间后的时间，即返回 `expr1` 增加 `expr2` 后的结果。
- 参数类型：

```
addtime(date,varchar)
addtime(time,varchar)
addtime(datetime,varchar)
addtime(timestamp,varchar)
addtime(varchar,varchar)
```

- 返回值类型：VARCHAR。
- 示例：

  示例1

```
select addtime(date '2022-01-01','01:01:01');
```

  返回结果如下。

```
+----------------------------------------+
| addtime(DATE '2022-01-01', '01:01:01') |
+----------------------------------------+
| 2022-01-01 01:01:01                    |
```

  示例2

```
select addtime(time '01:00:00','01:01:01');
```

  返回结果如下。

```
+--------------------------------------+
| addtime(TIME '01:00:00', '01:01:01') |
+--------------------------------------+
| 02:01:01                             |
```

  示例3

```
select addtime(datetime '2022-01-22 00:00:00','01:01:01');
```

  返回结果如下。

```
+-----------------------------------------------------+
| addtime(DATETIME '2022-01-22 00:00:00', '01:01:01')|
+-----------------------------------------------------+
| 2022-01-22 01:01:01                                 |
```

  示例4

```
select addtime(timestamp '2022-01-22 00:00:00','01:01:01');
```

  返回结果如下。

```
+-----------------------------------------------------+
| addtime(TIMESTAMP '2022-01-22 00:00:00', '01:01:01')|
+-----------------------------------------------------+
| 2022-01-22 01:01:01                                  |
```

  示例5

```
select addtime('2022-01-22 00:00:00','01:01:01');
```

  返回结果如下。

```
+-----------------------------------------+
| addtime('2022-01-22 00:00:00', '01:01:01')|
+-----------------------------------------+
| 2022-01-22 01:01:01                      |
```

CONVERT_TZ

```
CONVERT_TZ(dt,from_tz,to_tz)
```

- 命令说明：转换 `dt` ，从 `from_tz` 转到 `to_tz` 给出的时区，并返回结果。
- 参数类型：

```
convert_tz(varchar, varchar, varchar)
```

- 返回值类型：DATETIME。
- 示例：

  示例1

```
select convert_tz('2022-01-01 12:00:00','+00:00','+10:00');
```

  返回结果如下。

```
+-----------------------------------------------------+
| convert_tz('2022-01-01 12:00:00', '+00:00', '+10:00') |
+-----------------------------------------------------+
|                                 2022-01-01 22:00:00 |
```

  示例2

```
select convert_tz('2022-01-01 12:00:00','GMT','MET');
```

  返回结果如下。

```
+-------------------------------------------------+
| convert_tz('2022-01-01 12:00:00', 'GMT', 'MET') |
+-------------------------------------------------+
|                           2022-01-01 13:00:00 |
```

CURDATE

```
CURDATE()
```

- 命令说明：返回当前日期。
- 返回值类型：DATE。
- 示例：

```
select curdate;
```

  返回结果如下。

```
+------------+
| curdate()  |
+------------+
| 2022-01-01 |
```

CURTIME

```
CURTIME()
```

- 命令说明：返回当前时间。
- 返回值类型：TIME。
- 示例：

```
select curtime();
```

  返回结果如下。

```
+--------------+
| curtime()    |
+--------------+
| 14:39:22.109 |
```

DATE

```
DATE(expr)
```

- 命令说明：返回日期或日期时间表达式中的日期。
- 参数类型：

```
date(timestamp)
date(datetime)
date(varchar)
```

- 返回值类型：DATE。
- 示例：

  示例1

```
select date(timestamp '2022-01-01 01:02:03');
```

  返回结果如下。

```
+---------------------------------------+
| date(TIMESTAMP '2022-01-01 01:02:03') |
+---------------------------------------+
| 2022-01-01                            |
```

  示例2

```
select date(datetime '2022-01-01 01:02:03');
```

  返回结果如下。

```
+---------------------------------------+
| date(DATETIME '2022-01-01 01:02:03') |
+---------------------------------------+
| 2022-01-01                            |
```

  示例3

```
select date('2022-01-01 01:02:03');
```

  返回结果如下。

```
+------------------------------+
| date('2022-01-01 01:02:03') |
+------------------------------+
| 2022-01-01                  |
```

DATE_FORMAT

```
DATE_FORMAT(date,format)
```

- 命令说明：按照 `format` 指定的格式，将日期时间格式化成字符串。`format` 格式如下所示。

%a	星期的英文缩写名称（Sun~Sat）
%b	月份的英文缩写（Jan~Dec）
%c	月，数字（0~12）
%d	每月的第几天，数字（00~31）
%e	每月的第几天，数字（0~31）
%f	微秒（000000~999999）
%H	小时（00~23）
%h	小时（01~12）
%I	小时（01~12）
%i	分钟，数字（00~59）
%j	一年中的第几天（001~366）
%k	小时（0~23）
%l	小时（1~12）
%M	月份的英文全称（January~December）
%m	月，数字（00~12）
%p	AM或PM
%r	时间，12小时制（hh:mm:ss AM或PM）
%S	秒（00~59）
%s	秒（00~59）
%T	时间，24小时（hh:mm:ss）

%v	本周是当年的第几周，等同于WEEK()模式3；与%x 使用 说明 星期一作为一周的第一天。
%W	星期几（Sunday~Saturday）
%x	本周所属年份，四位数；常与%v一同使用 说明 星期一作为一周的第一天。
%Y	年份，数字，四位数
%y	年份，数字，两位数
%%	%字符

- 参数类型：

```
date_format(timestamp, varchar)
date_format(varchar, varchar)
date_format(datetime, varchar)
date_format(date, varchar)
```

- 返回值类型：VARCHAR。
- 示例：

  示例1

```
select date_format(timestamp '2022-01-27 13:23:00', '%W %M %Y')as result;
```

  返回结果如下。

```
+-----------------------+
| result                |
+-----------------------+
| Thursday January 2022 |
```

  示例2

```
select date_format('2022-01-27 13:23:00', '%W %M %Y')as result;
```

  返回结果如下。

```
+-----------------------+
| result                |
+-----------------------+
| Thursday January 2022 |
```

  示例3

```
select date_format(datetime '2022-01-27 13:23:00', '%W %M %Y')as result;
```

  返回结果如下。

```
+-----------------------+
| result                |
+-----------------------+
| Thursday January 2022 |
```

  示例4

```
select date_format(date '2022-01-27', '%W %M %Y')as result;
```

  返回结果如下。

```
+-----------------------+
| result                |
+-----------------------+
| Thursday January 2022 |
```

SUBDATE/DATE_SUB

```
DATE_SUB(date,INTERVAL expr unit)
```

- 命令说明：返回 `date` 减去指定 `INTERVAL` 间隔后的日期。

  `unit` 可取值为：`second` 、`minute` 、`hour` 、`day` 、`month` 、`year` 、`minute_second` 、`hour_second` 、`hour_minute` 、`day_second` 、`day_minute` 、`day_hour` 、`year_month` 。`unit` 默认值为 `day` 。

- 参数类型：

```
subdate(date, INTERVAL expr unit)
subdate(timestamp, INTERVAL expr unit)
subdate(datetime, INTERVAL expr unit)
subdate(varchar, INTERVAL expr unit)
subdate(date, bigint)
subdate(date, varchar)
subdate(datetime, bigint)
subdate(datetime, varchar)
subdate(timestamp, bigint)
subdate(timestamp, varchar)
subdate(varchar, bigint)
subdate(varchar, varchar)
```

- 返回值类型：DATE。
- 示例：

  示例1

```
select date_sub(date '2022-01-22',interval '3' day);
```

  返回结果如下。

```
+-----------------------------------------------+
| date_sub(DATE '2022-01-22', INTERVAL  '3' DAY)|
+-----------------------------------------------+
| 2022-01-19                                    |
```

  示例2

```
select date_sub(timestamp '2022-01-22 00:00:00',interval '3' day)as result;
```

  返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-19 00:00:00 |
```

  示例3

```
select date_sub(datetime '2022-01-22 00:00:00',interval '3' day)as result;
```

  返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-19 00:00:00 |
```

  示例4

```
select date_sub('2022-01-22 00:00:00',interval '3' day);
```

  返回结果如下。

```
+-------------------------------------------------+
| date_sub('2022-01-22 00:00:00', INTERVAL  '3' DAY)|
+-------------------------------------------------+
| 2022-01-19 00:00:00                             |
```

  示例5

```
select date_sub('2022-01-22 00:00:00',interval '3' second);
```

  返回结果如下。

```
+---------------------------------------------------+
| date_sub('2022-01-22 00:00:00', INTERVAL  '3' SECOND)|
+---------------------------------------------------+
| 2022-01-21 23:59:57                               |
```

  示例6

```
select date_sub('2022-01-22 00:00:00',interval '3' minute);
```

  返回结果如下。

```
+---------------------------------------------------+
| date_sub('2022-01-22 00:00:00', INTERVAL  '3' MINUTE)|
+---------------------------------------------------+
| 2022-01-21 23:57:00                               |
```

示例7

```
select date_sub('2022-01-22 00:00:00',interval '3' hour);
```

返回结果如下。

```
+-----------------------------------------------------+
| date_sub('2022-01-22 00:00:00', INTERVAL  '3' HOUR)|
+-----------------------------------------------------+
| 2022-01-21 21:00:00                                 |
```

示例8

```
select date_sub('2022-01-22 00:00:00',interval '3' month);
```

返回结果如下。

```
+-----------------------------------------------------+
| date_sub('2022-01-22 00:00:00', INTERVAL  '3' MONTH)|
+-----------------------------------------------------+
| 2021-10-22 00:00:00                                  |
```

示例9

```
select date_sub('2022-01-22 00:00:00',interval '3' year);
```

返回结果如下。

```
+-----------------------------------------------------+
| date_sub('2022-01-22 00:00:00', INTERVAL  '3' YEAR)|
+-----------------------------------------------------+
| 2019-01-22 00:00:00                                 |
```

示例10

```
select date_sub('2022-01-22 00:00:00',interval '01:10' minute_second)as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-21 23:58:50 |
```

示例11

```
select date_sub('2022-01-22 00:00:00',interval '01:01:10' hour_second)as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-21 22:58:50 |
```

示例12

```
select date_sub('2022-01-22 00:00:00',interval '01:01' hour_minute)as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-21 22:59:00 |
```

示例13

```
select date_sub('2022-01-22 00:00:00',interval '1 01:01:10' day_second)as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-20 22:58:50 |
```

示例14

```
select date_sub('2022-01-22 00:00:00',interval '1 01:01' day_minute)as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-20 22:59:00 |
```

示例15

```
select date_sub('2022-01-22 00:00:00',interval '1 01' day_hour)as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-20 23:00:00 |
```

示例16

```
select date_sub('2022-01-22 00:00:00',interval '2 2' year_month)as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2019-11-22 00:00:00 |
```

示例17

```
select date_sub(date '2022-01-22 00:00:00',3);
```

返回结果如下。

```
+---------------------------------------+
| date_sub(DATE '2022-01-22 00:00:00', 3)|
+---------------------------------------+
| 2022-01-19                            |
```

示例18

```
select date_sub(date '2022-01-22 00:00:00','3');
```

返回结果如下。

```
+----------------------------------------+
| date_sub(DATE '2022-01-22 00:00:00', '3')|
+----------------------------------------+
| 2022-01-19                             |
```

示例19

```
select date_sub(datetime '2022-01-22 00:00:00',3);
```

返回结果如下。

```
+-----------------------------------------+
| date_sub(DATETIME '2022-01-22 00:00:00', 3)|
+-----------------------------------------+
|                      2022-01-19 00:00:00 |
```

示例20

```
select date_sub(datetime '2022-01-22 00:00:00','3');
```

返回结果如下。

```
+-----------------------------------------+
| date_sub(DATETIME '2022-01-22 00:00:00', '3')|
+-----------------------------------------+
|                       2022-01-19 00:00:00 |
```

示例21

```
select date_sub(timestamp '2022-01-22 00:00:00',3);
```

返回结果如下。

```
+-----------------------------------------+
| date_sub(TIMESTAMP '2022-01-22 00:00:00', 3)|
+-----------------------------------------+
|                      2022-01-19 00:00:00 |
```

示例22

```
select date_sub(timestamp '2022-01-22 00:00:00','3');
```

返回结果如下。

```
+-----------------------------------------------+
| date_sub(TIMESTAMP '2022-01-22 00:00:00', '3')|
+-----------------------------------------------+
|                          2022-01-19 00:00:00 |
```

示例23

```
select date_sub('2022-01-22 00:00:00',3);
```

返回结果如下。

```
+-----------------------------------+
| date_sub('2022-01-22 00:00:00', 3)|
+-----------------------------------+
| 2022-01-19 00:00:00               |
```

示例24

```
select date_sub('2022-01-22 00:00:00','3');
```

返回结果如下。

```
+--------------------------------------+
| date_sub('2022-01-22 00:00:00', '3')|
+--------------------------------------+
| 2022-01-19 00:00:00                |
```

DATEDIFF

```
DATEDIFF(expr1,expr2)
```

- 命令说明：返回 `expr1` 减去 `expr2` 后的天数。
- 参数类型：

```
datediff(varchar, varchar)
datediff(datetime, varchar)
datediff(varchar, datetime)
datediff(datetime, datetime)
datediff(varchar, timestamp)
datediff(timestamp, timestamp)
datediff(timestamp, varchar)
datediff(date, date)
datediff(date, varchar)
datediff(varchar, date)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select datediff('2022-01-22 23:59:59','2022-1-21');
```

返回结果如下。

```
+-----------------------------------------------+
| datediff('2022-01-22 23:59:59','2022-1-21')   |
+-----------------------------------------------+
|                                           1 |
```

示例2

```
select datediff(datetime '2022-01-22 23:59:59','2022-1-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例3

```
select datediff('2022-01-22 23:59:59',datetime '2022-1-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例4

```
select datediff(datetime '2022-01-22 23:59:59',datetime '2022-1-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例5

```
select datediff('2022-01-22 23:59:59',timestamp '2022-1-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例6

```
select datediff(timestamp '2022-01-22 23:59:59',timestamp '2022-1-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例7

```
select datediff(timestamp '2022-01-22 23:59:59','2022-1-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例8

```
select datediff(date '2022-01-22 23:59:59',date '2022-01-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例9

```
select datediff(date '2022-01-22 23:59:59','2022-01-21')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|      1 |
```

示例10

```
select datediff('2022-01-22',date '2021-01-21');
```

返回结果如下。

```
+-----------------------------------------+
| datediff('2022-01-22', DATE '2021-01-21') |
+-----------------------------------------+
|                                     366 |
```

DAY/DAYOFMONTH

```
DAY(date)
DAYOFMONTH(date)
```

- 命令说明：返回 `date` 中的日，取值范围 `[1,31]` 。
- 参数类型：

```
dayofmonth(timestamp)
dayofmonth(datetime)
dayofmonth(date)
dayofmonth(time)
dayofmonth(varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select dayofmonth(timestamp '2022-01-22 12:23:09');
```

  返回结果如下。

```
+--------------------------------------------+
| dayofmonth(TIMESTAMP '2022-01-22 12:23:09') |
+--------------------------------------------+
|                                         22 |
```

  示例2

```
select dayofmonth(date '2022-01-22');
```

  返回结果如下。

```
+-------------------------------+
| dayofmonth(DATE '2022-01-22') |
+-------------------------------+
|                            22 |
```

  示例3

```
select dayofmonth(time '17:01:10');
```

  返回当前日期，结果如下。

```
+-----------------------------+
| dayofmonth(TIME '17:01:10') |
+-----------------------------+
|                          22 |
```

  示例4

```
select dayofmonth(datetime '2022-01-22 00:00:00');
```

  返回结果如下。

```
+-----------------------------------------+
| dayofmonth(DATETIME '2022-01-22 00:00:00') |
+-----------------------------------------+
|                                      22 |
```

  示例5

```
select day('2022-01-22');
```

  返回结果如下。

```
+-------------------+
| day('2022-01-22') |
+-------------------+
|                22 |
```

DAYNAME

```
DAYNAME(date)
```

- 命令说明：返回日期对应的星期几的名称，例如星期一为 `Monday` 。
- 参数类型：

```
dayname(timestamp)
dayname(datetime)
dayname(date)
dayname(varchar)
```

- 返回值类型：VARCHAR。
- 示例：

  示例1

```
select dayname(timestamp '2022-01-22 00:00:00');
```

  返回结果如下。

```
+-----------------------------------------+
| dayname(TIMESTAMP '2022-01-22 00:00:00') |
+-----------------------------------------+
| Saturday                                |
```

  示例2

```
select dayname(datetime '2022-01-22 00:00:00');
```

  返回结果如下。

```
+-----------------------------------------+
| dayname(DATETIME '2022-01-22 00:00:00') |
+-----------------------------------------+
| Saturday                                |
```

  示例3

```
select dayname(date '2022-01-22');
```

  返回结果如下。

```
+----------------------------+
| dayname(DATE '2022-01-22') |
+----------------------------+
| Saturday                   |
```

  示例4

```
select dayname('2022-01-22');
```

  返回结果如下。

```
+-----------------------+
| dayname('2022-01-22') |
+-----------------------+
| Saturday              |
```

DAYOFWEEK

```
DAYOFWEEK(date)
```

- 命令说明：返回日期对应的星期几索引值，即星期日为 `1` ，星期一为 `2` ，星期六为 `7` 。
- 参数类型：

```
dayofweek(timestamp)
dayofweek(datetime)
dayofweek(date)
dayofweek(varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select dayofweek(timestamp '2022-01-22 00:00:00');
```

  返回结果如下。

```
+-----------------------------------------+
| dayofweek(TIMESTAMP '2022-01-22 00:00:00') |
+-----------------------------------------+
|                                       7 |
```

  示例2

```
select dayofweek(datetime '2022-01-22 00:00:00');
```

返回结果如下。

```
+-----------------------------------------+
| dayofweek(DATETIME '2022-01-22 00:00:00') |
+-----------------------------------------+
|                                       7 |
```

示例3

```
select dayofweek(date '2022-01-22');
```

返回结果如下。

```
+------------------------------+
| dayofweek(DATE '2022-01-22') |
+------------------------------+
|                            7 |
```

示例4

```
select dayofweek('2022-01-22');
```

返回结果如下。

```
+--------------------------+
| dayofweek('2022-01-22') |
+--------------------------+
|                        7 |
```

DAYOFYEAR

```
DAYOFYEAR(date)
```

- 命令说明：返回指定日期是当年的第几天，返回值范围为 `[1,366]` 。
- 参数类型：

```
dayofyear(timestamp)
dayofyear(datetime)
dayofyear(date)
dayofyear(varchar)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select dayofyear(timestamp '2022-02-01 00:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| dayofyear(TIMESTAMP '2022-02-01 00:12:12') |
+-----------------------------------------+
|                                      32 |
```

示例2

```
select dayofyear(datetime '2022-02-01 00:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| dayofyear(DATETIME '2022-02-01 00:12:12') |
+-----------------------------------------+
|                                      32 |
```

示例3

```
select dayofyear(date '2022-02-01');
```

返回结果如下。

```
+------------------------------+
| dayofyear(DATE '2022-02-01') |
+------------------------------+
|                           32 |
```

示例4

```
select dayofyear('2022-02-01');
```

返回结果如下。

```
+--------------------------+
| dayofyear('2022-02-01') |
+--------------------------+
|                       32 |
```

EXTRACT

```
EXTRACT(unit FROM date)
```

- 命令说明：返回日期或时间的单独部分，由 `unit` 指定，比如年、月、日、小时、分钟等。

  `unit` 可取值为：`second` 、`minute` 、`hour` 、`day` 、`month` 、`year` 、`minute_second` 、`hour_second` 、`hour_minute` 、`day_second` 、`day_minute` 、`day_hour` 、`year_month` 。
- 支持抽取的入参时间类型：VARCHAR、TIMESTAMP、DATETIME、TIME。
- 返回值类型：BIGINT。
- 示例：

  示例1

```
select extract(second from '2022-01-22 00:12:34');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|    34 |
```

示例2

```
select extract(minute from '2022-01-22 00:12:34');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|    12 |
```

示例3

```
select extract(hour from '2022-01-22 00:12:34');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|     0 |
```

示例4

```
select extract(day from '2022-01-22 00:12:34');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|    22 |
```

示例5

```
select extract(month from '2022-01-22 00:12:34');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|     1 |
```

示例6

```
select extract(year from timestamp '2022-01-22');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|  2022 |
```

示例7

```
select extract(year from datetime  '2022-01-22');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|  2022 |
```

示例8

```
select extract(year from time '15:23:22');
```

返回当前的年份，结果如下。

```
+-------+
| _col0 |
+-------+
|  2022 |
```

示例9

```
select extract(minute_second from '2022-01-22 00:12:34');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|  1234 |
```

示例10

```
select extract(hour_second from '2022-01-22 12:12:34');
```

返回结果如下。

```
+--------+
| _col0  |
+--------+
| 121234 |
```

示例11

```
select extract(hour_minute from '2022-01-22 12:12:34');
```

返回结果如下。

```
+-------+
| _col0 |
+-------+
|  1212 |
```

示例12

```
select extract(day_second from '2022-01-22 12:12:34');
```

返回结果如下。

```
+----------+
| _col0    |
+----------+
| 22121234 |
```

示例13

```
select extract(day_minute from '2022-01-22 00:12:34');
```

返回结果如下。

```
+--------+
| _col0  |
+--------+
| 220012 |
```

示例14

```
select extract(day_hour from '2022-01-22 12:12:34');
```

返回结果如下。

```
+--------+
| _col0 |
+--------+
|  2212 |
```

示例15

```
select extract(year_month from '2022-01-22 00:12:34');
```

返回结果如下。

```
+---------+
| _col0  |
+---------+
| 202201 |
```

FROM_DAYS

```
FROM_DAYS(N)
```

- 命令说明：根据指定的天数 `N` ，返回对应的 `DATE` 值。
- 参数类型：

```
from_days(varchar)
from_days(bigint)
```

- 返回值类型：DATE。
- 示例：

示例1

```
select from_days(738565);
```

返回结果如下。

```
+--------------------+
| from_days(738565) |
+--------------------+
|  2022-02-14       |
```

示例2

```
select from_days('738565');
```

返回结果如下。

```
+----------------------+
| from_days('738565') |
+----------------------+
|  2022-02-14         |
```

FROM_UNIXTIME

```
FROM_UNIXTIME(unix_timestamp[,format])
```

- 命令说明：根据 `unix` 时间戳，返回符合 `format` 格式的值。

  `format` 遵从DATE_FORMAT函数中的 `format` 格式。
- 参数类型：

```
from_unixtime(varchar, varchar)
from_unixtime(varchar)
from_unixtime(double, varchar)
from_unixtime(double)
```

- 返回值类型：DATETIME。
- 示例：

示例1

```
select from_unixtime('1647738565','%Y %M %h:%i:%s %x');
```

返回结果如下。

```
+-------------------------------------------------+
| from_unixtime('1647738565', '%Y %M %h:%i:%s %x') |
+-------------------------------------------------+
|    2022 March 09:09:25 2022                     |
```

示例2

```
select from_unixtime('1647738565');
```

返回结果如下。

```
+-----------------------------+
| from_unixtime('1647738565') |
+-----------------------------+
|         2022-03-20 09:09:25 |
```

示例3

```
select from_unixtime(1647738456);
```

返回结果如下。

```
+---------------------------+
| from_unixtime(1647738456) |
+---------------------------+
|       2022-03-20 09:07:36 |
```

示例4

```
select from_unixtime(1647738456,'%Y %M %h:%i:%s %x');
```

返回结果如下。

```
+-------------------------------------------------+
| from_unixtime(1647738456, '%Y %M %h:%i:%s %x') |
+-------------------------------------------------+
|     2022 March 09:07:36 2022                    |
```

HOUR

```
HOUR(time)
```

- 命令说明：返回时间中的小时。
- 参数类型：

```
hour(timestamp)
hour(datetime)
hour(date)
hour(time)
hour(varchar)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select hour(timestamp '2022-01-22 10:05:03');
```

返回结果如下。

```
+--------------------------------------+
| hour(TIMESTAMP '2022-01-22 10:05:03') |
+--------------------------------------+
|                                   10 |
```

示例2

```
select hour(datetime '2022-01-22 10:05:03');
```

返回结果如下。

```
+-------------------------------------+
| hour(DATETIME '2022-01-22 10:05:03') |
+-------------------------------------+
|                                  10 |
```

示例3

```
select hour(date '2022-01-22');
```

返回结果如下。

```
+-------------------------+
| hour(DATE '2022-01-22') |
+-------------------------+
|                       0 |
```

示例4

```
select hour(time '10:05:03');
```

返回结果如下。

```
+-----------------------+
| hour(TIME '10:05:03') |
+-----------------------+
|                    10 |
```

示例5

```
select hour('10:05:03');
```

返回结果如下。

```
+------------------+
| hour('10:05:03') |
+------------------+
|       10         |
```

LAST_DAY

```
LAST_DAY(date)
```

- 命令说明：返回日期或者日期时间中对应月份的最后一天。
- 参数类型：

```
last_day(varchar)
last_day(timestamp)
last_day(datetime)
last_day(date)
```

- 返回值类型：DATE。
- 示例：

示例1

```
select last_day('2022-01-22');
```

返回结果如下。

```
+------------------------+
| last_day('2022-01-22') |
+------------------------+
| 2022-01-31             |
```

示例2

```
select last_day(timestamp '2022-01-22 12:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| last_day(TIMESTAMP '2022-01-22 12:12:12') |
+-----------------------------------------+
| 2022-01-31                              |
```

示例3

```
select last_day(datetime '2022-01-22 12:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| last_day(DATETIME '2022-01-22 12:12:12') |
+-----------------------------------------+
| 2022-01-31                              |
```

示例4

```
select last_day(date '2022-01-22');
```

返回结果如下。

```
+------------------------------+
| last_day(DATE '2022-01-22') |
+------------------------------+
| 2022-01-31                   |
```

LOCALTIME/LOCALTIMESTAMP/NOW

```
localtime
localtime()
localtimestamp
localtimestamp()
now()
```

- 命令说明：返回当前时间戳。
- 返回值类型：DATETIME。
- 示例：

  示例1

  ```
  select now();
  ```

  返回结果如下。

  ```
  +---------------------+
  | now()               |
  +---------------------+
  | 2022-01-22 16:28:37 |
  ```

  示例2

  ```
  select localtime;
  ```

  返回结果如下。

  ```
  +---------------------+
  | localtime()         |
  +---------------------+
  | 2022-01-22 16:28:37 |
  ```

  示例3

  ```
  select localtime();
  ```

  返回结果如下。

  ```
  +---------------------+
  | localtime()         |
  +---------------------+
  | 2022-01-22 16:28:37 |
  ```

  示例4

  ```
  select localtimestamp;
  ```

  返回结果如下。

  ```
  +---------------------+
  | localtimestamp()    |
  +---------------------+
  | 2022-01-22 17:28:37 |
  ```

  示例5

  ```
  select localtimestamp();
  ```

  返回结果如下。

  ```
  +---------------------+
  | localtimestamp()    |
  +---------------------+
  | 2022-01-22 17:38:13 |
  ```

MAKEDATE

```
MAKEDATE(year,dayofyear)
```

- 命令说明：按照参数 `year` 和 `dayofyear` ，返回一个日期。
- 参数类型：

```
makedate(bigint, bigint)
makedate(varchar, varchar)
```

- 返回值类型：DATE。
- 示例：

  示例1

```
select makedate(2022,31), makedate(2022,32);
```

  返回结果如下。

```
+--------------------+--------------------+
| makedate(2022, 31) | makedate(2022, 32) |
+--------------------+--------------------+
| 2022-01-31         | 2022-02-01         |
```

  示例2

```
select makedate('2022','31'), makedate('2022','32');
```

  返回结果如下。

```
+------------------------+------------------------+
| makedate('2022', '31') | makedate('2022', '32') |
+------------------------+------------------------+
| 2022-01-31             | 2022-02-01             |
```

MAKETIME

```
MAKETIME(hour,minute,second)
```

- 命令说明：按照参数 `hour` 、 `minute` 和 `second` ，返回一个时间。
- 参数类型：

```
maketime(bigint, bigint, bigint)
maketime(varchar, varchar, varchar)
```

- 返回值类型：TIME。
- 示例：

  示例1

```
select maketime(12,15,30);
```

  返回结果如下。

```
+----------------------+
| maketime(12, 15, 30) |
+----------------------+
| 12:15:30             |
```

  示例2

```
select maketime('12','15','30');
```

  返回结果如下。

```
+----------------------------+
| maketime('12', '15', '30') |
+----------------------------+
| 12:15:30                   |
```

MINUTE

```
MINUTE(time)
```

- 命令说明：返回时间中的分钟。
- 参数类型：

```
minute(timestamp)
minute(datetime)
minute(date)
minute(time)
minute(varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select minute(timestamp '2022-02-03 10:05:03');
```

返回结果如下。

```
+-----------------------------------------+
| minute(TIMESTAMP '2022-02-03 10:05:03') |
+-----------------------------------------+
|                                       5 |
```

示例2

```
select minute(datetime '2022-02-03 10:05:03');
```

返回结果如下。

```
+--------------------------------------+
| minute(DATETIME '2022-02-03 10:05:03') |
+--------------------------------------+
|                                    5 |
```

示例3

```
select minute(date '2022-02-03');
```

返回结果如下。

```
+---------------------------+
| minute(DATE '2022-02-03') |
+---------------------------+
|                         0 |
```

示例4

```
select minute(time '12:12:12');
```

返回结果如下。

```
+-------------------------+
| minute(TIME '12:12:12') |
+-------------------------+
|                      12 |
```

示例5

```
select minute('2022-02-03 10:05:03');
```

返回结果如下。

```
+-------------------------------+
| minute('2022-02-03 10:05:03') |
+-------------------------------+
|                             5 |
```

MONTH

```
MONTH(date)
```

- 命令说明：返回日期中的月份。
- 参数类型：

```
month(timestamp)
month(datetime)
month(date)
month(time)
month(varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select month(timestamp '2022-02-03 00:00:00');
```

返回结果如下。

```
+---------------------------------------+
| month(TIMESTAMP '2022-02-03 00:00:00') |
+---------------------------------------+
|                                     2 |
```

示例2

```
select month(datetime '2022-02-03 00:00:00');
```

返回结果如下。

```
+---------------------------------------+
| month(DATETIME '2022-02-03 00:00:00') |
+---------------------------------------+
|                                     2 |
```

示例3

```
select month(date '2022-02-03');
```

返回结果如下。

```
+--------------------------+
| month(DATE '2022-02-03') |
+--------------------------+
|                        2 |
```

示例4

MONTH函数也可以返回SQL执行时的月份，例如以下SQL是2021年5月执行的， 返回结果为5。

```
select month(time '12:12:12');
```

返回结果如下。

```
+-------------------------+
| month(TIME '12:12:12') |
+-------------------------+
|                       5 |
```

示例5

```
select month('2022-02-03');
```

返回结果如下。

```
+---------------------+
| month('2022-02-03') |
+---------------------+
|                   2 |
```

MONTHNAME

```
MONTHNAME(date)
```

- 命令说明：返回日期中月份的英文全称。
- 参数类型：

```
monthname(timestamp)
monthname(datetime)
monthname(date)
monthname(varchar)
```

- 返回值类型：VARCHAR。
- 示例：

示例1

```
select monthname(timestamp '2022-02-03 00:00:00');
```

返回结果如下。

```
+-------------------------------------------+
| monthname(TIMESTAMP '2022-02-03 00:00:00') |
+-------------------------------------------+
| February                                  |
```

示例2

```
select monthname(datetime '2022-02-03 00:00:00');
```

返回结果如下。

```
+-----------------------------------------+
| monthname(DATETIME '2022-02-03 00:00:00') |
+-----------------------------------------+
| February                                |
```

示例3

```
select monthname(date '2022-02-03');
```

返回结果如下。

```
+------------------------------+
| monthname(DATE '2022-02-03') |
+------------------------------+
| February                     |
```

示例4

```
select monthname('2022-02-03');
```

返回结果如下。

```
+-------------------------+
| monthname('2022-02-03') |
+-------------------------+
| February                |
```

PERIOD_ADD

```
PERIOD_ADD(P,N)
```

- 命令说明：将日期格式的参数 `P` 增加 `N` 个月。
- 参数类型：

```
period_add(bigint, bigint)
period_add(varchar, varchar)
period_add(varchar, bigint)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select period_add(202201,2);
```

返回结果如下。

```
+-----------------------+
| period_add(202201, 2) |
+-----------------------+
|                202203 |
```

示例2

```
select period_add('202201','2');
```

返回结果如下。

```
+---------------------------+
| period_add('202201', '2') |
+---------------------------+
|                    202203 |
```

示例3

```
select period_add('202201',2);
```

返回结果如下。

```
+-------------------------+
| period_add('202201', 2) |
+-------------------------+
|                  202203 |
```

PERIOD_DIFF

```
PERIOD_DIFF(P1,P2)
```

- 命令说明：返回 `P1` 和 `P2` 之间相差的月数。
- 参数类型：

```
period_diff(bigint, bigint)
period_diff(varchar, varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select period_diff(202202,202103);
```

返回结果如下。

```
+----------------------------+
| period_diff(202202,202103) |
+----------------------------+
|                         11 |
```

示例2

```
select period_diff('202202','202103');
```

返回结果如下。

```
+---------------------------------+
| period_diff('202202', '202103') |
+---------------------------------+
|                              11 |
```

QUARTER

```
QUARTER(date)
```

- 命令说明：返回日期在一年中的季度，取值范围为 `[1,4]` 。
- 参数类型：

```
quarter(datetime)
quarter(varchar)
quarter(timestamp)
quarter(date)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select quarter(datetime '2022-04-01 12:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| quarter(DATETIME '2022-04-01 12:12:12') |
+-----------------------------------------+
|                                       2 |
```

示例2

```
select quarter('2022-04-01');
```

返回结果如下。

```
+-----------------------+
| quarter('2022-04-01') |
+-----------------------+
|                     2 |
```

示例3

```
select quarter(timestamp '2022-04-01 12:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| quarter(TIMESTAMP '2022-04-01 12:12:12') |
+-----------------------------------------+
|                                       2 |
```

示例4

```
select quarter(date '2022-04-01');
```

返回结果如下。

```
+----------------------------+
| quarter(DATE '2022-04-01') |
+----------------------------+
|                          2 |
```

SEC_TO_TIME

```
SEC_TO_TIME(seconds)
```

- 命令说明：将 `seconds` 转换为时间。
- 参数类型

```
sec_to_time(bigint)
sec_to_time(varchar)
```

- 返回值类型：TIME。
- 示例：

  示例1

```
select sec_to_time(2378);
```

  返回结果如下。

```
+-------------------+
| sec_to_time(2378) |
+-------------------+
| 00:39:38          |
```

  示例2

```
select sec_to_time('2378');
```

  返回结果如下。

```
+---------------------+
| sec_to_time('2378') |
+---------------------+
| 00:39:38            |
```

SECOND

```
SECOND(time)
```

- 命令说明：返回时间中的秒，范围为 `[0,59]` 。
- 参数类型：

```
second(timestamp)
second(datetime)
second(date)
second(time)
second(varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select second(timestamp '2022-03-12 12:13:14');
```

  返回结果如下。

```
+-----------------------------------------+
| second(TIMESTAMP '2022-03-12 12:13:14') |
+-----------------------------------------+
|                                      14 |
```

  示例2

```
select second(datetime '2022-03-12 12:13:14');
```

  返回结果如下。

```
+--------------------------------------+
| second(DATETIME '2022-03-12 12:13:14') |
+--------------------------------------+
|                                   14 |
```

  示例3

```
select second(date '2022-03-12');
```

  返回结果如下。

```
+---------------------------+
| second(DATE '2022-03-12') |
+---------------------------+
|                         0 |
```

示例3

```
select second(time '12:13:14');
```

返回结果如下。

```
+-------------------------+
| second(TIME '12:13:14') |
+-------------------------+
|                      14 |
```

示例4

```
select second('12:12:23');
```

返回结果如下。

```
+--------------------+
| second('12:12:23') |
+--------------------+
|                 23 |
```

STR_TO_DATE

```
STR_TO_DATE(str,format)
```

- 命令说明：按照指定日期或时间显示格式，将字符串转换为日期或日期时间类型。

  `format` 遵从DATE_FORMAT函数中的 `format` 格式。
- 参数类型：

  ```
  str_to_date(varchar, varchar)
  ```
- 返回值类型：DATETIME。
- 示例：

  示例1

  ```
  select str_to_date('2022-01-06 10:20:30','%Y-%m-%d %H:%i:%s') as result;
  ```

  返回结果如下。

  ```
  +---------------------+
  | result              |
  +---------------------+
  | 2022-01-06 10:20:30 |
  ```

SUBTIME

```
SUBTIME(expr1,expr2)
```

- 命令说明：返回 `expr1` 减去 `expr2` 后的时间。
- 参数类型：

  ```
  subtime(date, varchar)
  subtime(datetime, varchar)
  subtime(timestamp, varchar)
  subtime(time, varchar)
  subtime(varchar, varchar)
  ```
- 返回值类型：DATETIME。
- 示例：

  示例1

  ```
  select subtime(date '2022-10-31','0:1:1');
  ```

  返回结果如下。

  ```
  +-------------------------------------+
  | subtime(DATE '2022-10-31', '0:1:1') |
  +-------------------------------------+
  |                 2022-10-30 23:58:59 |
  ```

  示例2

  ```
  select subtime(datetime '2022-10-31 12:12:12','0:1:1');
  ```

  返回结果如下。

```
+-------------------------------------------------+
| subtime(DATETIME '2022-10-31 12:12:12', '0:1:1') |
+-------------------------------------------------+
|                            2022-10-31 12:11:11 |
```

示例3

```
select subtime(timestamp '2022-10-31 12:12:12','0:1:1');
```

返回结果如下。

```
+-------------------------------------------------+
| subtime(TIMESTAMP '2022-10-31 12:12:12', '0:1:1') |
+-------------------------------------------------+
|                             2022-10-31 12:11:11 |
```

示例4

```
select subtime(time '12:12:12','0:1:1');
```

返回结果如下。

```
+-----------------------------------+
| subtime(TIME '12:12:12', '0:1:1') |
+-----------------------------------+
| 12:11:11                          |
+-----------------------------------+
```

示例5

```
select subtime('2022-10-31 23:59:59','0:1:1');
```

返回结果如下。

```
+----------------------------------------+
| subtime('2022-10-31 23:59:59', '0:1:1') |
+----------------------------------------+
| 2022-10-31 23:58:58                    |
```

SYSDATE

```
SYSDATE()
```

- 命令说明：获取系统时间。
- 返回值类型：DATETIME。
- 示例：

  示例1

  ```
  select sysdate();
  ```

  返回结果如下。

  ```
  +---------------------+
  | sysdate()           |
  +---------------------+
  | 2022-02-26 00:47:21 |
  ```

TIME

```
TIME(expr)
```

- 命令说明：以字符串形式返回 `expr` 中的时间。
- 参数类型：

  ```
  time(varchar)
  time(datetime)
  time(timestamp)
  ```

- 返回值类型：VARCHAR。
- 示例：

  示例1

  ```
  select time('2022-01-31 01:02:03');
  ```

  返回结果如下。

```
+-----------------------------+
| time('2022-01-31 01:02:03') |
+-----------------------------+
| 01:02:03                    |
```

示例2

```
select time(datetime '2022-01-31 01:02:03');
```

返回结果如下。

```
+--------------------------------------+
| time(DATETIME '2022-01-31 01:02:03') |
+--------------------------------------+
| 01:02:03                             |
```

示例3

```
select time(timestamp '2022-01-31 01:02:03');
```

返回结果如下。

```
+---------------------------------------+
| time(TIMESTAMP '2022-01-31 01:02:03') |
+---------------------------------------+
| 01:02:03                              |
```

TIME_FORMAT

```
TIME_FORMAT(time,format)
```

- 命令说明：按照 `format` 指定的格式，以字符串格式显示时间 `time` 。

  `format` 遵从DATE_FORMAT函数中的 `format` 格式。
- 参数类型：

```
time_format(varchar, varchar)
time_format(timestamp, varchar)
time_format(datetime, varchar)
time_format(time, varchar)
time_format(date, varchar)
```

- 返回值类型：VARCHAR。
- 示例：

  示例1

```
select time_format('12:00:00', '%H %k %h %I %l');
```

  返回结果如下。

```
+-------------------------------------------+
| time_format('12:00:00', '%H %k %h %I %l') |
+-------------------------------------------+
| 12 12 12 12 12                            |
```

  示例2

```
select time_format(timestamp '2022-01-22 23:00:00','%H %k %h %I %l')as result;
```

  返回结果如下。

```
+----------------+
| result         |
+----------------+
| 23 23 11 11 11 |
```

  示例3

```
select time_format(datetime '2022-01-22 23:00:00','%H %k %h %I %l')as result;
```

  返回结果如下。

```
+----------------+
| result         |
+----------------+
| 23 23 11 11 11 |
```

  示例4

```
select time_format(time '23:00:00','%H %k %h %I %l');
```

返回结果如下。

```
+-----------------------------------------------+
| time_format(TIME '23:00:00', '%H %k %h %I %l') |
+-----------------------------------------------+
| 23 23 11 11 11                                |
```

示例5

```
select time_format(date '2022-01-22','%H %k %h %I %l');
```

返回结果如下。

```
+-------------------------------------------------+
| time_format(DATE '2022-01-22', '%H %k %h %I %l') |
+-------------------------------------------------+
| 00 0 12 12 12                                   |
```

TIME_TO_SEC

```
TIME_TO_SEC(time)
```

- 命令说明：返回 `time` 转换为秒的结果。
- 参数类型：

```
time_to_sec(varchar)
time_to_sec(datetime)
time_to_sec(timestamp)
time_to_sec(date)
time_to_sec(time)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select time_to_sec(datetime '2022-01-22 22:23:00');
```

返回结果如下。

```
+-------------------------------------------+
| time_to_sec(DATETIME '2022-01-22 22:23:00') |
+-------------------------------------------+
|                                     80580 |
```

示例2

```
select time_to_sec(timestamp '2022-01-22 22:23:00');
```

返回结果如下。

```
+-------------------------------------------+
| time_to_sec(TIMESTAMP '2022-01-22 22:23:00') |
+-------------------------------------------+
|                                     80580 |
```

示例3

```
select time_to_sec(date '2022-01-22');
```

返回结果如下。

```
+-------------------------------+
| time_to_sec(DATE '2022-01-22') |
+-------------------------------+
|                             0 |
```

示例4

```
select time_to_sec(time '12:12:12');
```

返回结果如下。

```
+-----------------------------+
| time_to_sec(TIME '12:12:12') |
+-----------------------------+
|                       43932 |
```

示例5

```
select time_to_sec('22:23:00');
```

返回结果如下。

```
+-------------------------+
| time_to_sec('22:23:00') |
+-------------------------+
|                   80580 |
```

TIMEDIFF

```
TIMEDIFF(expr1,expr2)
```

- 命令说明：返回 `expr1` 减去 `expr2` 后的时间，与SUBTIME作用相同。
- 参数类型：

```
timediff(time, varchar)
timediff(time, time)
timediff(varchar, varchar)
```

- 返回值类型：DATETIME。
- 示例：

示例1

```
select timediff(time '12:00:00','10:00:00');
```

返回结果如下。

```
+---------------------------------------+
| timediff(TIME '12:00:00', '10:00:00') |
+---------------------------------------+
| 02:00:00                              |
```

示例2

```
select timediff('12:00:00','10:00:00');
```

返回结果如下。

```
+-----------------------------------+
| timediff('12:00:00', '10:00:00') |
+-----------------------------------+
| 02:00:00                          |
```

示例3

```
select timediff(time '12:00:00',time '10:00:00');
```

返回结果如下。

```
+-------------------------------------------+
| timediff(TIME '12:00:00', TIME '10:00:00') |
+-------------------------------------------+
| 02:00:00                                  |
```

TIMESTAMP

```
TIMESTAMP(expr)
```

- 命令说明：返回 `expr` 表示的日期或日期时间。
- 参数类型：

```
timestamp(date)
timestamp(varchar)
```

- 返回值类型：DATETIME。
- 示例：

示例1

```
select timestamp(date '2022-01-22');
```

返回结果如下。

```
+------------------------------+
| timestamp(DATE '2022-01-22') |
+------------------------------+
| 2022-01-22 00:00:00          |
```

示例2

```
select timestamp('2022-01-22');
```

返回结果如下。

```
+-------------------------+
| timestamp('2022-01-22') |
+-------------------------+
|     2022-01-22 00:00:00 |
```

TIMESTAMPADD

```
TIMESTAMPADD(unit,interval,datetime_expr)
```

- 命令说明：将 `interval` 添加到日期或日期时间表达式 `datetime_expr` 中。 `interval` 的单位由 `unit` 规定。

  `unit` 可取值为： `second` 、 `minute` 、 `hour` 、 `day` 、 `week` 、 `month` 、 `quarter` 、 `year` 。
- 参数类型：

```
timestampadd(varchar, varchar, timestamp)
timestampadd(varchar, bigint, timestamp)
timestampadd(varchar, varchar, date)
timestampadd(varchar, bigint, date)
timestampadd(varchar, varchar, datetime)
timestampadd(varchar, bigint, datetime)
timestampadd(varchar, varchar, varchar)
timestampadd(varchar, bigint, varchar)
```

- 返回值类型：DATETIME。
- 示例：

  示例1

```
select timestampadd(second,'1',timestamp '2022-01-02 12:12:12')as result;
```

  返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 12:12:13 |
```

  示例2

```
select timestampadd(second,1,timestamp '2022-01-02 12:12:12')as result;
```

  返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 12:12:13 |
```

  示例3

```
select timestampadd(second,'1',date '2022-01-02 12:12:12')as result;
```

  返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 00:00:01 |
```

  示例4

```
select timestampadd(second,1,date '2022-01-02 12:12:12')as result;
```

  返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 00:00:01 |
```

  示例5

```
select timestampadd(second,'1',datetime '2022-01-02 12:12:12')as result;
```

  返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 12:12:13 |
```

示例6

```
select timestampadd(second,1,datetime '2022-01-02 12:12:12')as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 12:12:13 |
```

示例7

```
select timestampadd(second,'1','2022-01-02 12:12:12')as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 12:12:13 |
```

示例8

```
select timestampadd(second,1,'2022-01-02 12:12:12')as result;
```

返回结果如下。

```
+---------------------+
| result              |
+---------------------+
| 2022-01-02 12:12:13 |
```

示例9

```
select timestampadd(second,1,'2022-01-02 12:12:12');
```

返回结果如下。

```
+-----------------------------------------------+
| timestampadd('second', 1, '2022-01-02 12:12:12') |
+-----------------------------------------------+
| 2022-01-02 12:12:13                           |
```

示例10

```
select timestampadd(minute,8820,'2022-02-24 09:00:00');
```

返回结果如下。

```
+-------------------------------------------------+
| timestampadd('MINUTE', 8820, '2022-02-24 09:00:00') |
+-------------------------------------------------+
| 2022-03-02 12:00:00                             |
```

示例11

```
select timestampadd(hour,1,'2022-01-02 12:12:12');
```

返回结果如下。

```
+-----------------------------------------------+
| timestampadd('hour', 1, '2022-01-02 12:12:12') |
+-----------------------------------------------+
| 2022-01-02 13:12:12                           |
```

示例12

```
select timestampadd(day,1,'2022-01-02 12:12:12');
```

返回结果如下。

```
+-----------------------------------------------+
| timestampadd('day', 1, '2022-01-02 12:12:12') |
+-----------------------------------------------+
| 2022-01-03 12:12:12                           |
```

示例13

```
select timestampadd(week,1,'2022-01-02 12:12:12');
```

返回结果如下。

```
+-----------------------------------------------+
| timestampadd('week', 1, '2022-01-02 12:12:12') |
+-----------------------------------------------+
| 2022-01-09 12:12:12                           |
```

示例14

```
select timestampadd(month,1,'2022-01-02 12:12:12');
```

返回结果如下。

```
+-----------------------------------------------+
| timestampadd('month', 1, '2022-01-02 12:12:12') |
+-----------------------------------------------+
| 2022-02-02 12:12:12                           |
```

示例15

```
select timestampadd(year,1,'2022-01-02 12:12:12');
```

返回结果如下。

```
+-----------------------------------------------+
| timestampadd('year', 1, '2022-01-02 12:12:12') |
+-----------------------------------------------+
| 2023-01-02 12:12:12                           |
```

示例16

```
select timestampadd(quarter,1,'2022-01-02 12:12:12');
```

返回结果如下。

```
+-------------------------------------------------+
| timestampadd('quarter', 1, '2022-01-02 12:12:12') |
+-------------------------------------------------+
| 2022-04-02 12:12:12                             |
```

TIMESTAMPDIFF

```
TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)
```

- 命令说明：返回日期或日期时间表达式 `datetime_expr2` 减去 `datetime_expr1` 后的结果，结果的单位由 `unit` 指定。

  `unit` 可取值为： `second` 、 `minute` 、 `hour` 、 `day` 、 `week` 、 `month` 、 `quarter` 或 `year` 。

  使用方法和TIMESTAMPADD相同。
- 参数类型：

  ```
  timestampdiff(varchar, timestamp, timestamp)
  timestampdiff(varchar, date, date)
  timestampdiff(varchar, datetime, datetime)
  timestampdiff(varchar, varchar, varchar)
  ```
- 返回值类型：BIGINT。
- 示例：

  示例1

  ```
  select timestampdiff(second,datetime '2022-02-01 10:12:13',datetime '2022-03-01 10:12:13')as result;
  ```

  返回结果如下。

  ```
  +----------+
  |  result  |
  +----------+
  |  2419200 |
  ```

  示例2

  ```
  select timestampdiff(minute,datetime '2022-02-01 10:12:13',datetime '2022-03-01 10:12:13')as result;
  ```

  返回结果如下。

  ```
  +---------+
  |  result |
  +---------+
  |  40320  |
  ```

  示例3

  ```
  select timestampdiff(hour,datetime '2022-02-01 10:12:13',datetime '2022-03-01 10:12:13')as result;
  ```

返回结果如下。

```
+--------+
| result |
+--------+
|   672  |
```

示例4

```
select timestampdiff(day,timestamp '2022-02-01',timestamp '2022-03-01')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|    28  |
```

示例5

```
select timestampdiff(day,date '2022-02-01',date '2022-03-01');
```

返回结果如下。

```
+-----------------------------------------------------------+
| timestampdiff('day', DATE '2022-02-01', DATE '2022-03-01') |
+-----------------------------------------------------------+
|                                                         28 |
```

示例6

```
select timestampdiff(day,datetime '2022-02-01 10:12:13',datetime '2022-03-01 10:12:13')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|    28  |
```

示例7

```
select timestampdiff(day,'2022-02-01','2022-03-01');
```

返回结果如下。

```
+-------------------------------------------------+
| timestampdiff('day', '2022-02-01', '2022-03-01') |
+-------------------------------------------------+
|                                              28  |
```

示例8

```
select timestampdiff(week,'2022-02-01','2022-03-01');
```

返回结果如下。

```
+-------------------------------------------------+
| timestampdiff('week', '2022-02-01', '2022-03-01') |
+-------------------------------------------------+
|                                               4  |
```

示例9

```
select timestampdiff(quarter,'2022-02-01','2022-03-01');
```

返回结果如下。

```
+---------------------------------------------------+
| timestampdiff('quarter', '2022-02-01', '2022-03-01') |
+---------------------------------------------------+
|                                                 0  |
```

示例10

```
select timestampdiff(month,'2022-02-01','2022-03-01');
```

返回结果如下。

```
+-------------------------------------------------+
| timestampdiff('month', '2022-02-01', '2022-03-01') |
+-------------------------------------------------+
|                                               1  |
```

示例11

```
select timestampdiff(year,datetime '2022-02-01 10:12:13',datetime '2020-05-01 10:12:13')as result;
```

返回结果如下。

```
+--------+
| result |
+--------+
|     -1 |
```

TO_DAYS

```
TO_DAYS(date)
```

- 命令说明：根据指定日期 `date` ，返回自 `0` 年开始的天数。
- 参数类型：

```
to_days(date)
to_days(time)
to_days(varchar)
to_days(timestamp)
to_days(datetime)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select to_days(date '2022-02-12');
```

返回结果如下。

```
+----------------------------+
| to_days(DATE '2022-02-12') |
+----------------------------+
|                     738563 |
```

示例2

```
select to_days(time '12:12:12');
```

返回当前日期距0年的天数，结果如下。

```
+--------------------------+
| to_days(TIME '12:12:12') |
+--------------------------+
|                   738563 |
```

示例3

```
select to_days(now());
```

返回结果如下。

```
+----------------+
| to_days(now()) |
+----------------+
|         738563 |
```

示例4

> **说明** 该示例等价于示例3。

```
select to_days(curdate());
```

返回结果如下。

```
+--------------------+
| to_days(curdate()) |
+--------------------+
|             738563 |
```

示例5

```
select to_days(datetime '2022-02-12 12:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| to_days(DATETIME '2022-02-12 12:12:12') |
+-----------------------------------------+
|                                  738563 |
```

示例6

```
select to_days('2022-02-12 12:12:12');
```

返回结果如下。

```
+---------------------------------+
| to_days('2022-02-12 12:12:12') |
+---------------------------------+
|                          738563 |
```

示例7

```
select to_days(timestamp '2022-02-12 12:12:12');
```

返回结果如下。

```
+-----------------------------------------+
| to_days(TIMESTAMP '2022-02-12 12:12:12') |
+-----------------------------------------+
|                                  738563 |
```

TO_SECONDS

```
TO_SECONDS(expr)
```

- 命令说明：根据给定的 `expr` ，返回自 `0` 年开始的秒数。
- 参数类型：

```
to_seconds(date)
to_seconds(datetime)
to_seconds(timestamp)
to_seconds(varchar)
to_seconds(time)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select to_seconds(date '2022-02-02');
```

返回结果如下。

```
+---------------------------------+
| to_seconds(DATE '2022-02-02') |
+---------------------------------+
|                   63810979200 |
```

示例2

```
select to_seconds(datetime '2022-02-02 09:09:00');
```

返回结果如下。

```
+-----------------------------------------+
| to_seconds(DATETIME '2022-02-02 09:09:00') |
+-----------------------------------------+
|                              63811012140 |
```

示例3

```
select to_seconds(timestamp '2022-02-02 09:09:00');
```

返回结果如下。

```
+-----------------------------------------+
| to_seconds(TIMESTAMP '2022-02-02 09:09:00') |
+-----------------------------------------+
|                               63811012140 |
```

示例4

执行以下SQL，系统将返回 `'09:09:00'` 加上 `curdate()` 后的结果。

```
select to_seconds(time '09:09:00');
```

返回结果如下。

```
+-----------------------------+
| to_seconds(TIME '09:09:00') |
+-----------------------------+
|                 63811012140 |
```

示例5

```
select to_seconds('2022-02-02');
```

返回结果如下。

```
+--------------------------+
| to_seconds('2022-02-02') |
+--------------------------+
|              63810979200 |
```

UNIX_TIMESTAMP

```
UNIX_TIMESTAMP([date])
```

- 命令说明：`UNIX_TIMESTAMP()` 返回自 `'1970-01-01 00:00:00' UTC` 至当前时间的秒数。`UNIX_TIMESTAMP(date)` 将参数的值返回为 `'1970-01-01 00:00:00'UTC` 至date指定时间的秒数。
- 参数类型：

```
unix_timestamp()
unix_timestamp(varchar)
unix_timestamp(timestamp)
unix_timestamp(date)
unix_timestamp(datetime)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select unix_timestamp();
```

  返回结果如下。

```
+------------------+
| unix_timestamp() |
+------------------+
|       1654759686 |
```

  示例2

```
select unix_timestamp(timestamp '2022-02-08 12:12:12');
```

  返回结果如下。

```
+-------------------------------------------------+
| unix_timestamp(TIMESTAMP '2022-02-08 12:12:12') |
+-------------------------------------------------+
|                                      1644293532 |
```

  示例3

```
select unix_timestamp(date '2022-02-08');
```

  返回结果如下。

```
+----------------------------------+
| unix_timestamp(DATE '2022-02-08') |
+----------------------------------+
|                       1644249600 |
```

  示例4

```
select unix_timestamp(datetime '2022-02-08 12:12:12');
```

  返回结果如下。

```
+-------------------------------------------------+
| unix_timestamp(DATETIME '2022-02-08 12:12:12') |
+-------------------------------------------------+
|                                      1644293532 |
```

  示例5

```
select unix_timestamp('2022-02-08 12:12:12');
```

返回结果如下。

```
+---------------------------------------+
| unix_timestamp('2022-02-08 12:12:12') |
+---------------------------------------+
|                            1644293532 |
```

UTC_DATE

```
UTC_DATE()
```

- 命令说明：返回UTC日期。
- 返回值类型：VARCHAR。
- 示例：

```
select utc_date();
```

返回结果如下。

```
+------------+
| utc_date() |
+------------+
| 2022-05-27 |
```

UTC_TIME

```
UTC_TIME()
```

- 命令说明：返回UTC时间。
- 返回值类型：VARCHAR。
- 示例：

```
select utc_time();
```

返回结果如下。

```
+------------+
| utc_time() |
+------------+
| 05:53:19   |
```

UTC_TIMESTAMP

```
utc_timestamp()
```

- 命令说明：返回UTC时间戳。
- 返回值类型：VARCHAR。
- 示例：

```
select utc_timestamp();
```

返回结果如下。

```
+---------------------+
| utc_timestamp()     |
+---------------------+
| 2022-05-27 15:55:15 |
```

WEEK

```
WEEK(date[,mode])
```

- 命令说明：返回`date`对应的周数，即`date`是日期年份中的第几周。
  - `date`是要获取周数的日期。
  - `mode`可选参数，用于确定周数计算的逻辑。默认一周的第一天是星期日。您也可以指定一周是从星期一还是星期日开始。`mode`支持的格式如下表所示。

0	星期日	0-53
1	星期一	0-53
2	星期日	1-53
3	星期一	1-53

4	星期日	0-53
5	星期一	0-53
6	星期日	1-53
7	星期一	1-53

- 参数类型：

```
week(varchar)
week(varchar, bigint)
week(date)
week(date, bigint)
week(datetime)
week(datetime, bigint)
week(timestamp)
week(timestamp, bigint)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select week('2022-02-27');
```

  返回结果如下。

```
+--------------------+
| week('2022-02-27') |
+--------------------+
|                  9 |
```

  示例2

```
select week('2022-02-20',1);
```

  返回结果如下。

```
+-----------------------+
| week('2022-02-20', 1) |
+-----------------------+
|                     7 |
```

  示例3

```
select week(date '2022-02-20');
```

  返回结果如下。

```
+--------------------------+
| week(DATE '2022-02-20') |
+--------------------------+
|                        8 |
```

  示例4

```
select week(date '2022-02-20',1);
```

  返回结果如下。

```
+-----------------------------+
| week(DATE '2022-02-20', 1) |
+-----------------------------+
|                           7 |
```

  示例5

```
select week(datetime '2022-02-20 00:00:00',1);
```

  返回结果如下。

```
+---------------------------------------+
| week(DATETIME '2022-02-20 00:00:00', 1) |
+---------------------------------------+
|                                     7 |
```

  示例6

```
select week(datetime '2022-02-20 00:00:00');
```

  返回结果如下。

```
+--------------------------------------+
| week(DATETIME '2022-02-20 00:00:00') |
+--------------------------------------+
|                                    8 |
```

示例7

```
select week(timestamp '2022-02-20 00:00:00');
```

返回结果如下。

```
+---------------------------------------+
| week(TIMESTAMP '2022-02-20 00:00:00') |
+---------------------------------------+
|                                     8 |
```

示例8

```
select week(timestamp '2022-02-20 00:00:00',1);
```

返回结果如下。

```
+----------------------------------------+
| week(TIMESTAMP '2022-02-20 00:00:00', 1) |
+----------------------------------------+
|                                      7 |
```

WEEKDAY

```
WEEKDAY(date)
```

- 命令说明：返回 `date` 是一周中的第几天。返回值0= Monday,1= Tuesday,2=Wednesday, 3=Thursday,4=Friday,5=Saturday,6= Sunday。
- 参数类型：

```
weekday(timestamp)
weekday(datetime)
weekday(date)
weekday(varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select weekday(timestamp '2022-02-20 00:09:00');
```

  返回结果如下。

```
+----------------------------------------+
| weekday(TIMESTAMP '2022-02-20 00:09:00') |
+----------------------------------------+
|                                      6 |
```

  示例2

```
select weekday(datetime '2022-02-20 00:09:00');
```

  返回结果如下。

```
+---------------------------------------+
| weekday(DATETIME '2022-02-20 00:09:00') |
+---------------------------------------+
|                                     6 |
```

  示例3

```
select weekday(date '2022-02-20 00:09:00');
```

  返回结果如下。

```
+-------------------------------------+
| weekday(DATE '2022-02-20 00:09:00') |
+-------------------------------------+
|                                   6 |
```

  示例4

```
select weekday('2022-02-20');
```

  返回结果如下。

```
+------------------------+
| weekday('2022-02-20') |
+------------------------+
|                      6 |
```

WEEKOFYEAR

```
WEEKOFYEAR(date)
```

- 命令说明：返回 `date` 对应的日历周，取值范围为 `[1,53]` 。
- 参数类型：

```
weekofyear(timestamp)
weekofyear(datetime)
weekofyear(date)
weekofyear(varchar)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select weekofyear(timestamp '2022-02-27 09:00:00');
```

  返回结果如下。

```
+---------------------------------------------+
| weekofyear(TIMESTAMP '2022-02-27 09:00:00') |
+---------------------------------------------+
|                                           8 |
```

  示例2

```
select weekofyear(datetime '2022-02-27 09:00:00');
```

  返回结果如下。

```
+-----------------------------------------+
| weekofyear(DATETIME '2022-02-27 09:00:00') |
+-----------------------------------------+
|                                       8 |
```

  示例3

```
select weekofyear(date '2022-02-27');
```

  返回结果如下。

```
+-------------------------------+
| weekofyear(DATE '2022-02-27') |
+-------------------------------+
|                             8 |
```

  示例4

```
select weekofyear('2022-02-27');
```

  返回结果如下。

```
+--------------------------+
| weekofyear('2022-02-27') |
+--------------------------+
|                        8 |
```

YEAR

```
YEAR(date)
```

- 命令说明：返回 `date` 中的年份。
- 参数类型：

```
year(timestamp)
year(datetime)
year(date)
year(time)
year(varchar)
```

- 返回值类型：BIGINT。
- 示例：

示例1

```
select year(timestamp '2022-02-27 00:00:00');
```

返回结果如下。

```
select year(timestamp '2019-05-27 00:00:00');
+---------------------------------------+
| year(TIMESTAMP '2022-02-27 00:00:00') |
+---------------------------------------+
|                                  2022 |
```

示例2

```
select year(datetime '2022-02-27 00:00:00')
```

返回结果如下。

```
+--------------------------------------+
| year(DATETIME '2022-02-27 00:00:00') |
+--------------------------------------+
|                                 2022 |
```

示例3

```
select year(date '2022-02-27');
```

返回结果如下。

```
+-------------------------+
| year(DATE '2022-02-27') |
+-------------------------+
|                    2022 |
```

示例4

执行以下SQL，系统将返回 `'00:00:00'` 加上 `curdate` 日期部分后的结果，结果数据类型为字符串。

```
select year(time '00:00:00');
```

返回结果如下。

```
+-----------------------+
| year(TIME '00:00:00') |
+-----------------------+
|                  2022 |
```

示例5

```
select year('2022-02-27');
```

返回结果如下。

```
+--------------------+
| year('2022-02-27') |
+--------------------+
|               2022 |
```

YEARWEEK

```
YEARWEEK(date)
YEARWEEK(date,mode)
```

- 命令说明：返回日期的年份和星期。

  返回结果中的年份可能与一年中第一周和最后一周的日期参数中的年份不同。

  `mode` 与WEEK函数中的 `mode` 作用相同。对于单参数语法， `mode` 值为 `0` 。
- 参数类型：

```
yearweek(timestamp)
yearweek(timestamp, bigint)
yearweek(datetime)
yearweek(datetime, bigint)
yearweek(date, bigint)
yearweek(date)
yearweek(varchar)
yearweek(varchar, bigint)
```

- 返回值类型：BIGINT。
- 示例：

  示例1

```
select yearweek(timestamp '2022-02-27 00:00:00');
```

返回结果如下。

```
+-----------------------------------------+
| yearweek(TIMESTAMP '2022-02-27 00:00:00') |
+-----------------------------------------+
|                                  202209 |
```

示例2

```
select yearweek(timestamp '2022-02-27 00:00:00',1);
```

返回结果如下。

```
+-----------------------------------------------+
| yearweek(TIMESTAMP '2022-02-27 00:00:00', 1) |
+-----------------------------------------------+
|                                       202208 |
```

示例3

```
select yearweek(datetime '2022-02-27 00:00:00');
```

返回结果如下。

```
+-----------------------------------------+
| yearweek(DATETIME '2022-02-27 00:00:00') |
+-----------------------------------------+
|                                  202209 |
```

示例4

```
select yearweek(datetime '2022-02-27 00:00:00',1);
```

返回结果如下。

```
+-----------------------------------------------+
| yearweek(DATETIME '2022-02-27 00:00:00', 1) |
+-----------------------------------------------+
|                                       202208 |
```

示例5

```
select yearweek(date '2022-02-27',1);
```

返回结果如下。

```
+--------------------------------+
| yearweek(DATE '2022-02-27', 1) |
+--------------------------------+
|                         202208|
```

示例6

```
select yearweek(date '2022-02-27');
```

返回结果如下。

```
+----------------------------+
| yearweek(DATE '2022-02-27') |
+----------------------------+
|                     202209 |
```

示例7

```
select yearweek('2022-02-27');
```

返回结果如下。

```
+------------------------+
| yearweek('2022-02-27') |
+------------------------+
|                 202209 |
```

示例8

```
select yearweek('2022-02-27',1);
```

返回结果如下。

```
+---------------------------+
| yearweek('2022-02-27', 1) |
+---------------------------+
|                    202208 |
```

4.4. 字符串函数

本文介绍AnalyticDB MySQL版支持的字符串函数。

- ASCII：返回字符或者字符串最左边字符对应的ASCII值。
- BIN：返回整数的二进制字符串。
- BIT_LENGTH：以位为单位返回字符串的长度。
- CHAR：返回整数对应的ASCII码组成的字符串。
- CHAR_LENGTH或CHARACTER_LENGTH：以字符为单位返回字符串的长度。
- CONCAT：连接字符串。
- CONCAT_WS：连接字符串，字符串中间以分隔串间隔。
- ELT：返回整数N指定的字符串。
- ENCRYPT：对字符串进行加密。
- EXPORT_SET：根据整数中的比特值，返回组合后的字符串。
- FIELD：返回指定字符串在字符串列表中的索引位置。
- FIND_IN_SET：返回字符或字符串在另一个字符串中的位置。
- FORMAT：将数字N格式化，返回字符串。
- FROM_BASE64：解码Base64编码的字符串并返回结果。
- FROM_UTF8：解码UTF-8编码的字符串并返回结果。
- HEX：将一个整数或字符串转换为其所对应的十六进制格式的字符串。
- INSTR：返回字符串中子字符串首次出现的位置。
- LEFT：从字符串最左边开始，返回N个字符。
- LENGTH或OCTET_LENGTH：字符串长度。
- LIKE：简单的模式匹配。
- LOCATE：返回字符串首次出现在另一个字符串中的位置信息。
- LOWER或LCASE：将字符串转换为小写。
- LPAD：左拼接字符串。
- LTRIM：删除字符串的前导空格。
- MAKE_SET：返回一组以逗号分隔的字符串。
- MID：从字符串的指定位置开始，返回指定长度的子字符串。作用同SUBSTR或SUBSTRING。
- OCT：返回指定整数的八进制字符串表示形式。
- ORD：如果字符串最左边的字符是多字节字符，则返回该字符的代码。
- POSITION：返回字符串中子字符串首次出现的位置。
- REPEAT：返回字符串重复多次后的字符串。
- REPLACE：用指定字符串替换另一个字符串中的部分字符。
- REVERSE：将字符串逆序。
- RIGHT：返回字符串最右边的指定数量的字符。
- RLIKE或REGEXP：将字符串expression与pattern进行正则匹配，匹配成功返回1，否则返回0。
- RPAD：右拼接字符串。
- RTRIM：删除字符串的后置空格。
- SPACE：返回由指定数量空格组成的字符串。
- SPLIT：将字符串按分隔符进行分隔，并返回数组。
- SPLIT_PART：将字符串按分隔符分隔，并返回分隔后指定数组下标的子串。
- SPLIT_TO_MAP：通过 `entryDelimiter` 和 `keyValueDelimiter` 拆分字符串，并返回 `map` 。
- STRCMP：根据两个字符串的大小，返回0、1或者-1。
- SUBSTR或SUBSTRING：返回从指定位置开始的指定长度的子字符串。
- SUBSTRING_INDEX：返回字符串中最后一次分隔符出现之前的子字符串。
- TO_BASE64：返回字符串的Base64编码形式。
- TO_UTF8：返回字符串的UTF-8编码形式。
- TRIM：删除字符串前后所有的空格。
- UPPER或UCASE：将字符串转换为大写。
- UNHEX：将十六进制数字转换为字符。

ASCII

```
ascii(str)
```

- 命令说明：返回字符 `str` 或者字符串 `str` 最左边字符对应的十进制ASCII值。

- 输入值类型：VARCHAR。
- 返回值类型：BIGINT。
- 示例：
  - 语句如下：

    ```
    SELECT ascii('2');
    ```

    返回结果如下：

    ```
    +------------+
    | ascii('2') |
    +------------+
    |         50 |
    +------------+
    ```

  - 语句如下：

    ```
    SELECT ascii('dx');
    ```

    返回结果如下：

    ```
    +-------------+
    | ascii('dx') |
    +-------------+
    |         100 |
    +-------------+
    ```

BIN

```
bin(N)
```

- 命令说明：返回 `N` 的二进制字符串。

  如果 `N` 为 `null`，则返回结果为 `NULL`。
- 输入值类型：BIGINT。
- 返回值类型：VARCHAR。
- 示例：

  ```
  SELECT bin(12);
  ```

  返回结果如下：

  ```
  +---------+
  | bin(12) |
  +---------+
  | 1100    |
  +---------+
  ```

BIT_LENGTH

```
bit_length(str)
```

- 命令说明：以位为单位返回字符串 `str` 的长度。
- 输入值类型：VARCHAR。
- 返回值类型：BIGINT。
- 示例：
  - 语句如下：

    ```
    SELECT bit_length('text');
    ```

    返回结果如下：

    ```
    +--------------------+
    | bit_length('text') |
    +--------------------+
    |                 32 |
    +--------------------+
    ```

- 语句如下：

```
SELECT bit_length('China');
```

返回结果如下：

```
+---------------------+
| bit_length('China') |
+---------------------+
|                  40 |
+---------------------+
```

CHAR

```
char(N1, N2, ...)
```

- 命令说明： 返回 `N1` 、 `N2` 等整数对应的十进制ASCII码组成的字符串。
- 输入值类型：BIGINT。
- 返回值类型：VARBINARY。
- 示例：

```
SELECT char(97,110,97,108,121,116,105,99,100,98);
```

返回结果如下：

```
+-----------------------------------------+
| char(97,110,97,108,121,116,105,99,100,98) |
+-----------------------------------------+
| analyticdb                              |
+-----------------------------------------+
```

CHAR_LENGTH或CHARACTER_LENGTH

```
char_length(str)
character_length(str)
```

- 命令说明： 以字符为单位返回字符串 `str` 的长度。

  一个汉字所对应的字符长度是 `1` 。
- 输入值类型：VARCHAR。
- 返回值类型：BIGINT。
- 示例：
  - 语句如下：

```
SELECT char_length('China');
```

返回结果如下：

```
+----------------------+
| char_length('China') |
+----------------------+
|                    5 |
+----------------------+
```

  - 语句如下：

```
SELECT char_length('abc');
```

返回结果如下：

```
+--------------------+
| char_length('abc') |
+--------------------+
|                  3 |
+--------------------+
```

CONCAT

```
concat(str 1, …, str n)
```

- 命令说明：字符串连接操作，其中任何一个参数为 `null` ，则返回值为 `null` 。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：

- 语句如下：

```
SELECT concat('aliyun', ', ', 'analyticdb');
```

返回结果如下：

```
+--------------------------------------+
| concat('aliyun', ', ', 'analyticdb') |
+--------------------------------------+
| aliyun, analyticdb                   |
+--------------------------------------+
```

- 语句如下：

```
SELECT concat('abc',null,'def');
```

返回结果如下：

```
+--------------------------+
| concat('abc',null,'def') |
+--------------------------+
| NULL                     |
+--------------------------+
```

CONCAT_WS

```
concat_ws(separator, str 1, …, str n)
```

- 命令说明：字符串连接操作，第一个参数 `separator` 是其余参数的分隔符，连接时会跳过任何为 `null` 值的字符串。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：
  - 语句如下：

```
SELECT concat_ws(',', 'First name', 'Second name', 'Last Name')AS result;
```

返回结果如下：

```
+---------------------------------+
| result                          |
+---------------------------------+
| First name,Second name,Last Name |
+---------------------------------+
```

  - 语句如下：

```
SELECT concat_ws(',','First name',NULL,'Last Name')AS result;
```

返回结果如下：

```
+----------------------+
| result               |
+----------------------+
| First name,Last Name |
+----------------------+
```

ELT

```
elt(N, str 1, ...,str n);
```

- 命令说明：返回第 `N` 个字符串。

  若 `N<1` 或大于后面字符串参数的数量，则返回结果为 `null` 。
- 输入值类型： `N` 为BIGINT类型， `str` 为VARCHAR类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT elt(4, 'Aa', 'Bb', 'Cc', 'Dd');
```

返回结果如下：

```
+-------------------------------+
| elt(4, 'Aa', 'Bb', 'Cc', 'Dd') |
+-------------------------------+
| Dd                            |
+-------------------------------+
```

ENCRYPT

```
encrypt(x, y);
```

- 命令说明：对参数 `x` 进行加密， `y` 为Salt值。
- 输入值类型： `x` 为VARBINARY类型， `y` 为VARCHAR类型。
- 返回值类型：VARBINARY
- 示例：

```
SELECT encrypt('abdABC123','key');
```

返回结果如下：

```
+------------------------------------------------------------+
| encrypt('abdABC123','key')                                 |
+------------------------------------------------------------+
| 0x6B657A617A6D63496F2E614377                               |
+------------------------------------------------------------+
```

EXPORT_SET

```
export_set(bits, onstr, offstr [, separator[,number_of_bits]]);
```

- 命令说明：将 `bits` 转换为二进制值。其中：
  - 系统会从右到左检查二进制值，如果二进制值为1，则会被替换为 `onstr` 值；如果二进制值为0，则会被替换为 `offstr` 值。
  - 返回值之间由 `separator` 分隔。
  - `number_of_bits` 指定了检查位数，默认值为64。如果指定 `number_of_bits` 大于64，则会被裁剪到64位；如果指定 `number_of_bits` 为-1，则检查位数仍默认为64。
- 输入值类型： `bits` 和 `number_of_bits` 均为BIGINT类型， `onstr` 、 `offstr` 和 `separator` 均为VARCHAR类型。
- 返回值类型：VARCHAR。
- 示例：
  - 将5转换为二进制，从右至左取其前两位的值，其中1用 `a` 表示，0用 `b` 表示， `a` 和 `b` 之间用逗号（,）分隔，语句如下：

```
SELECT export_set(5,'a','b',',',2);
```

返回结果如下：

```
+------------------------------+
| export_set(5,'a','b',',',2) |
+------------------------------+
| a,b                          |
+------------------------------+
```

  - 将6转换为二进制，从右至左取其前十位的值，其中1用 `1` 表示，0用 `0` 表示， `1` 和 `0` 之间用逗号（,）分隔，语句如下：

```
SELECT export_set(6,'1','0',',',10);
```

返回结果如下：

```
+-------------------------------+
| export_set(6,'1','0',',',10) |
+-------------------------------+
| 0,1,1,0,0,0,0,0,0,0           |
+-------------------------------+
```

FIELD

```
FIELD(str, str 1, str 2,..., str n);
```

- 命令说明：返回 `str` 在 `str 1` 、 `str 2` 、 `str n` 列表中的索引位置。 如果未找到 `str` ，则返回 `0` 。
- 输入值类型：VARCHAR。
- 返回值类型：BIGINT。
- 示例：

```
SELECT FIELD('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
```

返回结果如下：

```
+-----------------------------------------+
| FIELD('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff') |
+-----------------------------------------+
|                                       2 |
+-----------------------------------------+
```

FIND_IN_SET

```
find_in_set(str, strlist)
```

- 命令说明：返回 `str` 在列表 `strlist` 中的位置。

  如果 `str` 不在 `strlist` 中或者 `strlist` 是空字符串，返回结果为 `0` 。

  如果 `str` 、 `strlist` 任一参数为 `null` ，返回结果为 `null` 。
- 输入值类型： `str` 和 `strlist` 均为VARCHAR类型。
- 返回值类型：BIGINT。
- 示例：

```
SELECT find_in_set('b','a,b,c,d');
```

返回结果如下：

```
+----------------------------+
| find_in_set('b','a,b,c,d') |
+----------------------------+
|                          2 |
+----------------------------+
```

FORMAT

```
format(X, D)
```

- 命令说明：将数字 `X` 格式化为 `#,###,###.##` 样式，舍入到 `D` 小数位，并将结果作为字符串返回。

  如果 `D` 为 `0` ，则返回结果没有小数点或小数部分。
- 输入值类型： `X` 为DOUBLE类型， `D` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT format(12332.123456, 4)AS result1, format(12332.1,4)AS result2, format(12332.2,0)AS result3;
```

返回结果如下：

```
+-------------+-------------+---------+
| result1     | result2     | result3 |
+-------------+-------------+---------+
| 12,332.1235 | 12,332.1000 | 12,332  |
+-------------+-------------+---------+
```

FROM_BASE64

```
from_base64(x)
```

- 命令说明：解码Base64编码的参数 `x` 并返回结果。
- 输入值类型：VARBINARY或VARCHAR。
- 返回值类型：VARBINARY。

> **说明** 解码后的返回值为VARBINARY类型，您可以通过如下方式将返回结果转换为VARCHAR类型：
>
> - 若AnalyticDB MySQL版集群版本大于或等于3.1.4，您可以使用 `CAST AS VARCHAR` 函数来转换数据类型。更多信息，请参见CAST函数。
> - 若AnalyticDB MySQL版集群版本小于3.1.4，您可以使用 `FROM_UTF8` 函数来转换数据类型。更多信息，请参见FROM_UTF8。

- 示例：

- 输入的参数为VARCHAR类型，语句如下：

```
SELECT from_base64('Q2hpbmE=');
```

返回结果如下：

```
+-----------------------------------------------+
| from_base64('Q2hpbmE=')                       |
+-----------------------------------------------+
| 0x4368696E61                                  |
+-----------------------------------------------+
```

> **说明** 上述语句的返回值为VARBINARY类型，如需解析为原VARCHAR类型，您可以使用如下语句：
>
> ```
> SELECT cast(from_base64('Q2hpbmE=') AS varchar);
> ```
>
> 返回结果如下：
>
> ```
> +-----------------------------------------+
> | cast(from_base64('Q2hpbmE=') AS varchar) |
> +-----------------------------------------+
> | China                                   |
> +-----------------------------------------+
> ```

- 输入的参数为VARBINARY类型，语句如下

```
SELECT from_base64(cast(to_base64('China') AS varbinary));
```

返回结果如下：

```
+-----------------------------------------------------------------------------------------------------------------+
| from_base64(cast(to_base64('China') AS varbinary))                                                              |
+-----------------------------------------------------------------------------------------------------------------+
| 0x4368696E61                                                                                                    |
+-----------------------------------------------------------------------------------------------------------------+
```

FROM_UTF8

```
from_utf8(x)
from_utf8(x, y)
```

- 命令说明：
  - `from_utf8(x)`：解码UTF-8编码的 `x` 并返回结果。
  - `from_utf8(x, y)`：解码非UTF-8编码的参数 `x`，并将其替换为指定的非法字符。

> **说明**
> - `y` 可以不填。若不指定 `y` 则默认返回 `&#xfffd;`。
> - `y` 可以是非法字符本身（例如 `#`），也可以是非法字符所对应的ASCII码（例如 `35`）。

- 输入值类型：`x` 为VARBINARY类型，`y` 为VARCHAR或BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：
  - 解码UTF-8编码的参数并返回结果，语句如下：

```
SELECT from_utf8(to_utf8('hello'));
```

返回结果如下：

```
+-----------------------------+
| from_utf8(to_utf8('hello')) |
+-----------------------------+
| hello                       |
+-----------------------------+
```

  - 解码非UTF-8编码的参数，语句如下：

```
SELECT from_utf8(unhex('58BF'));
```

返回结果如下：

```
+--------------------------+
| from_utf8(unhex('58BF')) |
+--------------------------+
| X&#xfffd;                |
+--------------------------+
```

- 解码非UTF-8编码的参数，并将其替换为非法字符 `#` ，语句如下：

```
SELECT from_utf8(unhex('58BF'), '#');
```

返回结果如下：

```
+------------------------------+
| from_utf8(unhex('58BF'), '#') |
+------------------------------+
| X#                           |
+------------------------------+
```

- 解码非UTF-8编码的参数，并将其替换为非法字符，其中非法字符的ASCII码为35，语句如下：

```
SELECT from_utf8(unhex('58BF'), '35');
```

返回结果如下：

```
+------------------------------+
| from_utf8(unhex('58BF'), '35') |
+------------------------------+
| X#                           |
+------------------------------+
```

HEX

```
hex(x)
```

- 命令说明：将参数 `x` 转换为其所对应的十六进制格式的字符串。
- 输入值类型：BIGINT或VARCHAR。
- 返回值类型：VARCHAR。
- 示例：
  - 输入值类型为BIGINT，语句如下：

```
SELECT hex(16);
```

返回结果如下：

```
+---------+
| hex(16) |
+---------+
| 10      |
+---------+
```

  - 输入值类型为VARCHAR，语句如下：

```
SELECT hex('16');
```

返回结果如下：

```
+-----------+
| hex('16') |
+-----------+
| 3136      |
+-----------+
```

INSTR

```
instr(str, substr)
```

- 命令说明：返回字符串 `str` 中子字符串 `substr` 首次出现的位置。
- 输入值类型： `str` 和 `substr` 均为VARCHAR类型。
- 返回值类型：BIGINT。
- 示例：

```
SELECT instr('foobarbar', 'bar');
```

返回结果如下：

```
+--------------------------+
| instr('foobarbar', 'bar') |
+--------------------------+
|                        4 |
+--------------------------+
```

LEFT

```
LEFT(str, len)
```

- 命令说明：返回字符串 `str` 中最左边的 `len` 个字符。

  如果 `str` 或者 `len` 为 `null` ，则返回结果为 `null` 。
- 输入值类型： `str` 为VARCHAR类型， `len` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT LEFT('foobarbar', 5);
```

返回结果如下：

```
+----------------------+
| LEFT('foobarbar', 5) |
+----------------------+
| fooba                |
+----------------------+
```

LENGTH或OCTET_LENGTH

```
length(str)
octet_length(str)
```

- 命令说明：返回字符串 `str` 的长度。
- 输入值类型：VARCHAR。
- 返回值类型：BIGINT。
- 示例：

```
SELECT length('aliyun');
```

返回结果如下：

```
+------------------+
| length('aliyun') |
+------------------+
|                6 |
+------------------+
```

LIKE

```
expression [NOT] LIKE pattern [ESCAPE 'escape_char']
```

- 命令说明： `LIKE` 运算符用于将字符串 `expression` 与 `pattern` 进行匹配，匹配成功返回 `1` ，匹配失败返回 `0` 。其中：
  - `pattern` 为通配符模式，通配符包括：
    - `%` ：匹配任意长度的字符串。
    - `_` ：匹配单个字符。
  - `escape_char` ：对 `pattern` 中的 `%` 、 `_` 进行转义，使得转义字符后面的 `%` 、 `_` 不作通配符使用。
- 输入值类型： `expression` 、 `pattern` 均为VARCHAR类型。
- 返回值类型：BIGINT。
- 示例：
  - 语句如下：

    ```
    SELECT 'David!' LIKE 'David_' AS result1, 'David!' NOT LIKE 'David_' AS result2,  'David!' LIKE '%D%v%' AS result3;
    ```

    返回结果如下：

    ```
    +---------+---------+---------+
    | result1 | result2 | result3 |
    +---------+---------+---------+
    |       1 |       0 |       1 |
    +---------+---------+---------+
    ```
  - 语句如下：

    ```
    SELECT 'David_' LIKE 'David|_' ESCAPE '|';
    ```

    返回结果如下：

    ```
    +----------------------------------+
    | David_' LIKE 'David|_' ESCAPE '| |
    +----------------------------------+
    |                                1 |
    +----------------------------------+
    ```

LOCATE

```
locate(substr, str)
locate(substr, str, pos)
```

- 命令说明：返回字符串 `str` 中首次出现 `substr` 的位置信息，或者返回字符串 `str` 中从指定位置 `pos` 开始首次出现 `substr` 的位置信息。

  如果 `substr` 不在 `str` 中，返回结果为 `0` 。

  如果 `substr` 或者 `str` 为 `null` ，返回结果为 `null` 。
- 输入值类型： `str` 和 `substr` 均为VARCHAR类型， `pos` 为BIGINT类型。
- 返回值类型：BIGINT。
- 示例：
  - 语句如下：

    ```
    SELECT locate('bar', 'foobarbar');
    ```

    返回结果如下：

    ```
    +-----------------------------+
    | locate('bar', 'foobarbar') |
    +-----------------------------+
    |                           4 |
    +-----------------------------+
    ```
  - 语句如下：

    ```
    SELECT locate('bar', 'foobarbar', 7);
    ```

    返回结果如下：

    ```
    +--------------------------------+
    | locate('bar', 'foobarbar', 7) |
    +--------------------------------+
    |                              7 |
    +--------------------------------+
    ```

LOWER或LCASE

```
lower(str)
lcase(str)
```

- 命令说明：将字符串 `str` 中的字母转换为小写。
- 返回值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：

  ```
  SELECT lower('Aliyun');
  ```

  返回结果如下：

  ```
  +-----------------+
  | lower('Aliyun') |
  +-----------------+
  | aliyun          |
  +-----------------+
  ```

LPAD

```
lpad(str, len, padstr)
```

- 命令说明：将字符串 `str` 左边拼接 `padstr` 直到长度达到 `len` ，并返回拼接后的字符串。

  如果 `str` 长于 `len` ，则返回值将缩短为 `len` 个字符。
- 输入值类型： `str` 和 `padstr` 均为VARCHAR类型， `len` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：

  ```
  SELECT lpad('Aliyun',9,'#');
  ```

  返回结果如下：

  ```
  +----------------------+
  | lpad('Aliyun',9,'#') |
  +----------------------+
  | ###Aliyun            |
  +----------------------+
  ```

LTRIM

```
ltrim(str)
```

- 命令说明：删除字符串 `str` 所有前导空格。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：

  ```
  SELECT ltrim('  abc');
  ```

  返回结果如下：

  ```
  +----------------+
  | ltrim('  abc') |
  +----------------+
  | abc            |
  +----------------+
  ```

MAKE_SET

```
make_set(bits, str 1, str 2,...);
```

- 命令说明：返回一个设置值（包含由字符分隔的子字符串的字符串），其中包含具有相应位设置的字符串。

  `str 1` 对应于 `0` 位，`str 2` 对应于 `1` 位，依此类推。`str 1`，`str 2`，...中的 `null` 值不会附加到结果中。
- 输入值类型：`bits` 为BIGINT类型，`str` 为VARCHAR类型。
- 返回值类型：VARCHAR。
- 示例：
  - 语句如下：

    ```
    SELECT make_set(5,'hello','nice','world');
    ```

    返回结果如下：

    ```
    +-----------------------------------+
    | make_set(5,'hello','nice','world') |
    +-----------------------------------+
    | hello,world                       |
    +-----------------------------------+
    ```

  - 语句如下：

    ```
    SELECT make_set(1 | 4,'hello','nice',NULL,'world')AS result;
    ```

    返回结果如下：

    ```
    +--------+
    | result |
    +--------+
    | hello  |
    +--------+
    ```

MID

```
mid(str, pos, len)
```

- 命令说明：与SUBSTR或SUBSTRING功能相同，从字符串 `str` 的 `pos` 开始返回 `len` 长度的子字符串。
- 输入值类型：`str` 为VARCHAR类型，`pos` 和 `len` 均为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：
  - 语句如下：

    ```
    SELECT mid('Quadratically',5,6);
    ```

    返回结果如下：

    ```
    +--------------------------+
    | mid('Quadratically',5,6) |
    +--------------------------+
    | ratica                   |
    +--------------------------+
    ```

- 语句如下：

```
SELECT mid('Sakila', -5, 3);
```

返回结果如下：

```
+----------------------+
| mid('Sakila', -5, 3) |
+----------------------+
| aki                  |
+----------------------+
```

OCT

```
oct(N)
```

- 命令说明：返回整数 `N` 的八进制字符串表示形式。

  如果 `N` 为 `null`，返回结果为 `null`。
- 输入值类型：BIGINT。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT oct(12);
```

返回结果如下：

```
+---------+
| oct(12) |
+---------+
| 14      |
+---------+
```

ORD

```
ord(x)
```

- 命令说明：如果字符串 `x` 最左边的字符是多字节字符，则返回该字符的代码。
- 输入值类型：VARBINARY或VARCHAR。
- 返回值类型：LONG。
- 示例：
  - 输入的参数为VARCHAR类型，语句如下：

```
SELECT ord('China');
```

返回结果如下：

```
+--------------+
| ord('China') |
+--------------+
|           67 |
+--------------+
```

  - 输入的参数为VARBINARY类型，语句如下

```
SELECT ord(cast('China' AS varbinary));
```

返回结果如下：

```
+---------------------------------+
| ord(cast('China' AS varbinary)) |
+---------------------------------+
|                              67 |
+---------------------------------+
```

POSITION

```
position(substr IN str);
```

- 命令说明： 返回字符串 `str` 中子字符串 `substr` 首次出现位置，位置从 `1` 开始，如果未找到则返回 `0`。
- 输入值类型： `substr` 和 `str` 均为VARCHAR类型。
- 返回值类型：BIGINT。
- 示例：

```
SELECT position('bar' in 'foobarbar');
```

返回结果如下：

```
+--------------------------------+
| position('bar' in 'foobarbar') |
+--------------------------------+
|                              4 |
+--------------------------------+
```

REPEAT

```
repeat(str, count);
```

- 命令说明：返回由字符串 `str` 重复 `count` 次数组成的字符串。

  如果 `count<1` ，则返回空字符串。

  如果 `str` 或 `count` 为 `null` ，则返回 `null` 。
- 输入值类型： `str` 为VARCHAR类型， `count` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：
  - 语句如下：

    ```
    SELECT repeat('a', 3);
    ```

    返回结果如下：

    ```
    +----------------+
    | repeat('a', 3) |
    +----------------+
    | aaa            |
    +----------------+
    ```
  - 语句如下：

    ```
    SELECT repeat('abc', null);
    ```

    返回结果如下：

    ```
    +----------------------+
    | repeat('abc', null) |
    +----------------------+
    | NULL                |
    +----------------------+
    ```
  - 语句如下：

    ```
    SELECT repeat(null, 3);
    ```

    返回结果如下：

    ```
    +-----------------+
    | repeat(null, 3) |
    +-----------------+
    | NULL            |
    +-----------------+
    ```

REPLACE

```
replace(str, from_str, to_str);
```

- 命令说明：将 `str` 中的 `from_str` 内容替换为 `to_str` 。
- 输入值类型： `str` 、 `from_str` 和 `to_str` 均为VARCHAR类型。
- 返回值类型：VARCHAR。
- 示例：

  ```
  SELECT replace('WWW.aliyun.com', 'W', 'w');
  ```

  返回结果如下：

  ```
  +-------------------------------------+
  | replace('WWW.aliyun.com', 'W', 'w') |
  +-------------------------------------+
  | www.aliyun.com                      |
  +-------------------------------------+
  ```

REVERSE

```
reverse(str);
```

- 命令说明：返回 `str` 逆序后的字符串。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT reverse('123456');
```

返回结果如下：

```
+-------------------+
| reverse('123456') |
+-------------------+
| 654321            |
+-------------------+
```

RIGHT

```
RIGHT(str, len);
```

- 命令说明：返回字符串 `str` 中最右边的 `len` 个字符。

  如果 `str` 或者 `len` 为 `null` ，返回结果为 `null` 。
- 输入值类型： `str` 为VARCHAR类型， `len` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT RIGHT('abc',3);
```

返回结果如下：

```
+----------------+
| RIGHT('abc',3) |
+----------------+
| abc            |
+----------------+
```

RLIKE或REGEXP

```
expression RLIKE pattern;
expression REGEXP pattern;
```

- 命令说明：将字符串 `expression` 与 `pattern` 进行正则匹配，匹配成功返回 `1` ，否则返回 `0` 。

  如果 `expression` 或者 `pattern` 为 `null` ，返回结果为 `null` 。
- 输入值类型： `expression` 和 `pattern` 均为VARCHAR类型。
- 返回值类型：BOOLEAN。
- 示例：
  - 语句如下：

```
SELECT 'Michael!' REGEXP '.*';
```

返回结果如下：

```
+----------------------+
| Michael!' REGEXP '.* |
+----------------------+
|                    1 |
+----------------------+
```

  - 语句如下：

```
SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
```

返回结果如下：

```
+-------------------------------------+
| new*\n*line' REGEXP 'new\\*.\\*line |
+-------------------------------------+
|                                   0 |
+-------------------------------------+
```

- 语句如下：

```
SELECT 'c' REGEXP '^[a-d]';
```

返回结果如下：

```
+-------------------+
| c' REGEXP '^[a-d] |
+-------------------+
|                 1 |
+-------------------+
```

RPAD

```
rpad(str, len, padstr)
```

- 命令说明：将字符串 `str` 右边拼接 `padstr` 直到长度达到 `len` ，并返回拼接后的字符串。

  如果 `str` 长于 `len` ，则返回值将缩短为 `len` 个字符。
- 输入值类型： `str` 和 `padstr` 均为VARCHAR类型， `len` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT rpad('Aliyun',9,'#');
```

返回结果如下：

```
+----------------------+
| rpad('Aliyun',9,'#') |
+----------------------+
| Aliyun###            |
+----------------------+
```

RTRIM

```
rtrim(str)
```

- 命令说明：删除字符串 `str` 所有后置空格。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT rtrim('barbar   ');
```

返回结果如下：

```
+--------------------+
| rtrim('barbar   ') |
+--------------------+
| barbar             |
+--------------------+
```

SPACE

```
space(N);
```

- 命令说明：返回由指定数量空格组成的字符串。

> **说明** 您可以将该函数与 `contact()` 函数组合使用，方便展示返回结果。

- 输入值类型：BIGINT。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT concat("#", space(6), "#");
```

返回结果如下：

```
+----------------------------+
| concat("#", space(6), "#") |
+----------------------------+
| #      #                   |
+----------------------------+
```

SPLIT

```
split(string, delimiter)
```

- 命令说明：将字符串 `string` 按分隔符 `delimiter` 进行分隔，并返回数组。
- 输入值类型：`string` 和 `delimiter` 均为VARCHAR类型。
- 返回值类型：ARRAY<varchar>。
- 示例：

```
SELECT split('1#2#3', '#'), split('#1#2#3#', '#'),  split('123', '#');
```

返回结果如下：

```
+---------------------+----------------------+-------------------+
| split('1#2#3', '#') | split('#1#2#3#', '#') | split('123', '#') |
+---------------------+----------------------+-------------------+
| ["1","2","3"]       | ["","1","2","3",""]   | ["123"]           |
+---------------------+----------------------+-------------------+
```

SPLIT_PART

```
split_part(string, delimiter, index)
```

- 命令说明：将字符串 `string` 按分隔符 `delimiter` 分隔，并返回分隔后数组下标为 `index` 的子串。 `index` 以1开头，如果大于字段数则返回NULL。
- 输入值类型：`string` 和 `delimiter` 均为VARCHAR类型， `index` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT split_part('A#B#C', '#', 2), split_part('A#B#C', '#', 4);
```

返回结果如下：

```
+-----------------------------+-----------------------------+
| split_part('A#B#C', '#', 2) | split_part('A#B#C', '#', 4) |
+-----------------------------+-----------------------------+
| B                           | NULL                        |
+-----------------------------+-----------------------------+
```

SPLIT_TO_MAP

```
split_to_map(string, entryDelimiter, keyValueDelimiter)
```

- 命令说明：通过 `entryDelimiter` 和 `keyValueDelimiter` 拆分字符串并返回 `map` 。 `entryDelimiter` 将字符串分解为 `key-value` 对， `keyValueDelimiter` 将每对 `key-value` 分隔成 `key` 和 `value` 。
- 输入值类型：`string` 、 `entryDelimiter` 和 `keyValueDelimiter` 均为VARCHAR类型。
- 返回值类型：MAP<varchar, varchar>。
- 示例：

```
SELECT split_to_map('k1:v1,k2:v2', ',', ':'),split_to_map('', ',', ':');
```

返回结果如下：

```
+---------------------------------------+---------------------------+
| split_to_map('k1:v1,k2:v2', ',', ':') | split_to_map('', ',', ':') |
+---------------------------------------+---------------------------+
| {"k1":"v1","k2":"v2"}                 | {}                        |
+---------------------------------------+---------------------------+
```

STRCMP

```
strcmp(str 1, str 2);
```

- 命令说明：如果字符串 `str 1` 、 `str 2` 相同，返回结果为 `0` 。如果 `str 1` 根据当前排序顺序小于 `str 2` ，返回结果为 `-1` ，否则返回结果为 `1` 。
- 输入值类型：`str 1` 和 `str 2` 均为VARCHAR类型。
- 返回值类型：BIGINT。
- 示例：

```
SELECT strcmp('text', 'text2');
```

返回结果如下：

```
+--------------------------+
| strcmp('text', 'text2') |
+--------------------------+
|                       -1 |
+--------------------------+
```

SUBSTR或SUBSTRING

```
substr(str, pos)
substr(str FROM pos)
substr(str, pos, len)
substr(str FROM pos FOR len)
substring(str, pos)
substring(str FROM pos)
substring(str, pos, len)
substring(str FROM pos FOR len)
```

- 命令说明：
  - `SUBSTRING(varchar str, bigint pos)`、`SUBSTRING(varchar str FROM pos)`返回从`pos`位置开始到字符串结束的子串。如果`pos<0`，则起始位置从字符串的末尾开始倒数。
  - `SUBSTRING(varchar str, bigint pos, bigint len)`、`SUBSTRING(varchar str FROM pos FOR len)`返回从`pos`位置开始长度为`len`的子串。如果`pos<0`，则起始位置从字符串的末尾开始倒数。
- 输入值类型：`str`为VARCHAR类型，`pos`和`len`均为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：
  - 语句如下：

    ```
    SELECT substr('helloworld', 6);
    ```

    返回结果如下：

    ```
    +--------------------------+
    | substr('helloworld', 6) |
    +--------------------------+
    | world                    |
    +--------------------------+
    ```

  - 语句如下：

    ```
    SELECT substr('helloworld' FROM 6);
    ```

    返回结果如下：

    ```
    +------------------------------+
    | substr('helloworld' FROM 6) |
    +------------------------------+
    | world                        |
    +------------------------------+
    ```

  - 语句如下：

    ```
    SELECT substr('helloworld', 6, 3);
    ```

    返回结果如下：

    ```
    +-----------------------------+
    | substr('helloworld', 6, 3) |
    +-----------------------------+
    | wor                         |
    +-----------------------------+
    ```

  - 语句如下：

    ```
    SELECT substr('helloworld' from 6 for 3);
    ```

    返回结果如下：

    ```
    +-----------------------------------+
    | substr('helloworld' FROM 6 FOR 3) |
    +-----------------------------------+
    | wor                               |
    +-----------------------------------+
    ```

SUBSTRING_INDEX

```
substring_index(str, delim, count)
```

- 命令说明：返回字符串 `str` 中最后一次分隔符 `delim` 出现之前的子字符串。

  如果 `count>0` ，返回最后一次 `delim` 左侧的所有内容，即从左侧开始计算。

  如果 `count<0` ，返回最后一次 `delim` 右侧的所有内容，即从右侧开始计算。

  搜索 `delim` 时， `SUBSTRING_INDEX` 函数区分大小写。
- 输入值类型： `str` 和 `delim` 均为VARCHAR类型， `count` 为BIGINT类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT substring_index('www.aliyun.com', '.', 2);
```

返回结果如下：

```
+-----------------------------------------+
| substring_index('www.aliyun.com', '.', 2) |
+-----------------------------------------+
| www.aliyun                              |
+-----------------------------------------+
```

TO_BASE64

```
to_base64(x)
```

- 命令说明：返回参数 `x` 的Base64编码形式。
- 输入值类型：VARBINARY或VARCHAR。
- 返回值类型：VARCHAR。
- 示例：
  - 输入的参数为VARCHAR类型，语句如下：

    ```
    SELECT to_base64('China');
    ```

    返回结果如下：

    ```
    +--------------------+
    | to_base64('China') |
    +--------------------+
    | Q2hpbmE=           |
    +--------------------+
    ```
  - 输入的参数为VARBINARY类型，语句如下

    ```
    SELECT to_base64(cast('China' AS varbinary));
    ```

    返回结果如下：

    ```
    +---------------------------------------+
    | to_base64(cast('China' AS varbinary)) |
    +---------------------------------------+
    | Q2hpbmE=                              |
    +---------------------------------------+
    ```

TO_UTF8

```
to_utf8(x)
```

- 命令说明：返回参数 `x` 的UTF-8编码形式。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT to_utf8('China');
```

返回结果如下：

```
+-------------------------------------+
| to_utf8('China')                    |
+-------------------------------------+
| 0x4368696E61                        |
+-------------------------------------+
```

TRIM

```
trim([remstr FROM] str)
trim([{BOTH | LEADING | TRAILING} [remstr] FROM] str)
```

- 命令说明：通过删除前导空格和尾随空格或删除与可选的指定字符串 `remstr` 匹配的字符来剪裁字符串 `str` 。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：
  - 语句如下：

    ```
    SELECT trim('  bar   ');
    ```

    返回结果如下：

    ```
    +------------------+
    | trim('  bar   ') |
    +------------------+
    | bar              |
    +------------------+
    ```

  - 语句如下：

    ```
    SELECT trim(BOTH 'x' FROM 'xxxbarxxx');
    ```

    返回结果如下：

    ```
    +---------------------------------+
    | trim(BOTH 'x' FROM 'xxxbarxxx') |
    +---------------------------------+
    | bar                             |
    +---------------------------------+
    ```

  - 语句如下：

    ```
    SELECT trim(LEADING 'x' FROM 'xxxbarxxx');
    ```

    返回结果如下：

    ```
    +------------------------------------+
    | trim(LEADING 'x' FROM 'xxxbarxxx') |
    +------------------------------------+
    | barxxx                             |
    +------------------------------------+
    ```

  - 语句如下：

    ```
    SELECT trim(TRAILING 'x' from 'xxxbarxxx');
    ```

    返回结果如下：

    ```
    +------------------------------------+
    | trim(TRAILING 'x' from 'xxxbarxxx') |
    +------------------------------------+
    | xxxbar                             |
    +------------------------------------+
    ```

UPPER或UCASE

```
upper(str)
ucase(str)
```

- 命令说明：将字符串 `str` 中的字母转换为大写。
- 输入值类型：VARCHAR。
- 返回值类型：VARCHAR。
- 示例：

  ```
  SELECT upper('Aliyun');
  ```

  返回结果如下：

  ```
  +-----------------+
  | upper('Aliyun') |
  +-----------------+
  | ALIYUN          |
  +-----------------+
  ```

UNHEX

```
unhex(x);
```

- 命令说明：将参数 `x` 中的每对十六进制数字解释为一个数字，并将其转换为该数字表示的字符。
- 输入值类型：VARBINARY或VARCHAR。
- 返回值类型：VARBINARY。

> **说明**
> - 解码后的返回值为VARBINARY类型，您可以通过如下方式将返回结果转换为VARCHAR类型：
>   - 若AnalyticDB MySQL版集群版本大于或等于3.1.4，您可以使用 `CAST AS VARCHAR` 函数来转换数据类型。更多信息，请参见CAST函数。
>   - 若AnalyticDB MySQL版集群版本小于3.1.4，您可以使用 `FROM_UTF8` 函数来转换数据类型。更多信息，请参见FROM_UTF8。
> - 如果UNHEX的输入值中包含任何非十六进制数字，则返回NULL。

- 示例：
  - 输入的参数为VARCHAR类型，语句如下：

```
SELECT unhex(hex('China'));
```

返回结果如下：

```
+----------------------------------------+
| unhex(hex('China'))                    |
+----------------------------------------+
| 0x4368696E61                           |
+----------------------------------------+
```

> **说明** 上述语句的返回值为VARBINARY类型，如需解析为原VARCHAR类型，您可以使用如下语句：
>
> ```
> SELECT cast(unhex(hex('China')) AS varchar);
> ```
>
> 返回结果如下：
>
> ```
> +--------------------------------------+
> | cast(unhex(hex('China')) AS varchar) |
> +--------------------------------------+
> | China                                |
> +--------------------------------------+
> ```

  - 输入的参数为VARBINARY类型，语句如下

```
SELECT unhex(cast(hex('China') AS varbinary));
```

返回结果如下：

```
+-------------------------------------------------------------------------------+
| unhex(cast(hex('China') AS varbinary))                                        |
+-------------------------------------------------------------------------------+
| 0x4368696E61                                                                  |
+-------------------------------------------------------------------------------+
```

4.5. 位函数和操作符

AnalyticDB for MySQL支持以下位函数和操作符。

- BIT_COUNT：系统先将参数转换为二进制，然后返回二进制中1的个数。
- &：按位AND。
- ~：反转所有位。
- |：按位OR。
- ^：按位异或。
- >>（BITWISE_RIGHT_SHIFT）：向右移位。
- <<（BITWISE_LEFT_SHIFT）：向左移位。

BIT_COUNT

```
bit_count(bigint x)
bit_count(double x)
bit_count(varchar x)
```

- 命令说明：系统先将参数转换为二进制，然后返回二进制中 `1` 的个数。
- 返回值类型：BIGINT。
- 示例：

```
select bit_count(2);
+--------------+
| bit_count(2) |
+--------------+
|            1 |
```

```
select bit_count(pi());
+-----------------+
| bit_count(pi()) |
+-----------------+
|               2 |
```

```
select bit_count('123');
+------------------+|
bit_count('123') |
+------------------+
|                6 |
```

&

- 命令说明：按位 `AND` 。
- 返回值类型：BIGINT。
- 示例：

```
 select 12 & 15;
+---------------------+
| bitwise_and(12, 15) |
+---------------------+
|                  12 |
```

~

- 命令说明：反转所有位。
- 返回值类型：BIGINT。
- 示例：

```
select 2 & ~1;
+--------------------------------+
| bitwise_and(2, bitwise_not(1)) |
+--------------------------------+
|                              2 |
```

|

- 命令说明：按位 `OR` 。
- 返回值类型：BIGINT。
- 示例：

```
select 29 | 15;
+--------------------+
| bitwise_or(29, 15) |
+--------------------+
|                 31
```

^

- 命令说明：按位异或。
- 返回值类型：BIGINT。
- 示例：

```
select 1 ^ 10;
+--------------------+
| bitwise_xor(1, 10) |
+--------------------+
|                 11 |
```

>>（BITWISE_RIGHT_SHIFT）

```
bitwise_right_shift(double x, double y)
bitwise_right_shift(varchar x, varchar y)
bitwise_right_shift(bigint x, bigint y)
```

- 命令说明：向右移位。
- 返回值类型：BIGINT。
- 示例：

```
select 3 >> 2;
+---------------------------+
| bitwise_right_shift(3, 2) |
+---------------------------+
|                         0 |
```

```
select 3.4 >> 23.2;
+--------------------------------+
| bitwise_right_shift(3.4, 23.2) |
+--------------------------------+
|                              0 |
```

<<（BITWISE_LEFT_SHIFT）

```
bitwise_left_shift(double x, double y)
bitwise_left_shift(varchar x, varchar y)
bitwise_left_shift(bigint x, bigint y)
```

- 命令说明：向左移位。
- 返回值类型：BIGINT。
- 示例：

```
SELECT 3 << 2;
+--------------------------+
| bitwise_left_shift(3, 2) |
+--------------------------+
|                       12 |
```

```
select '3' << '2';
+------------------------------+
| bitwise_left_shift('3', '2') |
+------------------------------+
|                           12 |
```

```
select 3.4 << 23.2;
+--------------------------------+
| bitwise_left_shift(3.4, 23.2) |
+--------------------------------+
|                       25165824 |
```

4.6. GEO函数

4.6.1. 操作函数

使用操作函数可对geometry数据类型值执行操作。

- ST_Intersection：返回g1与g2两个图对象的交集。
- ST_Envelope：获取作为输入line、polygon、multiline和multipolygon geometry数据类型。不支持point geometry数据类型。返回一个信封的二进制表示，其中信封是一个围绕指定geometry数据类型的矩形。
- ST_Union：返回表示指定几何体的点集并集的几何体数据类型。
- geometry_union：返回表示指定几何体的点集并集的几何体数据类型。
- ST_Boundary：采用一个geometry数据类型作为输入，并返回该boundary geometry数据类型的二进制表示。
- ST_EnvelopeAsPts：返回两个点的数组，表示几何体的边界矩形多边形的左下角和右上角。如果指定的几何体为空，则返回null。
- ST_Difference：返回左几何体和右几何体之间的差异的几何体。
- ST_ExteriorRing：返回输入类型polygon的外部环的几何体。
- ST_SymDifference：返回左几何体和右几何体之间的几何对称差异的几何体。

ST_Intersection

```
ST_Intersection(g1, g2)
```

- 命令说明：返回g1与g2两个图对象的交集。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_Intersection(ST_GeometryFromText('MULTIPOINT (50 100, 50 200)'), ST_GeometryFromText('Point (50 100)'));
+-----------------------------------------------------------------------------------------------------------------+
| ST_Intersection(ST_GeometryFromText('MULTIPOINT (50 100, 50 200)'), ST_GeometryFromText('Point (50 100)')) |
+-----------------------------------------------------------------------------------------------------------------+
|                          ?       ?                                                                              |
```

```
SELECT ST_AsText(ST_Intersection(ST_GeometryFromText('MULTIPOINT (50 100, 50 200)'), ST_GeometryFromText('Point (50 100)')));
+-------------------------------------------------------------------------------------------------------------------------+
| ST_AsText(ST_Intersection(ST_GeometryFromText('MULTIPOINT (50 100, 50 200)'), ST_GeometryFromText('Point (50 100)'))) |
+-------------------------------------------------------------------------------------------------------------------------+
|                         POINT (50 100)                                                                                  |
```

ST_Envelope

```
ST_Envelope(g)
```

- 命令说明：获取作为输入line、polygon、multiline和multipolygon geometry数据类型。不支持point geometry数据类型。返回一个信封的二进制表示，其中信封是一个围绕指定geometry数据类型的矩形。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_Envelope(ST_GeometryFromText('LINESTRING (1 1, 2 2, 1 3)')))  +
*------------------------------------------------------------------------------+
|ST_AsText(ST_Envelope(ST_GeometryFromText('LINESTRING (1 1, 2 2, 1 3)')))        |
+------------------------------------------------------------------------------+
|               POLYGON ((1 1, 2 1, 2 3, 1 3, 1 1))                              |
```

```
SELECT ST_AsText(ST_Envelope(ST_GeometryFromText('MULTIPOINT (1 2, 2 4, 3 6, 4 8)')));
+------------------------------------------------------------------------------+
| ST_AsText(ST_Envelope(ST_GeometryFromText('MULTIPOINT (1 2, 2 4, 3 6, 4 8)')))|
+------------------------------------------------------------------------------+
|                       POLYGON ((1 2, 4 2, 4 8, 1 8, 1 2))                    |
```

ST_Union

```
ST_Union(g1, g2)
```

- 命令说明：返回表示指定几何体的点集并集的几何体数据类型。
- 返回值类型：int。
- 示例：

```
SELECT ST_ASText(ST_Union(ST_GeometryFromText('MULTIPOLYGON (((1 1, 3 1, 3 3, 1 3, 1 1)))'), ST_GeometryFromText('MULTIPOLYGON (((2 2, 4
2, 4 4, 2 4, 2 2)))')));
+------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| ST_ASText(ST_Union(ST_GeometryFromText('MULTIPOLYGON (((1 1, 3 1, 3 3, 1 3, 1 1)))'), ST_GeometryFromText('MULTIPOLYGON (((2 2, 4 2, 4
4, 2 4, 2 2)))')))      |
+------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
|                                POLYGON ((1 1, 3 1, 3 2, 4 2, 4 4, 2 4, 2 3, 1 3, 1 1))
|
```

geometry_union

```
geometry_union(array[g1, g2, ...])
```

- 命令说明：返回表示指定几何体的点集并集的几何体数据类型。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT geometry_union(ARRAY[ST_Point(61.56, -158.54), ST_Point(61.56, -158.55)]);
+------------------------------------------------------------------------------+
|  geometry_union(ARRAY[ST_Point(61.56, -158.54), ST_Point(61.56, -158.55)])   |
+------------------------------------------------------------------------------+
|                  ?       ?                                                   |
```

ST_Boundary

```
ST_Boundary(g)
```

- 命令说明：采用一个geometry数据类型作为输入，并返回该boundary geometry数据类型的二进制表示。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_Boundary(ST_GeometryFromText('LINESTRING (8 4, 5 7)')))  +
*------------------------------------------------------------------------------+
|ST_AsText(ST_Boundary(ST_GeometryFromText('LINESTRING (8 4, 5 7)')))        |
+------------------------------------------------------------------------------+
|               MULTIPOINT ((8 4), (5 7))                                     |
```

ST_EnvelopeAsPts

```
ST_EnvelopeAsPts(g)
```

- 命令说明：返回两个点的数组，表示几何体的边界矩形多边形的左下角和右上角。如果指定的几何体为空，则返回null。
- 返回值类型：Array[Geometry]。
- 示例：

```
SELECT ST_EnvelopeAsPts(ST_GeometryFromText('LINESTRING EMPTY'))  +
*--------------------------------------------------------------+
|ST_EnvelopeAsPts(ST_GeometryFromText('LINESTRING EMPTY'))      |
+--------------------------------------------------------------+
|               null                                           |
```

ST_Difference

```
ST_EnvelopeAsPts(g1, g2)
```

- 命令说明：返回左几何体和右几何体之间的差异的几何体。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_Difference(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)')))  +
*-----------------------------------------------------------------------------------------------------------+
|ST_AsText(ST_Difference(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)')))      |
+-----------------------------------------------------------------------------------------------------------+
|               POINT (50 100)                                                                              |
```

ST_ExteriorRing

```
ST_ExteriorRing(g1)
```

- 命令说明：返回输入类型polygon的外部环的几何体。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_ExteriorRing(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 1))')))  +
*-----------------------------------------------------------------------------------+
|ST_AsText(ST_ExteriorRing(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 1))')))      |
+-----------------------------------------------------------------------------------+
|               LINESTRING (1 1, 4 1, 1 4, 1 1)                                    |
```

ST_SymDifference

```
ST_SymDifference(g1, g2)
```

- 命令说明：返回左几何体和右几何体之间的几何对称差异的几何体。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_SymDifference(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (50 150)')))  +
*-------------------------------------------------------------------------------------------------------------+
|ST_AsText(ST_SymDifference(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (50 150)')))      |
+-------------------------------------------------------------------------------------------------------------+
|               MULTIPOINT ((50 100), (50 150))                                                               |
```

4.6.2. 空间关系函数

使用空间关系函数可以对几何体的空间关系进行判断。

- ST_Contains：当g2的所有点都在g1的范围中，返回TRUE，否则返回FALSE。
- ST_Crosses：当且仅当左几何体穿过右几何体时返回TRUE。
- ST_Disjoint：当且仅当左几何体和右几何体的交集为空时返回TRUE。
- ST_Equals：当且仅当左几何体等于右几何体时返回TRUE。
- ST_Intersects：当且仅当左几何体与右几何体相交时返回 TRUE。
- ST_Overlaps：当且仅当左几何体与右几何体重叠时返回TRUE。
- ST_Relate：当且仅当左几何体与右几何体具有指定的尺寸扩展九交集模型 （TRUEDE-9IM） 关系时返回 。第三个 （varchar） 输入接受此关系。
- ST_Touches：当且仅当左几何体与右几何体接触时返回TRUE。
- ST_Within：当且仅当左几何体位于右几何体内时返回TRUE。

ST_Contains

```
ST_Contains(g1, g2)
```

- 命令说明：当g2的所有点都在g1的范围中，返回TRUE，否则返回FALSE。

> **说明** g1的范围不包括其边界。

- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Contains(ST_GeometryFromText('POLYGON (0 0, 0 4, 4 4, 4 0)'), ST_GeometryFromText('POINT (2 2)'));
+-----------------------------------------------------------------------------------------+
|ST_Contains(ST_GeometryFromText('POLYGON (0 0, 0 4, 4 4, 4 0)'), ST_GeometryFromText('POINT (2 2)'))|
+-----------------------------------------------------------------------------------------+
|                       true                                                              |
```

ST_Crosses

```
ST_Crosses(g1, g2)
```

- 命令说明：当且仅当左几何体穿过右几何体时返回TRUE。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Crosses(ST_GeometryFromText('POINT (20 20)'), ST_GeometryFromText('POINT (25 25)'))  +
*-----------------------------------------------------------------------------------------------+
|ST_Crosses(ST_GeometryFromText('POINT (20 20)'), ST_GeometryFromText('POINT (25 25)'))        |
+-----------------------------------------------------------------------------------------------+
|               false                                                                           |
```

ST_Disjoint

```
ST_Disjoint(g1, g2)
```

- 命令说明：当且仅当左几何体和右几何体的交集为空时返回TRUE。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Disjoint(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))  +
*-----------------------------------------------------------------------------------------------+
|ST_Disjoint(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))        |
+-----------------------------------------------------------------------------------------------+
|                true                                                                           |
```

ST_Equals

```
ST_Equals(g1, g2)
```

- 命令说明：当且仅当左几何体等于右几何体时返回TRUE。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Equals(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))  +
*-----------------------------------------------------------------------------------------------+
|ST_Equals(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))        |
+-----------------------------------------------------------------------------------------------+
|                false                                                                           |
```

ST_Intersects

```
ST_Intersects(g1, g2)
```

- 命令说明：当且仅当左几何体与右几何体相交时返回TRUE。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Intersects(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))  +
*-----------------------------------------------------------------------------------------------+
|ST_Intersects(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))        |
+-----------------------------------------------------------------------------------------------+
|                false                                                                           |
```

ST_Overlaps

```
ST_Overlaps(g1, g2)
```

- 命令说明：当且仅当左几何体与右几何体重叠时返回TRUE。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Overlaps(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))'), ST_GeometryFromText('POLYGON ((3 3, 3 5, 5 5, 5 3))'))  +
*-----------------------------------------------------------------------------------------------------------------------------+
|ST_Overlaps(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))'), ST_GeometryFromText('POLYGON ((3 3, 3 5, 5 5, 5 3))'))        |
+-----------------------------------------------------------------------------------------------------------------------------+
|                true                                                                                                         |
```

ST_Relate

```
ST_Relate(g1, g2, s1)
```

- 命令说明：当且仅当左几何体与右几何体具有指定的尺寸扩展九交集模型 （TRUEDE-9IM） 关系时返回 。第三个 （varchar） 输入接受此关系。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Relate(ST_GeometryFromText('LINESTRING (0 0, 3 3)'), ST_GeometryFromText('LINESTRING (1 1, 4 1)'), '****T****')  +
*-----------------------------------------------------------------------------------------------------------------------------+
|ST_Relate(ST_GeometryFromText('LINESTRING (0 0, 3 3)'), ST_GeometryFromText('LINESTRING (1 1, 4 1)'), '****T****')        |
+-----------------------------------------------------------------------------------------------------------------------------+
|                false                                                                                                        |
```

ST_Touches

```
ST_Touches(g1, g2)
```

- 命令说明：当且仅当左几何体与右几何体接触时返回TRUE。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Touches(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))  +
*-----------------------------------------------------------------------------------------------+
|ST_Touches(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))      |
+-----------------------------------------------------------------------------------------------+
|                false                                                                          |
```

ST_Within

```
ST_Within(g1, g2)
```

- 命令说明：当且仅当左几何体位于右几何体内时返回TRUE。
- 返回值类型：Boolean。
- 示例：

```
SELECT ST_Within(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))  +
*-----------------------------------------------------------------------------------------------+
|ST_Within(ST_GeometryFromText('POINT (50 100)'), ST_GeometryFromText('POINT (150 150)'))      |
+-----------------------------------------------------------------------------------------------+
|                false                                                                          |
```

4.6.3. 访问器函数

访问器函数可用于从不同的字符串中获取bigint、double或geometry类型的值。

- ST_XMax：返回g1的X坐标上的最大值。
- ST_YMax：返回g1的Y坐标上的最大值。
- ST_XMin：返回g1的X坐标上的最小值。
- ST_YMin：返回g1的Y坐标上的最小值。
- ST_Distance：返回g1、g2之间的直线距离。
- ST_Distance_Sphere：返回g1、g2之间的球面距离，可以指定球的半径radius，radius默认值为6370986米。
- ST_Area：返回g1的欧几里得2维空间值。
- ST_Centroid：返回g1的中心点对象。
- ST_ConvexHull：返回g1的中最小化的凸geometry对象。
- ST_CoordDim：返回g1的中坐标尺寸。
- ST_Dimension：返回g1中固有的dimension，肯定小于或者等于coordinate dimension。
- ST_IsClosed：如果LineString或者Muti-LineString的start和end是一致的，返回true。
- ST_IsEmpty：如果g1是一个空的geometrycollection，或polygon，point，返回true。
- ST_IsValid：g1是不是合法的格式。
- geometry_invalid_reason：返回g1是不是合法的格式的原因。
- ST_Length：如果输入是LineString或者Multi-LineString的长度（计算是基于Euclidean）； 如果输入是GEOGRAPHY类型的格式，返回的是球面上的greate-circle的长度。
- line_locate_point：返回一个float值，标识的是给定点在线上的比率。

- line_interpolate_point：给定一个比率和线，返回这个比率的点。
- line_interpolate_points：给定一个比率和线，返回按照这个比率分割的所有点。
- ST_NumInteriorRing：返回polygon的内部环的集合的基数。
- ST_InteriorRings：返回在指定几何体中找到的所有内部环的几何体数组，如果多边形没有内部环，则返回空数组。如果指定的几何体为空，则返回 null。如果指定的几何体不是多边形，则会引发错误。
- ST_NumGeometries：以整数形式返回集合中的几何体数。如果几何体是几何体的集合 （例如 GEOMETRYCOLLECTION 或MULTI* 对象），则返回几何体的数量。单个几何体返回 1；空几何体返回0。对象中的空几何体计为一个几何体。
- ST_GeometryN：返回位于指定整数索引的 GEOMETRY元素（作为 GEOMETRY 数据类型）。索引从1开始。如果指定的几何体是几何体集合（例如，GEOMETRYCOLLECTION或MULTI*对象），则将返回指定索引处的几何体。如果指定的索引小于1或大于集合中的元素总数，则返回null。
- ST_PointN：返回指定整数索引处的指定行字符串顶点（作为点几何体数据类型）。索引从1开始。如果给定索引小于1或大于集合中的元素总数，则返回null。
- ST_Geometries：返回指定集合中的几何体数组。如果指定的几何体不是多几何体，则返回一个元素数组。如果指定的几何体为空，则返回null。
- ST_InteriorRingN：返回指定索引处的内部环元素（索引从1 开始）。如果给定索引小于1或大于指定几何体中的内部环总数，则返回null。如果指定的几何体不是多边形，则引发错误。
- ST_NumPoints：返回几何体中的点数（类型为 bigint）。
- ST_IsRing：当且仅当line类型闭合且简单时返回TRUE（类型Boolean）。
- ST_StartPoint：当且仅当line类型闭合且简单时返回TRUE（类型Boolean）。
- simplify_geometry：使用Ramer-Douglas-Peucker算法返回的GEOMETRY数据类型是指定GEOMETRY的简化版本。避免创建无效的派生几何体（特别是多边形）。
- ST_EndPoint：返回line GEOMETRY数据类型中point GEOMETRY数据类型的最后一个点。
- ST_Points：从指定的线串几何体对象返回点数组。
- ST_X：返回点的X坐标 （类型为DOUBLE）。
- ST_Y：返回点的Y坐标 （类型为DOUBLE）。
- ST_GeometryType：以varchar形式返回几何体的类型。

ST_XMax

```
ST_XMax(g1)
```

- 命令说明：返回g1的X坐标上的最大值。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_XMax(ST_GeomFromText('POINT (1.5 2.5)'));
+---------------------------------------------+
| ST_XMax(ST_GeomFromText('POINT (1.5 2.5)')) |
+---------------------------------------------+
|                         1.5                 |
```

ST_YMax

```
ST_YMax(g1)
```

- 命令说明：返回g1的Y坐标上的最大值。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_YMax(ST_GeomFromText('POINT (1.5 2.5)'));
+---------------------------------------------+| ST_YMax(ST_GeomFromText('POINT (1.5 2.5)')) |
+---------------------------------------------+
|                         2.5                 |
```

ST_XMin

```
ST_XMin(g1)
```

- 命令说明：返回g1的X坐标上的最小值。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_XMin(ST_GeomFromText('MULTIPOINT (1 2, 2 4, 3 6, 4 8)'));
+-------------------------------------------------------------+
| ST_XMin(ST_GeomFromText('MULTIPOINT (1 2, 2 4, 3 6, 4 8)')) |
+-------------------------------------------------------------+
|                         1                                   |
```

ST_YMin

```
ST_YMin(g1)
```

- 命令说明：返回g1的Y坐标上的最小值。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_XMin(ST_GeomFromText('MULTIPOINT (1 2, 2 4, 3 6, 4 8)'));
+-------------------------------------------------------------+
| ST_XMin(ST_GeomFromText('MULTIPOINT (1 2, 2 4, 3 6, 4 8)')) |
+-------------------------------------------------------------+
|                          2                                  |
```

ST_Distance

```
ST_Distance(g1, g2)
```

- 命令说明：返回g1、g2之间的直线距离。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_Distance(ST_Point(1,1), ST_Point(2,2));
+-------------------------------------------+
| ST_Distance(ST_Point(1,1), ST_Point(2,2)) |
+-------------------------------------------+
|                        1.4142135623730951 |
```

ST_Distance_Sphere

```
ST_Distance_Sphere(g1, g2 [, radius])
```

- 命令说明：返回g1、g2之间的球面距离，可以指定球的半径radius，radius默认值为6370986米。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_Distance_Sphere(point(1,1), point(2,2));
+---------------------------------------------+
| ST_Distance_Sphere(point(1,1), point(2,2)) |
+---------------------------------------------+
|                         157225.08654191086 |
```

ST_Area

```
ST_Area(g1)
```

- 命令说明：返回g1的欧几里得2维空间值。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_Area(ST_GeometryFromText('POLYGON ((2 2, 2 6, 6 6, 6 2))'));
+-----------------------------------------------------------------------+
| ST_Area(ST_GeometryFromText('POLYGON ((2 2, 2 6, 6 6, 6 2))'))        |
+-----------------------------------------------------------------------+
|                        16.0                                           |
```

ST_Centroid

```
ST_Centroid(g1)
```

- 命令说明：返回g1的中心点对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_Centroid(ST_GeometryFromText('POINT (3 5)')));
+-------------------------------------------------------------------+
| ST_AsText(ST_Centroid(ST_GeometryFromText('POINT (3 5)')))        |
+-------------------------------------------------------------------+
|                       POINT (3 5)                                 |
```

ST_ConvexHull

```
ST_ConvexHull(g1)
```

- 命令说明：返回g1的中最小化的凸geometry对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_ConvexHull(ST_GeometryFromText('LINESTRING (20 20, 30 30)')));
+-----------------------------------------------------------------------------+
| ST_AsText(ST_ConvexHull(ST_GeometryFromText('LINESTRING (20 20, 30 30)'))) |
+-----------------------------------------------------------------------------+
| POLYGON ((1 1, 5 1, 6 6, 1 1))                                              |
+-----------------------------------------------------------------------------+
```

ST_CoordDim

```
ST_CoordDim(g1)
```

- 命令说明：返回g1的中坐标尺寸。
- 返回值类型：TINYINT。
- 示例：

```
SELECT ST_CoordDim(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))'));
+-----------------------------------------------------------------------+
| ST_CoordDim(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))')) |
+-----------------------------------------------------------------------+
| 2                                                                     |
+-----------------------------------------------------------------------+
```

ST_Dimension

```
ST_Dimension(g1)
```

- 命令说明：返回g1中固有的dimension，肯定小于或者等于coordinate dimension。
- 返回值类型：TINYINT。
- 示例：

```
SELECT ST_Dimension(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))'));
+-----------------------------------------------------------------------+
| ST_Dimension(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))')) |
+-----------------------------------------------------------------------+
| 2                                                                     |
+-----------------------------------------------------------------------+
```

ST_IsClosed

```
ST_IsClosed(g1)
```

- 命令说明：如果LineString或者Muti-LineString的start和end是一致的，返回true。
- 返回值类型：BOOLEAN。
- 示例：

```
SELECT ST_IsClosed(ST_GeometryFromText('LINESTRING (1 1, 2 2, 1 3, 1 1)'));
+-----------------------------------------------------------------------+
| ST_IsClosed(ST_GeometryFromText('LINESTRING (1 1, 2 2, 1 3, 1 1)')) |
+-----------------------------------------------------------------------+
| true                                                                  |
+-----------------------------------------------------------------------+
```

ST_IsEmpty

```
ST_IsEmpty(g1)
```

- 命令说明：如果g1是一个空的geometrycollection，或polygon，point，返回true。
- 返回值类型：BOOLEAN。
- 示例：

```
SELECT ST_IsEmpty(ST_GeometryFromText('POINT (1.5 2.5)'));
+-----------------------------------------------------+
| ST_IsEmpty(ST_GeometryFromText('POINT (1.5 2.5)')) |
+-----------------------------------------------------+
| false                                               |
+-----------------------------------------------------+
```

ST_IsValid

```
ST_IsValid(g1)
```

- 命令说明：g1是不是合法的格式。
- 返回值类型：BOOLEAN。
- 示例：

```
SELECT ST_IsValid(ST_GeometryFromText('MULTIPOINT (1 2, 3 4)');
+-------------------------------------------------------------+
| ST_IsValid(ST_GeometryFromText('MULTIPOINT (1 2, 3 4)')     |
+-------------------------------------------------------------+
|          true                                               |
```

geometry_invalid_reason

```
ST_IsValid(g1)
```

- 命令说明：返回g1是不是合法的格式的原因。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT geometry_invalid_reason(ST_GeometryFromText('POLYGON ((0 0, 1 1, 0 1, 1 0, 0 0))'));
+-------------------------------------------------------------------------------------------+
| geometry_invalid_reason(ST_GeometryFromText('POLYGON ((0 0, 1 1, 0 1, 1 0, 0 0))'))    |
+-------------------------------------------------------------------------------------------+
|          Intersecting or overlapping segments at or near (1.0 0.0) and (1.0 1.0)
|
```

ST_Length

```
ST_IsValid(g1)
```

- 命令说明：如果输入是LineString或者Multi-LineString的长度（计算是基于Euclidean）； 如果输入是GEOGRAPHY类型的格式，返回的是球面上的greate-circle的长度。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_Length(ST_GeometryFromText('LINESTRING EMPTY'));
+-----------------------------------------------------+
| ST_Length(ST_GeometryFromText('LINESTRING EMPTY'))   |
+-----------------------------------------------------+
|          0.0                                        |
```

line_locate_point

```
line_locate_point(g1, g2)
```

- 命令说明：返回一个float值，标识的是给定点在线上的比率。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT line_locate_point(ST_GeometryFromText('LINESTRING (0 0, 0 1)'), ST_Point(0, 0.2));
+-------------------------------------------------------------------------------------+
| line_locate_point(ST_GeometryFromText('LINESTRING (0 0, 0 1)'), ST_Point(0, 0.2))   |
+-------------------------------------------------------------------------------------+
|                                      0.2                                            |
```

line_interpolate_point

```
line_interpolate_point(g1, d)
```

- 命令说明：给定一个比率和线，返回这个比率的点。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT line_interpolate_point(ST_GeometryFromText('LINESTRING EMPTY'), 0.5);
+-----------------------------------------------------------------------------------------+
| line_interpolate_point(ST_GeometryFromText('LINESTRING EMPTY'), 0.5)               |
+-----------------------------------------------------------------------------------------+
|                                    null                                                 |
```

line_interpolate_points

```
line_interpolate_points(g1, d)
```

- 命令说明：给定一个比率和线，返回按照这个比率分割的所有点。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT transform(line_interpolate_points(ST_GeometryFromText('LINESTRING (0 0, 1 1, 10 10)'), 0.4), x -> ST_AsText(x));
+-----------------------------------------------------------------------------------------------------------------+
| transform(line_interpolate_points(ST_GeometryFromText('LINESTRING (0 0, 1 1, 10 10)'), 0.4), x -> ST_AsText(x)) |
+-----------------------------------------------------------------------------------------------------------------+
|       POINT(4.000000000000001 4.000000000000001)      POINT(8,8)                                                |
```

ST_NumInteriorRing

```
ST_NumInteriorRing(g1)
```

- 命令说明：返回polygon的内部环的集合的基数。
- 返回值类型：BIGINT。
- 示例：

```
SELECT ST_NumInteriorRing(ST_GeometryFromText('POLYGON ((0 0, 0 5, 5 5, 5 0, 0 0))'));
+--------------------------------------------------------------------------+
| ST_NumInteriorRing(ST_GeometryFromText('POLYGON ((0 0, 0 5, 5 5, 5 0, 0 0))')) |
+--------------------------------------------------------------------------+
|                                  0                                       |
```

ST_InteriorRings

```
ST_InteriorRings(g1)
```

- 命令说明：返回在指定几何体中找到的所有内部环的几何体数组，如果多边形没有内部环，则返回空数组。如果指定的几何体为空，则返回 null。如果指定的几何体不是多边形，则会引发错误。
- 返回值类型：Array[GEOMETRY[GEOMETRY]。
- 示例：

```
SELECT ST_InteriorRings(ST_GeometryFromText('POLYGON EMPTY'));
+-----------------------------------------------------------------------+
| ST_InteriorRings(ST_GeometryFromText('POLYGON EMPTY'))                |
+-----------------------------------------------------------------------+
|                                  null                                 |
```

ST_NumGeometries

```
ST_NumGeometries(g1)
```

- 命令说明：以整数形式返回集合中的几何体数。如果几何体是几何体的集合 （例如GEOMETRYCOLLECTION或MULTI* 对象），则返回几何体的数量。单个几何体返回 1；空几何体返回0。对象中的空几何体计为一个几何体。
- 返回值类型：int。
- 示例：

```
SELECT ST_NumGeometries(ST_GeometryFromText('POINT (1 2)'));
+-----------------------------------------------------------------------+
| ST_NumGeometries(ST_GeometryFromText('POINT (1 2)'))                  |
+-----------------------------------------------------------------------+
|                                  1                                    |
```

ST_GeometryN

```
ST_GeometryN(g, i)
```

- 命令说明：返回位于指定整数索引的 GEOMETRY元素（作为GEOMETRY数据类型）。索引从1开始。如果指定的几何体是几何体集合（例如，GEOMETRYCOLLECTION或MULTI*对象），则将返回指定索引处的几何体。如果指定的索引小于1或大于集合中的元素总数，则返回null。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_ASText(ST_GeometryN(ST_GeometryFromText('POINT (1 2)'), 1));
+-------------------------------------------------------------------------------+
|   ST_ASText(ST_GeometryN(ST_GeometryFromText('POINT (1 2)'), 1))              |
+-------------------------------------------------------------------------------+
|                 POINT (1 2)                                                   |
```

ST_PointN

```
ST_PointN(g, i)
```

- 命令说明：返回指定整数索引处的指定行字符串顶点（作为点几何体数据类型）。索引从1开始。如果给定索引小于1或大于集合中的元素总数，则返回null。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_ASText(ST_PointN(ST_GeometryFromText('LINESTRING(1 2, 3 4, 5 6, 7 8)'), 3));
+-------------------------------------------------------------------------------+
|   ST_ASText(ST_PointN(ST_GeometryFromText('LINESTRING(1 2, 3 4, 5 6, 7 8)'), 3))      |
+-------------------------------------------------------------------------------+
|                 POINT (5 6)                                                   |
```

ST_Geometries

```
ST_Geometries(g)
```

- 命令说明：返回指定集合中的几何体数组。如果指定的几何体不是多几何体，则返回一个元素数组。如果指定的几何体为空，则返回null。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT transform(ST_Geometries(ST_GeometryFromText('POINT (1 5)')), x -> ST_ASText(x))  +
*-------------------------------------------------------------------------------+
|   transform(ST_Geometries(ST_GeometryFromText('POINT (1 5)')), x -> ST_ASText(x))     |
+-------------------------------------------------------------------------------+
|                 POINT (1 5)                                                   |
```

ST_InteriorRingN

```
ST_InteriorRingN(g, d)
```

- 命令说明：返回指定索引处的内部环元素（索引从1开始）。如果给定索引小于1或大于指定几何体中的内部环总数，则返回null。如果指定的几何体不是多边形，则引发错误。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_ASText(ST_InteriorRingN(ST_GeometryFromText('POLYGON ((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2 2, 2 1, 1 1))'), 1))  +
*-----------------------------------------------------------------------------------------------------------------------+
|    ST_ASText(ST_InteriorRingN(ST_GeometryFromText('POLYGON ((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2 2, 2 1, 1 1))'), 1))    |
+-----------------------------------------------------------------------------------------------------------------------+
|                LINESTRING (1 1, 1 2, 2 2, 2 1, 1 1)                                                                   |
```

ST_NumPoints

```
 ST_NumPoints(g)
```

- 命令说明：返回几何体中的点数（类型为bigint）。
- 返回值类型：BIGINT。
- 示例：

```
SELECT  ST_NumPoints(ST_GeometryFromText('POINT (1 2)'))                  +
*-------------------------------------------------------------------------+
|     ST_NumPoints(ST_GeometryFromText('POINT (1 2)'))                    |
+-------------------------------------------------------------------------+
|                1                                                        |
```

ST_IsRing

```
ST_IsRing(g)
```

- 命令说明：当且仅当line类型闭合且简单时返回TRUE（类型boolean）。
- 返回值类型：BOOLEAN。
- 示例：

```
SELECT  ST_IsRing(ST_GeometryFromText('LINESTRING (8 4, 4 8)'))           +
*-------------------------------------------------------------------------+
|     ST_IsRing(ST_GeometryFromText('LINESTRING (8 4, 4 8)'))             |
+-------------------------------------------------------------------------+
|                false                                                    |
```

ST_StartPoint

```
ST_StartPoint(g)
```

- 命令说明：当且仅当line类型闭合且简单时返回TRUE（类型boolean）。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT  ST_AsText(ST_StartPoint(ST_GeometryFromText('LINESTRING (8 4, 4 8, 5 6)')))   +
*-------------------------------------------------------------------------------------+
|     ST_AsText(ST_StartPoint(ST_GeometryFromText('LINESTRING (8 4, 4 8, 5 6)')))     |
+-------------------------------------------------------------------------------------+
|                POINT (5 6)                                                          |
```

simplify_geometry

```
simplify_geometry(g, d)
```

- 命令说明：使用Ramer-Douglas-Peucker算法返回的GEOMETRY数据类型是指定GEOMETRY的简化版本。避免创建无效的派生几何体（特别是多边形）。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT  ST_AsText(simplify_geometry(ST_GeometryFromText('POLYGON ((1 0, 2 1, 3 1, 3 1, 4 1, 1 0))'), 1.5))  +
*-----------------------------------------------------------------------------------------------------------+
|     ST_AsText(simplify_geometry(ST_GeometryFromText('POLYGON ((1 0, 2 1, 3 1, 3 1, 4 1, 1 0))'), 1.5))    |
+-----------------------------------------------------------------------------------------------------------+
|                  POLYGON ((1 0, 4 1, 2 1, 1 0))                                                           |
```

ST_EndPoint

```
ST_EndPoint(g)
```

- 命令说明：返回line geometry数据类型中point geometry数据类型的最后一个点。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_StartPoint(ST_GeometryFromText('LINESTRING (8 4, 4 8, 5 6)')))  +
*-------------------------------------------------------------------------------------+
| ST_AsText(ST_StartPoint(ST_GeometryFromText('LINESTRING (8 4, 4 8, 5 6)')))       |
+-------------------------------------------------------------------------------------+
|                  POINT (8 4)                                                        |
```

ST_Points

```
ST_Points(g)
```

- 命令说明：从指定的线串几何体对象返回点数组。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT transform(ST_Points(ST_GeometryFromText('POINT (0 0)')), x -> ST_AsText(x))  +
*-----------------------------------------------------------------------------------+
| transform(ST_Points(ST_GeometryFromText('POINT (0 0)')), x -> ST_AsText(x))       |
+-----------------------------------------------------------------------------------+
|                  0 0                                                              |
```

ST_X

```
ST_X(g)
```

- 命令说明：返回点的X坐标 （类型为double）。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_X(ST_GeometryFromText('POINT (1 2)'))  +
*-----------------------------------------------+
|ST_X(ST_GeometryFromText('POINT (1 2)'))        |
+-----------------------------------------------+
|               1.0                             |
```

ST_Y

```
ST_Y(g)
```

- 命令说明：返回点的Y坐标 （类型为double）。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT ST_Y(ST_GeometryFromText('POINT (1 2)'))  +
*-----------------------------------------------+
|ST_Y(ST_GeometryFromText('POINT (1 2)'))       |
+-----------------------------------------------+
|               2.0                            |
```

ST_GeometryType

```
ST_Within(g1)
```

- 命令说明：以varchar形式返回几何体的类型。
- 返回值类型：VARCHAR。
- 示例：

```
SELECT ST_GeometryType(ST_Point(1, 4))  +
*--------------------------------------+
|ST_GeometryType(ST_Point(1, 4))       |
+--------------------------------------+
|                false                 |
```

4.6.4. 空间构造函数

使用空间构造函数可获取point、line或polygon geometry数据类型的二进制表示。您也可以使用这些函数来将二进制数据转换为文本数据，以及获取以已知文本（WKT）格式表示的geometry数据的二进制值。

- ST_Point：先将两个DOUBLE类型的数值x和y进行除法运算 `x/y` ，然后将 `x/y` 的结果值为坐标构造一个POINT类型的值。
- ST_AsText：返回WKT（Well-Known Text）格式。
- ST_GeometryFromText或ST_GeomFromText：使用WKT格式的字符串构造GEOMETRY值。
- ST_IsValidWKT：返回WKT的是否是正确的WKT格式的文本。
- ST_LineFromText：返回WKT的文本对应的LineString的对象。
- ST_PointFromText：返回WKT的文本对应的PointString的对象。
- ST_PolygonFromText：返回WKT的文本对应的PolygonString的对象。
- ST_LineString：返回g1，g2，g3一组点的LineString。
- ST_MultiPoint：返回g1，g2，g3等一组点组成的多点GEOMETRY对象。
- ST_GeomFromBinary：返回wkb类型对象的GEOMETRY对象。
- geometry_from_hadoop_shape：返回spatial framework for hadoop 、类型对象的GEOMETRY对象。
- to_spherical_geography：从GEOMETRY对象转换成SphericalGeography对象，也就是从2D的标识到3D的标识。
- to_geometry：从SphericalGeography对象转换成GEOMETRY对象，其实每个Geography都是一个合理的geometry的对象。
- ST_AsBinary：返回g1对应的WKB格式内容。
- ST_Buffer：返回离g1的距离小于或等于d对应的所有点。

ST_Point

```
ST_Point(x, y)
```

- 命令说明：先将两个DOUBLE类型的数值x和y进行除法运算 `x/y` ，然后将 `x/y` 的结果值为坐标构造一个POINT类型的值。
- 返回值类型：GEOMETRY类型的POINT对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_Point(1,1);
+----------------------+
| ST_Point(1,1)        |
+----------------------+
|        ?      ?    |
```

ST_AsText

```
ST_AsText(geometry)
```

- 命令说明：将每个指定的GEOMETRY数据类型转换为文本。
- 返回值类型：返回GEOMETRY的WKT（Well-Known Text）格式。
- 示例：

```
SELECT ST_AsText(ST_Point(1,1));
+--------------------------+
| ST_AsText(ST_Point(1,1)) |
+--------------------------+
| POINT (1 1)              |
```

ST_GeometryFromText或ST_GeomFromText

```
ST_GeometryFromText(wkt)
```

- 命令说明：使用WKT格式的字符串构造GEOMETRY值。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_GeometryFromText('Point(1 1)');
+----------------------------------+
| ST_GeometryFromText('Point(1 1)') |
+----------------------------------+
|        ?       ?                 |
```

```
SELECT ST_AsText(ST_GeometryFromText('Point(1 1)'));
+-----------------------------------------------+
| ST_AsText(ST_GeometryFromText('Point(1 1)')) |
+-----------------------------------------------+
| POINT (1 1)                                   |
```

ST_IsValidWKT

```
ST_IsValidWKT(wkt)
```

- 命令说明：返回WKT的是否是正确的WKT格式的文本。
- 返回值类型：BOOLEAN。
- 示例：

```
SELECT ST_IsValidWKT('MULTIPOINT (1 2, 2 4, 3 6, 4 8)');
+-----------------------------------------------------------+
|        ST_IsValidWKT('MULTIPOINT (1 2, 2 4, 3 6, 4 8)')     |
+-----------------------------------------------------------+
|                        true                                |
```

ST_LineFromText

```
ST_LineFromText(wkt)
```

- 命令说明：返回WKT的文本对应的LineString的对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_LineFromText('LINESTRING (1 1, 2 2, 1 3)');
+-----------------------------------------------------------+
|        ST_LineFromText('LINESTRING (1 1, 2 2, 1 3)')        |
+-----------------------------------------------------------+
|                         ?       ?                          |
```

```
SELECT ST_AsText(ST_LineFromText('LINESTRING (1 1, 2 2, 1 3)'));
+-----------------------------------------------------------------------+
|        ST_AsText(ST_LineFromText('LINESTRING (1 1, 2 2, 1 3)'))       |
+-----------------------------------------------------------------------+
|                        LINESTRING (1 1, 2 2, 1 3)                     |
```

ST_PointFromText

```
ST_PointFromText(wkt)
```

- 命令说明：返回WKT的文本对应的PointString的对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_PointFromText('POINT (1 2)');
+-----------------------------------------------------------+
|        ST_PointFromText('POINT (1 2)')                     |
+-----------------------------------------------------------+
|                         ?       ?                          |
```

```
SELECT ST_AsText(ST_PointFromText('POINT (1 2)'));
+-----------------------------------------------------------+
|        ST_AsText(ST_PointFromText('POINT (1 2)'))          |
+-----------------------------------------------------------+
|                       POINT (1 2)                          |
```

ST_PolygonFromText

```
ST_PolygonFromText(wkt)
```

- 命令说明：返回WKT的文本对应的PolygonString的对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_PolygonFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))');
+-------------------------------------------------------------+
|        ST_PolygonFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))') |
+-------------------------------------------------------------+
|                          ?       ?                          |
```

```
SELECT ST_AsText(ST_PolygonFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))'));
+-----------------------------------------------------------------------+
|        ST_AsText(ST_PolygonFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))')) |
+-----------------------------------------------------------------------+
|                        POLYGON ((1 1, 1 4, 4 4, 4 1))                 |
```

ST_LineString

```
ST_LineString(array(g1,g2...))
```

- 命令说明：返回g1，g2，g3一组点的LineString。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]));
+-----------------------------------------------------------------------+
| ST_AsText(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]))         |
+-----------------------------------------------------------------------+
|           LINESTRING (1 2, 3 4)                                      |
```

ST_MultiPoint

```
ST_MultiPoint(array(g1,g2...))
```

- 命令说明：返回g1，g2，g3等一组点组成的多点geometry对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_MultiPoint(array[ST_GeometryFromText('POINT(1 2)'), ST_GeometryFromText('POINT (3 4)')]));
+-----------------------------------------------------------------------------------------------------------------+
|  ST_AsText(ST_MultiPoint(array[ST_GeometryFromText('POINT(1 2)'), ST_GeometryFromText('POINT (3 4)')]))  |
+-----------------------------------------------------------------------------------------------------------------+
|          MULTIPOINT ((1 2), (3 4))                                                                             |
```

ST_GeomFromBinary

```
ST_GeomFromBinary(wkb)
```

- 命令说明：返回WKB类型对象的geometry对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_GeomFromBinary(ST_AsBinary(ST_GeometryFromText('MULTIPOINT ((1 2), (3 4)')))));
+-----------------------------------------------------------------------------------------------------------------+
|  ST_AsText(ST_GeomFromBinary(ST_AsBinary(ST_GeometryFromText('MULTIPOINT ((1 2), (3 4)'))))                     |
+-----------------------------------------------------------------------------------------------------------------+
|          MULTIPOINT ((1 2), (3 4))                                                                             |
```

geometry_from_hadoop_shape

```
geometry_from_hadoop_shape(wkb)
```

- 命令说明：返回spatial framework for hadoop 、类型对象的GEOMETRY对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(geometry_from_hadoop_shape(from_hex('00000000101000000000000000F03F0000000000000040')));
+-----------------------------------------------------------------------------------------------------+
| ST_AsText(geometry_from_hadoop_shape(from_hex('00000000101000000000000000F03F0000000000000040'))) |
+-----------------------------------------------------------------------------------------------------+
|          MULTIPOINT ((1 2), (3 4))                                                                 |
```

to_spherical_geography

```
to_spherical_geography(g1)
```

- 命令说明：从GEOMETRY对象转换成SphericalGeography对象，也就是从2D的标识到3D的标识。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT to_spherical_geography(ST_Point(-71.0882, 42.3607));
+-----------------------------------------------------------+
| to_spherical_geography(ST_Point(-71.0882, 42.3607))       |
+-----------------------------------------------------------+
|          ?       ?                                        |
```

to_geometry

```
to_geometry(g1)
```

- 命令说明：从SphericalGeography对象转换成GEOMETRY对象，其实每个Geography都是一个合理的GEOMETRY的对象。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT to_geometry(to_spherical_geography(ST_Point(61.56, -158.54)));
+-------------------------------------------------------------------+
| SELECT to_geometry(to_spherical_geography(ST_Point(61.56, -158.54))) |
+-------------------------------------------------------------------+
|           ?       ?                                               |
```

ST_AsBinary

```
ST_AsBinary(g1)
```

- 命令说明：返回g1对应的WKB格式内容。
- 返回值类型：VARBINARY。
- 示例：

```
SELECT ST_AsText(ST_GeomFromBinary(ST_AsBinary(ST_GeometryFromText('MULTIPOINT ((1 2), (3 4)')))));
+-----------------------------------------------------------------------------------------------------------------------+
|  ST_AsText(ST_GeomFromBinary(ST_AsBinary(ST_GeometryFromText('MULTIPOINT ((1 2), (3 4)')))))                          |
+-----------------------------------------------------------------------------------------------------------------------+
|          MULTIPOINT ((1 2), (3 4))                                                                                   |
```

ST_Buffer

```
ST_Buffer(g1, d)
```

- 命令说明：返回离g1的举例小于或等于d对应的所有点。
- 返回值类型：GEOMETRY类型的对象，直接通过SELECT查询函数结果将显示乱码。
- 示例：

```
SELECT ST_AsText(ST_Buffer(ST_Point(0, 0), 0.5));
+-----------------------------------------------------------------------+
|  ST_AsText(ST_Buffer(ST_Point(0, 0), 0.5))                            |
+-----------------------------------------------------------------------+
|          POLYGON ((0.5 0, 0.4989294616193014 0.03270156461507146,
0.49572243068690486 0.0652630961100257, 0.4903926402016149 0.09754516100806403,
0.4829629131445338 0.12940952255126026, 0.47346506474755257 0.16071973265158065,
0.46193976625564315 0.19134171618254472, 0.4484363707663439 0.22114434510950046,
0.43301270189221913 0.2499999999999998, 0.41573480615127245 0.2777851165098009,
0.39667667014561747 0.30438071450436016, 0.3759199037394886 0.32967290755003426,
0.3535533905932737 0.3535533905932736, 0.32967290755003437 0.3759199037394886,
0.3043807145043603 0.39667667014561747, 0.2777851165098011 0.4157348061512725,
0.24999999999999997 0.43301270189221924, 0.22114434510950062 0.4484363707663441,
0.19134171618254486 0.4619397662556433, 0.16071973265158077 0.4734650647475528,
0.12940952255126037 0.48296291314453416, 0.09754516100806412 0.4903926402016152,
0.06526309611002579 0.4957224306869052, 0.03270156461507153 0.49892946161930174,
0 0.5, -0.03270156461507146 0.4989294616193014, -0.0652630961100257
0.49572243068690486, -0.09754516100806403 0.4903926402016149, -0.12940952255126026
0.4829629131445338, -0.16071973265158065 0.47346506474755257, -0.19134171618254472
0.46193976625564315, -0.22114434510950046 0.4484363707663439, -0.2499999999999998
0.43301270189221913, -0.2777851165098009 0.41573480615127245, -0.30438071450436016
0.39667667014561747, -0.32967290755003426 0.3759199037394886, -0.3535533905932736
0.3535533905932737, -0.3759199037394886 0.32967290755003437, -0.39667667014561747
0.3043807145043603, -0.4157348061512725 0.2777851165098011, -0.43301270189221924
0.24999999999999997, -0.4484363707663441 0.22114434510950062, -0.4619397662556433
0.19134171618254486, -0.4734650647475528 0.16071973265158077, -0.48296291314453416
0.12940952255126037, -0.4903926402016152 0.09754516100806412, -0.4957224306869052
0.06526309611002579, -0.49892946161930174 0.03270156461507153, -0.5 0,
-0.4989294616193014 -0.03270156461507146, -0.49572243068690486 -0.0652630961100257,
-0.4903926402016149 -0.09754516100806403, -0.4829629131445338 -0.12940952255126026,
-0.47346506474755257 -0.16071973265158065, -0.46193976625564315
-0.19134171618254472, -0.4484363707663439 -0.22114434510950046,
-0.43301270189221913 -0.2499999999999998, -0.41573480615127245 -0.2777851165098009,
-0.39667667014561747 -0.30438071450436016, -0.3759199037394886 -0.32967290755003426,
-0.3535533905932737 -0.3535533905932736, -0.32967290755003437 -0.3759199037394886,
-0.3043807145043603 -0.39667667014561747, -0.2777851165098011 -0.4157348061512725,
-0.24999999999999997 -0.43301270189221924, -0.22114434510950062 -0.4484363707663441,
-0.19134171618254486 -0.4619397662556433, -0.16071973265158077 -0.4734650647475528,
-0.12940952255126037 -0.48296291314453416, -0.09754516100806412 -0.4903926402016152,
-0.06526309611002579 -0.4957224306869052, -0.03270156461507153 -0.49892946161930174,
0 -0.5, 0.03270156461507146 -0.4989294616193014, 0.0652630961100257
-0.49572243068690486, 0.09754516100806403 -0.4903926402016149, 0.12940952255126026
-0.4829629131445338, 0.16071973265158065 -0.47346506474755257, 0.19134171618254472
-0.46193976625564315, 0.22114434510950046 -0.4484363707663439, 0.2499999999999998
-0.43301270189221913, 0.2777851165098009 -0.41573480615127245, 0.30438071450436016
-0.39667667014561747, 0.32967290755003426 -0.3759199037394886, 0.3535533905932736
-0.3535533905932737, 0.3759199037394886 -0.32967290755003437, 0.39667667014561747
-0.3043807145043603, 0.4157348061512725 -0.2777851165098011, 0.43301270189221924
-0.24999999999999997, 0.4484363707663441 -0.22114434510950062, 0.4619397662556433
-0.19134171618254486, 0.4734650647475528 -0.16071973265158077, 0.48296291314453416
-0.12940952255126037, 0.4903926402016152 -0.09754516100806412, 0.4957224306869052
-0.06526309611002579, 0.49892946161930174 -0.03270156461507153, 0.5 0))                |
```

4.7. JSON函数

本文介绍AnalyticDB MySQL版集群支持的JSON函数。

- JSON _ARRAY_CONTAINS：判断JSON中是否包含 `value` 指定的值。
- JSON_ARRAY_LENGTH：返回JSON数组的长度。
- JSON _EXTRACT：从JSON中返回 `jsonpath` 指定的值。
- JSON_SIZE：返回JSON的大小。
- JSON_KEYS：获取JSON在指定路径下的所有键值。
- JSON_UNQUOTE：去除 `json_value` 的双引号并将 `json_value` 中的部分转义符进行转义后，返回处理结果。
- JSON_CONTAINS：判断指定Path中是否包含 `candidate` 的值；若未指定Path，则判断Target中是否包含 `candidate` 指定的值。
- JSON_CONTAINS_PATH：判断JSON中是否包含Path列表中的路径。

JSON _ARRAY_CONTAINS

```
json_array_contains(json, value)
```

- 命令说明：判断JSON数组中是否包含 `value` 指定的值。
- 输入值类型： `value` 可以是数值、字符串类型或BOOLEAN类型。
- 返回值类型：BOOLEAN。
- 示例：

```
SELECT json_array_contains('[1, 2, 3]', 2);
```

返回结果如下：

```
+--------------------------------------+
| json_array_contains('[1, 2, 3]', 2) |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

JSON_ARRAY_LENGTH

```
json_array_length(json)
```

- 命令说明：返回JSON数组的长度。
- 输入值类型：字符串类型或JSON类型。
- 返回值类型：BIGINT。
- 示例：

```
SELECT json_array_length('[1, 2, 3]');
```

返回结果如下：

```
+---------------------------------+
| json_array_length('[1, 2, 3]') |
+---------------------------------+
|                               3 |
+---------------------------------+
```

JSON _EXTRACT

> 注意 JSON_EXTRACT函数的返回值，同JSON类型的列一样，均不支持`ORDER BY`。

```
json_extract(json, jsonpath)
```

- 命令说明：从JSON中返回`jsonpath`指定的值。
- 输入值类型：字符串类型或JSON类型。
- 返回值类型：JSON。
- 示例：

```
SELECT json_extract('[10, 20, [30, 40]]', '$.1');
```

返回结果如下：

```
+-------------------------------------------+
| json_extract('[10, 20, [30, 40]]', '$.1') |
+-------------------------------------------+
| 20                                        |
+-------------------------------------------+
```

JSON_SIZE

```
json_size(json, jsonpath)
```

- 命令说明：从JSON中返回`jsonpath`指定JSON对象或JSON数组的大小。

> 说明 若`jsonpath`指向的不是JSON对象或者JSON数组时，返回0。

- 输入值类型：字符串类型或JSON类型。
- 返回值类型：BIGINT。
- 示例：
  - `jsonpath`指向的是JSON对象，语句如下：

```
SELECT json_size('{"x":{"a":1, "b": 2}}', '$.x') as result;
```

返回结果如下：

```
+--------+
| result |
+--------+
|      2 |
+--------+
```

- `jsonpath` 指向的不是JSON对象或者JSON数组，语句如下：

```
SELECT json_size('{"x": {"a": 1, "b": 2}}', '$.x.a') as result;
```

返回结果如下：

```
+--------+
| result |
+--------+
|      0 |
+--------+
```

JSON_KEYS

```
json_keys(json, jsonpath)
json_keys(json)
```

- 命令说明
  - 若指定了 `jsonpath` ，表示获取JSON在指定路径下的所有键。
  - 若未指定 `jsonpath` ，表示获取根路径（即 `jsonpath='$'` ）下的所有键。
- 输入值类型：仅支持输入JSON类型的参数。

  您可以通过如下方式构造JSON数据：
  - 直接使用JSON数据。例如 `json '{"a": 1, "b": {"c": 30}}'` 。
  - 通过CAST函数将字符串进行显式转换为JSON数据。例如 `CAST('{"a": 1, "b": {"c": 30}}' AS json)` 。
- 返回值类型：JSON ARRAY。
- 示例：
  - 返回 `$.b` 路径下的所有键，语句如下：

```
SELECT json_keys(CAST('{"a": 1, "b": {"c": 30}}' AS json),'$.b');
```

返回结果如下：

```
+-----------------------------------------------------------+
| json_keys(CAST('{"a": 1, "b": {"c": 30}}' AS json),'$.b') |
+-----------------------------------------------------------+
| ["c"]                                                     |
+-----------------------------------------------------------+
```

  - 返回根路径下的所有键，语句如下：

```
SELECT JSON_KEYS(json '{"a": 1, "b": {"c": 30}}');
```

返回结果如下：

```
+-------------------------------------------+
| JSON_KEYS(json '{"a": 1, "b": {"c": 30}}') |
+-------------------------------------------+
| ["a","b"]                                 |
+-------------------------------------------+
```

JSON_UNQUOTE

```
json_unquote(json_value)
```

> **说明** AnalyticDB MySQL版集群的内核版本需为V3.1.5.0或以上版本才支持该函数。
>
> 如何查看集群版本，请参见查看版本。
>
> 如需升级版本，请提交工单联系技术支持。

- 命令说明：去除 `json_value` 的双引号并将其中的部分转义符进行转义后，返回处理结果。

  AnalyticDB MySQL版不会判断 `json_value` 的合法性，即无论 `json_value` 是否符合JSON语法都会按上述逻辑进行处理。

  支持的转义符如下表。

转义前	转义后
`\"`	双引号（ `"` ）。
`\b`	退格键。
`\f`	换页符。
`\n`	换行符。

转义前	转义后
`\r`	回车符。
`\t`	Tab键。
`\\`	反斜线（ `\` ）。
`\uXXXX`	UTF-8字符表示。

- 输入值类型：字符串类型或JSON类型。
- 返回值类型：VARCHAR。
- 示例：
  - 语句如下：

    ```
    SELECT json_unquote('"abc"');
    ```

    返回结果如下：

    ```
    +-----------------------+
    | json_unquote('"abc"') |
    +-----------------------+
    | abc                   |
    +-----------------------+
    ```

  - 语句如下：

    ```
    SELECT json_unquote('"\\t\\u0032"');
    ```

    返回结果如下：

    ```
    +------------------------------+
    | json_unquote('"\\t\\u0032"') |
    +------------------------------+
    |    2                         |
    +------------------------------+
    ```

JSON_CONTAINS

```
json_contains(target, candidate[, path])
```

> **说明** AnalyticDB MySQL版集群的内核版本需为V3.1.5.0或以上版本才支持该函数。
>
> 如何查看集群版本，请参见查看版本。
>
> 如需升级版本，请提交工单联系技术支持。

- 命令说明：
  - 若指定了 `path` ，则判断指定Path中是否包含 `candidate` 的值。包含返回1，不包含返回0。
  - 若未指定 `path` ，则判断Target中是否包含 `candidate` 指定的值。包含返回1，不包含返回0。

  规则如下：
  - 若 `target` 和 `candidate` 均为PRIMITIVE类型（即NUMBER，BOOLEAN，STRING，NULL），当二者相等时，视为Target包含Candidate。
  - 若 `target` 和 `candidate` 均为ARRAY类型的JSON，当Candidate的所有元素均包含于Target的某个元素中时，视为Target包含Candidate。
  - 若 `target` 为ARRAY类型且 `candidate` 为非ARRAY类型，当Candidate包含于Target的某个元素中时，视为Target包含Candidate。
  - 若 `target` 和 `candidate` 均为OBJECT类型的JSON，当Candidate中的每个Key都包含于Target的Key中，且Candidate的Key对应的Value包含于Target中该Key对应的Value时，视为Target包含Candidate。
- 输入值类型： `target` 和 `candidate` 为JSON类型， `path` 为jsonpath。
- 返回值类型：BOOLEAN。
- 示例：
  - 语句如下：

    ```
    select json_contains(json '{"a": 1, "b": 2, "c": {"d": 4}}', json '1', '$.a') as result;
    ```

    返回结果如下：

    ```
    +--------+
    | result |
    +--------+
    |      1 |
    +--------+
    ```

- 语句如下：

```
select json_contains(json '{"a": 1, "b": 2, "c": {"d": 4}}', json '1', '$.b') as result;
```

返回结果如下：

```
+--------+
| result |
+--------+
|      0 |
+--------+
```

- 语句如下：

```
select json_contains(json '{"a": 1, "b": 2, "c": {"d": 4}}', json '{"d": 4}') as result;
```

返回结果如下：

```
+--------+
| result |
+--------+
|      0 |
+--------+
```

JSON_CONTAINS_PATH

```
json_contains_path(json, one_or_all, path[, path ...])
```

> **说明** AnalyticDB MySQL版集群的内核版本需为V3.1.5.0或以上版本才支持该函数。
> 如何查看集群版本，请参见查看版本。
> 如需升级版本，请提交工单联系技术支持。

- 命令说明：判断JSON中是否包含Path列表中的路径。
  - 当 `one_or_all` 为 `'one'` ，JSON中包含所有Path中的其中之一时，返回1，否则返回0。
  - 当 `one_or_all` 为 `'all'` ，JSON中包含所有Path路径时，返回1，否则返回0。
- 输入值类型： `json` 为JSON类型， `one_or_all` 为VARCHAR类型（为 `'one'` 或者 `'all'` ，不区分大小写）， `path` 为Jsonpath。
- 返回值类型：BOOLEAN。
- 示例：
  - 语句如下：

```
select json_contains_path(json '{"a": 1, "b": 2, "c": {"d": 4}}', 'one', '$.a', '$.e') as result;
```

返回结果如下：

```
+--------+
| result |
+--------+
|      1 |
+--------+
```

  - 语句如下：

```
select json_contains_path(json '{"a": 1, "b": 2, "c": {"d": 4}}', 'all', '$.a', '$.e') as result;
```

返回结果如下：

```
+--------+
| result |
+--------+
|      0 |
+--------+
```

4.8. 窗口函数

AnalyticDB for MySQL支持以下窗口函数。

- 聚合函数
- 排序函数
  - CUME_DIST：返回一组数值中每个值的累计分布。
  - RANK：返回数据集中每个值的排名。
  - DENSE_RANK：返回一组数值中每个数值的排名。
  - NTILE：将每个窗口分区的数据分散到桶号从1到n的n个桶中。
  - ROW_NUMBER：根据行在窗口分区内的顺序，为每行数据返回一个唯一的有序行号，行号从1开始。

- PERCENT_RANK：返回数据集中每个数据的排名百分比，其结果由 `(r - 1) / (n - 1)` 计算得出。其中r为RANK()计算的当前行排名， n为当前窗口分区内总的行数。

- 值函数
  - FIRST_VALUE：返回窗口分区第1行的值。
  - LAST_VALUE返回窗口分区最后1行的值。
  - LAG：返回窗口内距离当前行之前偏移offset后的值。
  - LEAD：返回窗口内距离当前行偏移offset后的值。
  - NTH_VALUE：返回窗口内偏移指定offset后的值，偏移量从1开始。

概述

窗口函数基于查询结果的行数据进行计算，窗口函数运行在 `HAVING` 子句之后、 `ORDER BY` 子句之前。窗口函数需要特殊的关键字 `OVER` 子句来指定窗口即触发一个窗口函数。

分析型数据库MySQL版支持三种类型的窗口函数：聚合函数、排序函数和值函数。

语法

```
function over (partition by a order by b RANGE|ROWS BETWEEN start AND end)
```

窗口函数包含以下三个部分。

- 分区规范：用于将输入行分散到不同的分区中，过程和 `GROUP BY` 子句的分散过程相似。
- 排序规范：决定输入数据行在窗口函数中执行的顺序。
- 窗口区间：指定计算数据的窗口边界。

  窗口区间支持 `RANGE` 、 `ROWS` 两种模式：
  - `RANGE` 按照计算列值的范围进行定义。
  - `ROWS` 按照计算列的行数进行范围定义。
  - `RANGE` 、 `ROWS` 中可以使用 `BETWEEN start AND end` 指定边界可取值。 `BETWEEN start AND end` 取值为：
    - `CURRENT ROW` ，当前行。
    - `N PRECEDING` ，前 `n` 行。
    - `UNBOUNDED PRECEDING` ，直到第 `1` 行。
    - `N FOLLOWING` ，后 `n` 行。
    - `UNBOUNDED FOLLOWING` ，直到最后 `1` 行。

例如，以下查询根据当前窗口的每行数据计算 `profit` 的部分总和。

```
select year,country,profit,sum(profit) over (partition by country order by year ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW) as slidew
indow from testwindow;
+------+---------+--------+-------------+
| year | country | profit | slidewindow |
+------+---------+--------+-------------+
| 2001 | USA     |     50 |          50 |
| 2001 | USA     |   1500 |        1550 |
| 2000 | India   |     75 |          75 |
| 2000 | India   |     75 |         150 |
| 2001 | India   |     79 |         229 |
| 2000 | Finland |   1500 |        1500 |
| 2001 | Finland |     10 |        1510 |
```

而以下查询只能计算出 `profit` 的总和。

```
select country,sum(profit) over (partition by country) from testwindow;
+---------+-------------------------------------------+
| country | sum(profit) OVER (PARTITION BY country) |
+---------+-------------------------------------------+
| India   |                                       229 |
| India   |                                       229 |
| India   |                                       229 |
| USA     |                                      1550 |
| USA     |                                      1550 |
| Finland |                                      1510 |
| Finland |                                      1510 |
```

注意事项

边界值的取值有如下要求：

- `start` 不能为 `UNBOUNDED FOLLOWING` ，否则提示 `Window frame start cannot be UNBOUNDED FOLLOWING` 错误。
- `end` 不能为 `UNBOUNDED PRECEDING` ，否则提示 `Window frame end cannot be UNBOUNDED PRECEDING` 错误。
- `start` 为 `CURRENT ROW` 并且 `end` 为 `N PRECEDING` 时，将提示 `Window frame starting from CURRENT ROW cannot end with PRECEDING` 错误。
- `start` 为 `N FOLLOWING` 并且 `end` 为 `N PRECEDING` 时，将提示 `Window frame starting from FOLLOWING cannot end with PRECEDING` 错误。

- `start` 为 `N FOLLOWING` 并且 `end` 为 `CURRENT ROW`，将提示 `Window frame starting from FOLLOWING cannot end with CURRENT ROW` 错误。

当模式为 `RANGE` 时：

- `start` 或者 `end` 为 `N PRECEDING` 时，将提示 `Window frame RANGE PRECEDING is only supported with UNBOUNDED` 错误。
- `start` 或者 `end` 为 `N FOLLOWING` 时，将提示 `Window frame RANGE FOLLOWING is only supported with UNBOUNDED` 错误。

准备工作

本文中的窗口函数均以 `testwindow` 表为测试数据。

```
create table testwindow(year int, country varchar(20), product varchar(20), profit int) distributed by hash(year);
```

```
insert into testwindow values (2000,'Finland','Computer',1500);
insert into testwindow values (2001,'Finland','Phone',10);
insert into testwindow values (2000,'India','Calculator',75);
insert into testwindow values (2000,'India','Calculator',75);
insert into testwindow values (2001,'India','Calculator',79);
insert into testwindow values (2001,'USA','Calculator',50);
insert into testwindow values (2001,'USA','Computer',1500);
```

```
SELECT * FROM testwindow;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2001 | Finland | Phone      |     10 |
| 2000 | India   | Calculator |     75 |
| 2000 | India   | Calculator |     75 |
| 2001 | India   | Calculator |     79 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   1500 |
```

聚合函数

所有聚合函数都可以通过添加 `OVER` 子句来作为窗口函数使用，聚合函数将基于当前滑动窗口内的数据行计算每一行数据。

例如，通过以下查询循环显示每个店员每天的订单额总和。

```
SELECT clerk, orderdate, orderkey, totalprice,sum(totalprice) OVER (PARTITION BY clerk ORDER BY orderdate) AS rolling_sum FROM orders ORDER
BY clerk, orderdate, orderkey
```

CUME_DIST

```
CUME_DIST()
```

- 命令说明：返回一组数值中每个值的累计分布。

  返回结果：在窗口分区中对窗口进行排序后的数据集，包括当前行和当前行之前的数据行数。排序中任何关联值均会计算成相同的分布值。
- 返回值类型：DOUBLE。
- 示例：

```
select year,country,product,profit,cume_dist() over (partition by country order by profit) as cume_dist from testwindow;
+------+---------+------------+--------+--------------------+
| year | country | product    | profit | cume_dist          |
+------+---------+------------+--------+--------------------+
| 2001 | USA     | Calculator |     50 |                0.5 |
| 2001 | USA     | Computer   |   1500 |                1.0 |
| 2001 | Finland | Phone      |     10 |                0.5 |
| 2000 | Finland | Computer   |   1500 |                1.0 |
| 2000 | India   | Calculator |     75 | 0.6666666666666666 |
| 2000 | India   | Calculator |     75 | 0.6666666666666666 |
| 2001 | India   | Calculator |     79 |                1.0 |
```

RANK

```
RANK()
```

- 命令说明：返回数据集中每个值的排名。

  排名值是将当前行之前的行数加1，不包含当前行。因此，排序的关联值可能产生顺序上的空隙，而且这个排名会对每个窗口分区进行计算。
- 返回值类型：BIGINT。
- 示例：

```
select year,country,product,profit,rank() over (partition by country order by profit) as rank from testwindow;
+------+---------+------------+--------+------+
| year | country | product    | profit | rank |
+------+---------+------------+--------+------+
| 2001 | Finland | Phone      |     10 |    1 |
| 2000 | Finland | Computer   |   1500 |    2 |
| 2001 | USA     | Calculator |     50 |    1 |
| 2001 | USA     | Computer   |   1500 |    2 |
| 2000 | India   | Calculator |     75 |    1 |
| 2000 | India   | Calculator |     75 |    1 |
| 2001 | India   | Calculator |     79 |    3 |
```

DENSE_RANK

```
DENSE_RANK()
```

- 命令说明：返回一组数值中每个数值的排名。

  `DENSE_RANK()` 与 `RANK()` 功能相似，但是 `DENSE_RANK()` 关联值不会产生顺序上的空隙。
- 返回值类型：BIGINT。
- 示例：

```
select year,country,product,profit,dense_rank() over (partition by country order by profit) as dense_rank from testwindow;
+------+---------+------------+--------+------------+
| year | country | product    | profit | dense_rank |
+------+---------+------------+--------+------------+
| 2001 | Finland | Phone      |     10 |          1 |
| 2000 | Finland | Computer   |   1500 |          2 |
| 2001 | USA     | Calculator |     50 |          1 |
| 2001 | USA     | Computer   |   1500 |          2 |
| 2000 | India   | Calculator |     75 |          1 |
| 2000 | India   | Calculator |     75 |          1 |
| 2001 | India   | Calculator |     79 |          2 |
```

NTILE

```
NTILE(n)
```

- 命令说明：将每个窗口分区的数据分散到桶号从 `1` 到 `n` 的 `n` 个桶中。

  桶号值最多间隔 `1` ，如果窗口分区中的数据行数不能均匀地分散到每一个桶中，则剩余值将从第 `1` 个桶开始，每 `1` 个桶分 `1` 行数据。例如，有6行数据和4个桶， 最终桶号值为 `1 1 2 2 3 4` 。
- 返回值类型：BIGINT。
- 示例：

```
select year,country,product,profit,ntile(2) over (partition by country order by profit) as ntile2 from testwindow;
+------+---------+------------+--------+--------+
| year | country | product    | profit | ntile2 |
+------+---------+------------+--------+--------+
| 2001 | USA     | Calculator |     50 |      1 |
| 2001 | USA     | Computer   |   1500 |      2 |
| 2001 | Finland | Phone      |     10 |      1 |
| 2000 | Finland | Computer   |   1500 |      2 |
| 2000 | India   | Calculator |     75 |      1 |
| 2000 | India   | Calculator |     75 |      1 |
| 2001 | India   | Calculator |     79 |      2 |
```

ROW_NUMBER

```
ROW_NUMBER()
```

- 命令说明：根据行在窗口分区内的顺序，为每行数据返回一个唯一的有序行号，行号从 `1` 开始。
- 返回值类型：BIGINT。
- 示例：

```
SELECT year, country, product, profit, ROW_NUMBER() OVER(PARTITION BY country) AS row_num1 FROM testwindow;
+------+---------+------------+--------+----------+
| year | country | product    | profit | row_num1 |
+------+---------+------------+--------+----------+
| 2001 | USA     | Calculator |     50 |        1 |
| 2001 | USA     | Computer   |   1500 |        2 |
| 2000 | India   | Calculator |     75 |        1 |
| 2000 | India   | Calculator |     75 |        2 |
| 2001 | India   | Calculator |     79 |        3 |
| 2000 | Finland | Computer   |   1500 |        1 |
| 2001 | Finland | Phone      |     10 |        2 |
```

PERCENT_RANK

```
PERCENT_RANK()
```

- 命令说明：返回数据集中每个数据的排名百分比，其结果由 `(r - 1) / (n - 1)` 计算得出。其中，`r` 为 `RANK()` 计算的当前行排名，`n` 为当前窗口分区内总的行数。
- 返回值类型：DOUBLE。
- 示例：

```
select year,country,product,profit,PERCENT_RANK() over (partition by country order by profit) as ntile3 from testwindow;
+------+---------+------------+--------+--------+
| year | country | product    | profit | ntile3 |
+------+---------+------------+--------+--------+
| 2001 | Finland | Phone      |     10 |    0.0 |
| 2000 | Finland | Computer   |   1500 |    1.0 |
| 2001 | USA     | Calculator |     50 |    0.0 |
| 2001 | USA     | Computer   |   1500 |    1.0 |
| 2000 | India   | Calculator |     75 |    0.0 |
| 2000 | India   | Calculator |     75 |    0.0 |
| 2001 | India   | Calculator |     79 |    1.0 |
```

FIRST_VALUE

```
FIRST_VALUE(x)
```

- 命令说明：返回窗口分区第一行的值。
- 返回值类型：与输入参数类型相同。
- 示例：

```
select year,country,product,profit,first_value(profit) over (partition by country order by profit) as firstValue from testwindow;
+------+---------+------------+--------+------------+
| year | country | product    | profit | firstValue |
+------+---------+------------+--------+------------+
| 2000 | India   | Calculator |     75 |         75 |
| 2000 | India   | Calculator |     75 |         75 |
| 2001 | India   | Calculator |     79 |         75 |
| 2001 | USA     | Calculator |     50 |         50 |
| 2001 | USA     | Computer   |   1500 |         50 |
| 2001 | Finland | Phone      |     10 |         10 |
| 2000 | Finland | Computer   |   1500 |         10 |
```

LAST_VALUE

```
LAST_VALUE(x)
```

- 命令说明：返回窗口分区最后一行的值。LAST_VALUE默认统计范围是 rows between unbounded preceding and current row，即取当前行数据与当前行之前的数据进行比较。如果像FIRST_VALUE那样直接在每行数据中显示最后一行数据，需要在 order by 条件的后面加上语句：rows between unbounded preceding and unbounded following。
- 返回值类型：与输入参数类型相同。
- 示例1：

```
select year,country,product,profit,last_value(profit) over (partition by country order by profit) as firstValue from testwindow;
+---------------+------------------+------------------+------------------+----------------------+
| year          | country          | product          | profit           | firstValue           |
+---------------+------------------+------------------+------------------+----------------------+
|          2001 | USA              | Calculator       |               50 |                   50 |
|          2001 | USA              | Computer         |             1500 |                 1500 |
|          2001 | Finland          | Phone            |               10 |                   10 |
|          2000 | Finland          | Computer         |             1500 |                 1500 |
|          2000 | India            | Calculator       |               75 |                   75 |
|          2000 | India            | Calculator       |               75 |                   75 |
|          2001 | India            | Calculator       |               79 |                   79 |
```

- 示例2：

```
select year,country,product,profit,last_value(profit) over (partition by country order by profit rows between unbounded preceding and unb
ounded following) as lastValue from testwindow;
+------+---------+------------+--------+-----------+
| year | country | product    | profit | lastValue |
+------+---------+------------+--------+-----------+
| 2001 | Finland | Phone      |     10 |      1500 |
| 2000 | Finland | Computer   |   1500 |      1500 |
| 2000 | India   | Calculator |     75 |        79 |
| 2000 | India   | Calculator |     75 |        79 |
| 2001 | India   | Calculator |     79 |        79 |
| 2001 | USA     | Calculator |     50 |      1500 |
| 2001 | USA     | Computer   |   1500 |      1500 |
+------+---------+------------+--------+-----------+
```

LAG

```
LAG(x[, offset[, default_value]])
```

- 命令说明：返回窗口内距离当前行之前偏移 `offset` 后的值。

  偏移量起始值是 `0` ，也就是当前数据行。偏移量可以是标量表达式，默认 `offset` 是 `1` 。

  如果偏移量的值是 `null` 或者大于窗口长度，则返回 `default_value` ；如果没有指定 `default_value` ，则返回 `null` 。
- 返回值类型：与输入参数类型相同。
- 示例：

```
select year,country,product,profit,lag(profit) over (partition by country order by profit) as lag from testwindow;
+------+---------+------------+--------+------+
| year | country | product    | profit | lag  |
+------+---------+------------+--------+------+
| 2001 | USA     | Calculator |     50 | NULL |
| 2001 | USA     | Computer   |   1500 |   50 |
| 2000 | India   | Calculator |     75 | NULL |
| 2000 | India   | Calculator |     75 |   75 |
| 2001 | India   | Calculator |     79 |   75 |
| 2001 | Finland | Phone      |     10 | NULL |
| 2000 | Finland | Computer   |   1500 |   10 |
```

LEAD

```
LEAD(x[,offset[, default_value]])
```

- 命令说明：返回窗口内距离当前行偏移 `offset` 后的值。

  偏移量 `offset` 起始值是 `0` ，也就是当前数据行。偏移量可以是标量表达式，默认 `offset` 是 `1` 。

  如果偏移量的值是 `null` 或者大于窗口长度，则返回 `default_value` ；如果没有指定 `default_value` ，则返回 `null` 。
- 返回值类型：与输入参数类型相同。
- 示例：

```
select year,country,product,profit,lead(profit) over (partition by country order by profit) as lead from testwindow;
+------+---------+------------+--------+------+
| year | country | product    | profit | lead |
+------+---------+------------+--------+------+
| 2000 | India   | Calculator |     75 |   75 |
| 2000 | India   | Calculator |     75 |   79 |
| 2001 | India   | Calculator |     79 | NULL |
| 2001 | Finland | Phone      |     10 | 1500 |
| 2000 | Finland | Computer   |   1500 | NULL |
| 2001 | USA     | Calculator |     50 | 1500 |
| 2001 | USA     | Computer   |   1500 | NULL |
```

NTH_VALUE

```
NTH_VALUE(x, offset)
```

- 命令说明：返回窗口内偏移指定 `offset` 后的值，偏移量从 `1` 开始。

  如果偏移量 `offset` 是 `null` 或者大于窗口内值的个数，则返回 `null` ；如果偏移量 `offset` 为 `0` 或者负数，则系统提示报错。
- 返回值类型：与输入参数类型相同。
- 示例：

```
select year,country,product,profit,nth_value(profit,1) over (partition by country order by profit) as nth_value from testwindow;
+------+---------+------------+--------+-----------+
| year | country | product    | profit | nth_value |
+------+---------+------------+--------+-----------+
| 2001 | Finland | Phone      |     10 |        10 |
| 2000 | Finland | Computer   |   1500 |        10 |
| 2001 | USA     | Calculator |     50 |        50 |
| 2001 | USA     | Computer   |   1500 |        50 |
| 2000 | India   | Calculator |     75 |        75 |
| 2000 | India   | Calculator |     75 |        75 |
| 2001 | India   | Calculator |     79 |        75 |
```

4.9. 聚合函数

本文介绍AnalyticDB MySQL版支持的聚合函数。

AnalyticDB MySQL版支持如下聚合函数：

- ARBITRARY：随机返回一组数据中的任意一个值。
- AVG：该函数用于计算平均值。
- BIT_AND：返回参数所有位按位AND后的结果。
- BIT_OR：返回参数所有位按位OR后的结果。
- BIT_XOR：返回参数所有位按位异或后的结果。
- COUNT：该函数用于计算记录数。
- MAX：该函数用于计算最大值。
- MIN：该函数用于计算最小值。
- STD或STDDEV：返回数值的样本标准差。
- STDDEV_POP：返回数值的总体标准差。
- STDDEV_SAMP：返回一组数值（整数、小数或浮点）的样本标准差。
- SUM：该函数用于计算汇总值。
- VARIANCE（非标准SQL函数）：返回一组数值（整数、小数或浮点）的总体方差。
- VAR_POP（标准SQL函数）：返回一组数值（整数、小数或浮点）的总体方差。
- VAR_SAMP：返回一组数值（整数、小数或浮点）的样本方差。
- GROUP_CONCAT：该函数用于将 `GROUP BY` 返回结果中属于同一个分组的值连接起来，返回一个字符串结果。

> 说明
>
> 除 `GROUP_CONCAT` 函数外，本文中的其它聚合函数均以 `testtable` 表为例，建表语句如下：
>
> ```
> CREATE TABLE testtable(a INT) DISTRIBUTED BY HASH(a);
> ```
>
> 已使用如下语句往 `testtable` 表中插入测试数据：
>
> ```
> INSERT INTO testtable VALUES (1),(2),(3);
> ```

ARBITRARY

```
arbitrary(x)
```

- 命令说明：随机返回一组数据中的任意一个值。
- 输入值类型：支持输入任意类型的参数。
- 返回值类型：与该函数的输入值类型保持一致。
- 示例：

```
SELECT arbitrary(a) FROM testtable;
```

返回结果如下：

```
+--------------+
| arbitrary(a) |
+--------------+
|            2 |
+--------------+
```

AVG

```
avg(x)
```

- 命令说明：该函数用于计算平均值。
- 输入值类型：BIGINT、DOUBLE或FLOAT。

- 返回值类型：DOUBLE。
- 示例：

```
SELECT avg(a) FROM testtable;
```

返回结果如下：

```
+--------+
| avg(a) |
+--------+
|    2.0 |
+--------+
```

BIT_AND

```
bit_and(x)
```

- 命令说明：返回参数所有位按位`AND`后的结果。
- 输入值类型：BIGINT、DOUBLE或FLOAT。
- 返回值类型：BIGINT。
- 示例：

```
SELECT bit_and(a) FROM testtable;
```

返回结果如下：

```
+------------+
| bit_and(a) |
+------------+
|          0 |
+------------+
```

BIT_OR

```
bit_or(x)
```

- 命令说明：返回参数所有位按位`OR`后的结果。
- 输入值类型：BIGINT、DOUBLE或FLOAT。
- 返回值类型：BIGINT。
- 示例：

```
SELECT bit_or(a) FROM testtable;
```

返回结果如下：

```
+-----------+
| bit_or(a) |
+-----------+
|         3 |
+-----------+
```

BIT_XOR

```
bit_xor(x)
```

- 命令说明：返回参数所有位按位异或后的结果。
- 输入值类型：BIGINT、DOUBLE或FLOAT。
- 返回值类型：BIGINT。
- 示例：

```
SELECT bit_xor(a) FROM testtable;
```

返回结果如下：

```
+------------+
| bit_xor(a) |
+------------+
|          0 |
+------------+
```

COUNT

```
count([distinct|all] x)
```

- 命令说明：该函数用于计算记录数。

> **说明** `distinct`、`all`指明在计数时是否去除重复记录，默认`all`，即返回全部记录。如果指定`distinct`，返回结果只计算唯一值数量。

- 输入值类型：数值、字符串类型或BOOLEAN类型。
- 返回值类型：BIGINT。
- 示例：
  - 计算`testtable`中值是唯一的记录数，语句如下：

    ```
    SELECT count(distinct a) FROM testtable;
    ```

    返回结果如下：

    ```
    +--------------------+
    | count(distinct a) |
    +--------------------+
    |                  3 |
    +--------------------+
    ```

  - 计算`testtable`中所有的记录数，语句如下：

    ```
    SELECT count(all a) FROM testtable;
    ```

    返回结果如下：

    ```
    +---------------+
    | count(all a) |
    +---------------+
    |            3 |
    +---------------+
    ```

MAX

```
max(x)
```

- 命令说明：该函数用于计算最大值。
- 输入值类型：该函数支持输入任意类型的参数，但是BOOLEAN类型的数据不允许参与运算。

> **说明** 当列中的值为`NULL`时，该行不参与计算。

- 返回值类型：与该函数的输入值类型保持一致。
- 示例：

```
SELECT max(a) FROM testtable;
```

返回结果如下：

```
+---------+
| max(a) |
+---------+
|      3 |
+---------+
```

MIN

```
min(value x)
```

- 命令说明：该函数用于计算最小值。
- 输入值类型：该函数支持输入任意类型的参数，但是BOOLEAN类型的数据不允许参与运算。

> **说明** 当列中的值为`NULL`时，该行不参与计算。

- 返回值类型：与该函数的输入值类型保持一致。
- 示例：

```
SELECT min(a) FROM testtable;
```

返回结果如下：

```
+---------+
| min(a) |
+---------+
|      1 |
+---------+
```

STD或STDDEV

```
std(x)
stddev(x)
```

- 命令说明：返回数值的样本标准差。
- 输入值类型：BIGINT或DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT std(a) FROM testtable;
```

返回结果如下：

```
+-------------------+
| std(a)            |
+-------------------+
| 0.816496580927726 |
+-------------------+
```

STDDEV_POP

```
stddev_pop(x)
```

- 命令说明：返回数值的总体标准差。
- 输入值类型：BIGINT或DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT stddev_pop(a) FROM testtable;
```

返回结果如下：

```
+-------------------+
| stddev_pop(a)     |
+-------------------+
| 0.816496580927726 |
+-------------------+
```

STDDEV_SAMP

```
stddev_samp(x)
```

- 命令说明：返回一组数值（整数、小数或浮点）的样本标准差。
- 输入值类型：BIGINT或DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT stddev_samp(a) FROM testtable;
```

返回结果如下：

```
+----------------+
| stddev_samp(a) |
+----------------+
|            1.0 |
+----------------+
```

SUM

```
sum(x)
```

- 命令说明：该函数用于计算汇总值。
- 输入值类型：BIGINT、DOUBLE或FLOAT。
- 返回值类型：BIGINT。
- 示例：

```
SELECT sum(a) FROM testtable;
```

返回结果如下：

```
+--------+
| sum(a) |
+--------+
|      6 |
+--------+
```

VARIANCE

```
variance(x)
```

- 命令说明：返回一组数值（整数、小数或浮点）的总体标准方差。

  > 说明
  > - VARIANCE() 会忽略值为NULL的行。因此若一组数值全为NULL，VARIANCE() 会直接返回NULL。
  > - VARIANCE() 函数作为标准SQL的延伸，您也可以使用标准SQL函数 VAR_POP() 来代替。

- 输入值类型：BIGINT或DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT variance(a)  FROM testtable;
+----------------------------+
|         variance(a)        |
+----------------------------+
|    0.6666666666666666      |
```

VAR_POP

```
var_pop(x)
```

- 命令说明：返回一组数值 x （整数、小数或浮点）的总体标准方差。

  > 说明
  > - VAR_POP() 会忽略值为NULL的行。因此若一组数值全为NULL，VAR_POP() 会直接返回NULL。
  > - 也可以使用 VARIANCE() 函数，具有相同的意义，但 VARIANCE() 不是标准的SQL函数。

- 输入值类型：BIGINT或DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT var_pop(a) FROM testtable;
```

返回结果如下：

```
+--------------------+
| var_pop(a)         |
+--------------------+
| 0.6666666666666666 |
+--------------------+
```

VAR_SAMP

```
var_samp(x)
```

- 命令说明：返回一组数值（整数、小数或浮点）的样本方差。
- 输入值类型：BIGINT或DOUBLE。
- 返回值类型：DOUBLE。
- 示例：

```
SELECT var_samp(a)  FROM testtable;
```

返回结果如下：

```
+-------------+
| var_samp(a) |
+-------------+
|         1.0 |
+-------------+
```

GROUP_CONCAT

```
GROUP_CONCAT([DISTINCT] col_name
             [ORDER BY col_name [ASC | DESC]]
             [SEPARATOR str_val])
```

参数	是否必填	说明
DISTINCT	选填	指定需要去重的列。
ORDER BY		指定需要在分组内部进行排序的列，支持如下排序方式： • ASC：升序。 • DESC：降序。 若未指定排序方式，默认按照升序排序。
SEPARATOR		指定分组内部各值间的分隔符。 若未指定分隔符，默认使用英文逗号（,）分隔。

- 命令说明：该函数用于将 GROUP BY 返回结果中属于同一个分组的值连接起来，返回一个字符串结果。

  > **说明** 仅当需要使用 GROUP_CONCAT 函数进行连接的列中所有取值均为NULL时，才会输出 NULL 。

- 输入值类型：字符串。
- 返回值类型：字符串。
- 示例

  本示例以 person 表为例介绍如何使用 GROUP_CONCAT 函数， person 表创建语句如下：

  ```
  CREATE TABLE person(id INT,name VARCHAR,age INT ) DISTRIBUTED BY HASH(id);
  ```

  使用如下语句往表中插入数据：

  ```
  INSERT INTO person VALUES (1,'mary',13),(2,'eva',14),(2,'adam',13),(3,'eva',13),(3,null,13),(3,null,null),(4,null,13),(4,null,null);
  ```

  现需要根据 id 列进行分组，并通过 GROUP_CONCAT 函数将ID相同的 name 列展示出来，展示结果时需要对 name 列去重，并按照 name 列进行降序排序，多个 name 列间用 # 分隔，语句如下：

  ```
  SELECT
    id,
    GROUP_CONCAT(DISTINCT name ORDER BY name DESC SEPARATOR '#')
  FROM person
    GROUP BY id;
  ```

  返回结果如下：

  ```
  +------+-----------------------------------------------------------------+
  | id   | GROUP_CONCAT(DISTINCT name ORDER BY name DESC SEPARATOR '#') |
  +------+-----------------------------------------------------------------+
  |    2 | eva#adam                                                        |
  |    1 | mary                                                            |
  |    4 | NULL                                                            |
  |    3 | eva                                                             |
  +------+-----------------------------------------------------------------+
  ```

4.10. 可变长二进制函数

AnalyticDB for MySQL支持以下可变长二进制函数。

- LENGTH：以字节为单位返回参数的长度。
- CHAR_LENGTH：以字符为单位返回字符串的长度。
- ORD：如果字符串最左边的字符是多字节字符，则返回该字符的代码。
- HEX：将字符串或数字转换为十六进制形式，并返回结果。
- UNHEX：将参数中的每对字符转换为十六进制形式，再将其转换为数字表示的字节，最后以二进制字符串形式返回结果。
- MD5：计算参数MD5的hash值。
- SHA1：计算字符串的SHA-1校验和。
- CRC32：返回参数 x 的循环冗余码。
- LOWER：返回参数的小写形式。
- UPPER：返回参数的大写形式。
- UPPER：返回参数的大写形式。
- LTRIM：移除字符串左边的空白字符。
- RTRIM：移除字符串右边的空白字符。
- TRIM：移除字符串左右两边的空白字符
- COMPRESS：压缩一个字符串并将结果作为二进制字符串返回。
- UNCOMPRESS：解压缩由 COMPRESS() 函数压缩的字符串。
- UNCOMPRESSED_LENGTH：在压缩之前返回一个字符串的长度。
- ENCRYPT：对字符串进行加密。

- AES_ENCRYPT：使用AES算法进行数据加密。
- AES_DECRYPT：使用AES算法进行数据解密。
- SUBSTR：返回指定子字符串。
- LEFT：返回字符串最左边的`y`个字符。
- RIGHT：返回字符串最右边的`y`个字符。
- REPEAT：返回重复指定次数的字符串。
- REVERSE：反转字符串中的字符。
- SHA2：计算SHA-2校验和。
- LPAD：左拼接字符串。
- RPAD：右拼接字符串。
- TO_BASE64：返回字符串的BASE64编码形式。
- FROM_BASE64：解码BASE64编码的字符串并返回结果。

LENGTH

```
length(varbinary x)
```

- 命令说明：以字节为单位返回参数 `x` 的长度。
- 返回值类型：LONG
- 示例：

```
select length(CAST('abc' AS VARBINARY));
+----------------------------------+
| length(CAST('abc' AS varbinary)) |
+----------------------------------+
|                                3 |
```

CHAR_LENGTH

```
char_length(varbinary x)
```

- 命令说明：以字符为单位返回字符串 `x` 的长度。
- 返回值类型：LONG
- 示例：

```
select char_length(CAST('abc' AS VARBINARY));
+---------------------------------------+
| char_length(CAST('abc' AS varbinary)) |
+---------------------------------------+
|                                     3 |
```

ORD

```
ord(varbinary x)
```

- 命令说明：如果字符串 `x` 最左边的字符是多字节字符，则返回该字符的代码。
- 返回值类型：LONG
- 示例：

```
select ord(CAST('中国' AS VARBINARY));
+----------------------------------+
| ord(CAST('中国' AS varbinary))   |
+----------------------------------+
|                              228 |
```

HEX

```
hex(varbinary x)
```

- 命令说明：将字符串 `x` 或数字转换为十六进制格式的字符串。
- 返回值类型：VARCHAR
- 示例：

```
select hex(CAST('中国' AS VARBINARY));
+----------------------------------+
| hex(CAST('中国' AS varbinary))   |
+----------------------------------+
| E4B8ADE59BBD                     |
```

UNHEX

```
unhex(varbinary x)
```

- 命令说明：将参数 `x` 中的每对字符解释为十六进制数，并将其转换为数字表示的字节，以二进制字符串将其返回。
- 返回值类型：VARBINARY
- 示例：

```
select unhex(CAST('中国' AS VARBINARY));
+-----------------------------------+
| unhex(CAST('中国' AS varbinary))   |
+-----------------------------------+
| NULL                              |
```

MD5

```
md5(varbinary x)
```

- 命令说明：返回参数 `x` MD5的hash值。
- 返回值类型：VARCHAR
- 示例：

```
select md5(CAST('中国' AS VARBINARY));
+-----------------------------------+
| md5(CAST('中国' AS varbinary))   |
+-----------------------------------+
| c13dceabcb143acd6c9298265d618a9f |
```

SHA1

```
sha1(varbinary x)
```

- 命令说明：计算字符串 `x` 的SHA-1校验和。
- 返回值类型：VARCHAR
- 示例：

```
select sha1(CAST('中国' AS VARBINARY));
+-------------------------------------------+
| sha1(CAST('中国' AS varbinary))          |
+-------------------------------------------+
| 101806f57c322fb403a9788c4c24b79650d02e77 |
```

CRC32

```
crc32(varbinary x)
```

- 命令说明：返回参数 `x` 的循环冗余码。
- 返回值类型：LONG
- 示例：

```
select crc32(CAST('中国' AS VARBINARY));
+-----------------------------------+
| crc32(CAST('中国' AS varbinary))   |
+-----------------------------------+
|                         737014929 |
```

LOWER

```
lower(varbinary x)
```

- 命令说明：返回参数 `x` 的小写形式。
- 返回值类型：VARBINARY
- 示例：

```
select lower(CAST('ABC' AS VARBINARY));
+---------------------------------+
| lower(CAST('ABC' AS varbinary)) |
+---------------------------------+
| abc                             |
```

UPPER

```
upper(varbinary x)
```

- 命令说明：返回参数 `x` 的大写形式。
- 返回值类型：VARBINARY
- 示例：

```
select upper(CAST('abc' AS VARBINARY));
+---------------------------------+
| upper(CAST('abc' AS varbinary)) |
+---------------------------------+
| ABC                             |
```

LTRIM

```
ltrim(varbinary x)
```

- 命令说明：移除字符串 `x` 左边的空白字符。
- 返回值类型：VARBINARY
- 示例：

```
select ltrim(CAST('  中国  ' AS VARBINARY));
+---------------------------------------+
| ltrim(CAST('  中国  ' AS varbinary))  |
+---------------------------------------+
| 中国                                  |
```

RTRIM

```
rtrim(varbinary x)
```

- 命令说明：移除字符串 `x` 右边的空白字符。
- 返回值类型：VARBINARY
- 示例：

```
select rtrim(CAST('  中国  ' AS VARBINARY));
+---------------------------------------+
| rtrim(CAST('  中国  ' AS varbinary))  |
+---------------------------------------+
|   中国                                |
```

TRIM

```
trim(varbinary x)
```

- 命令说明：移除字符串 `x` 左右两边的空白字符。
- 返回值类型：VARBINARY
- 示例：

```
select trim(CAST('  中国  ' AS VARBINARY));
+---------------------------------------+
| trim(CAST('  中国  ' AS varbinary))   |
+---------------------------------------+
| 中国                                  |
```

COMPRESS

```
compress(varbinary x)
```

- 命令说明：压缩字符串 `x` 并将结果作为二进制字符串返回。
- 返回值类型：VARBINARY
- 示例：

```
select hex(compress(CAST('中国' AS VARBINARY)));
+-----------------------------------------------+
| hex(compress(CAST('中国' AS varbinary)))      |
+-----------------------------------------------+
| 06000000789C7BB263EDD3D97B01104C0487          |
```

UNCOMPRESS

```
uncompress(varbinary x)
```

- 命令说明：解压缩 `x` 由 `COMPRESS()` 函数压缩的字符串。
- 返回值类型：VARBINARY

- 示例：

```
select uncompress(compress(CAST('中国' AS VARBINARY)));
+-------------------------------------------------+
| uncompress(compress(CAST('中国' AS varbinary)))  |
+-------------------------------------------------+
| 中国                                             |
```

UNCOMPRESSED_LENGTH

```
uncompressed_length(varbinary x)
```

- 命令说明：在压缩之前返回字符串 `x` 的长度。
- 返回值类型：LONG
- 示例：

```
select uncompressed_length(compress(CAST('中国' AS VARBINARY)));
+-----------------------------------------------------------+
| uncompressed_length(compress(CAST('中国' AS varbinary)))  |
+-----------------------------------------------------------+
|                                                         6 |
```

ENCRYPT

```
encrypt(varbinary x, varchar y)
```

- 命令说明：对参数 `x` 进行加密， `y` 为Salt值。
- 返回值类型：VARBINARY
- 示例：

```
select encrypt('abdABC123','key');
+-----------------------------+
| encrypt('abdABC123', 'key') |
+-----------------------------+
| kezazmcIo.aCw               |
```

AES_ENCRYPT

```
aes_encrypt(varbinary x, varchar y)
```

- 命令说明：使用AES算法加密 `x` ， `y` 为密钥。
- 返回值类型：VARBINARY
- 示例：

```
select HEX(aes_encrypt(CAST('中国' AS VARBINARY), '0123'));
+-----------------------------------------------------------+
| HEX(aes_encrypt(CAST('中国' AS VARBINARY), '0123'))       |
+-----------------------------------------------------------+
| DFB166F0A03113AA848C0CE545D58757                          |
```

AES_DECRYPT

```
aes_decrypt(varbinary x, varchar y)
```

- 命令说明：使用AES算法解密 `x` ， `y` 为密钥。
- 返回值类型：VARBINARY
- 示例：

```
select HEX(aes_decrypt(aes_encrypt(CAST('中国' AS VARBINARY), '0123'),'0123'));
+-------------------------------------------------------------------------------+
| HEX (aes_decrypt(aes_encrypt(CAST('中国' AS VARBINARY), '0123'),'0123'))      |
+-------------------------------------------------------------------------------+
| E4B8ADE59BBD                                                                  |
```

SUBSTR

```
substr(varbinary x, bigint y, double z)
substr(varbinary x, double y, double z)
substr(varbinary x, bigint y, bigint z)
substr(varbinary x, double y, bigint z)
substr(varbinary x, double y)
substr(varbinary x, bigint y)
```

- 命令说明：返回指定子字符串。

- 返回值类型：VARBINARY
- 示例：

```
select HEX(substr(CAST('中国' AS VARBINARY), 2));
+--------------------------------------------+
| HEX(substr(CAST('中国' AS VARBINARY), 2))  |
+--------------------------------------------+
| B8ADE59BBD                                 |
+--------------------------------------------+
```

```
+-----------------------------------------------------+
| HEX(substr(CAST('中国' AS VARBINARY), 2.7, 1.7))  |
+-----------------------------------------------------+
| ADE5                                                |
+-----------------------------------------------------+
```

LEFT

```
left(varbinary x, bigint y)
left(varbinary x, double y)
```

- 命令说明：返回字符串 `x` 最左边的 `y` 个字符。
- 返回值类型：VARBINARY
- 示例：

```
select left(CAST('中国' AS VARBINARY),1000);
+---------------------------------------------+
| LEFT(CAST('中国' AS VARBINARY), 1000)  |
+---------------------------------------------+
| 中国                                        |
```

RIGHT

```
right(varbinary x, bigint y)
right(varbinary x, double y)
```

- 命令说明：返回字符串 `x` 最右边的 `y` 个字符。
- 返回值类型：VARBINARY
- 示例：

```
select HEX(right(CAST('中国' AS VARBINARY),1));
+-----------------------------------------------+
| HEX(RIGHT(CAST('中国' AS VARBINARY), 1))  |
+-----------------------------------------------+
| BD                                            |
```

REPEAT

```
repeat(varbinary x, double y)
repeat(varbinary x, bigint y)
```

- 命令说明：返回重复指定次数 `y` 的字符串 `x` 。
- 返回值类型：VARBINARY
- 示例：

```
select HEX(repeat(CAST('中国' AS VARBINARY),1));
+-----------------------------------------------+
| HEX(repeat(CAST('中国' AS VARBINARY), 1))  |
+-----------------------------------------------+
| E4B8ADE59BBD                                  |
```

REVERSE

```
reverse(varbinary x)
```

- 命令说明：反转字符串 `x` 中的字符。
- 返回值类型：VARBINARY
- 示例：

```
select HEX(reverse(CAST('中国' AS VARBINARY)));
+-----------------------------------------------+
| HEX(reverse(CAST('中国' AS VARBINARY)))  |
+-----------------------------------------------+
| BD9BE5ADB8E4                                  |
```

SHA2

```
sha2(varbinary x, integer y)
sha2(varbinary x, varchar y)
```

- 命令说明：计算字符串 `x` SHA-2系列散列函数（SHA-224、SHA-256、SHA-384和SHA-512）。 `x` 是要清理的明文字符串， `y` 表示结果所需的位数长度，其值必须为224、256、384、512或0。
- 返回值类型：VARCHAR
- 示例：

```
select sha2(CAST('中国' AS VARBINARY),256);
+------------------------------------------------------------------+
| sha2(CAST('中国' AS varbinary), 256)                             |
+------------------------------------------------------------------+
| f0e9521611bb290d7b09b8cd14a63c3fe7cbf9a2f4e0090d8238d22403d35182 |
```

LPAD

```
lpad(varbinary x, bigint y, varchar z)
lpad(varbinary x, double y, varchar z)
```

- 命令说明：将字符串 `x` 左边拼接字符串 `z` 直到长度达到 `y` ，并返回拼接后的字符串。

  如果 `x` 长于 `y` ，则返回值将缩短为 `y` 个字符。
- 返回值类型：VARBINARY
- 示例：

```
select HEX(lpad(CAST('中国' AS VARBINARY), 7,'-'));
+-----------------------------------------------+
| HEX(lpad(CAST('中国' AS VARBINARY), 7, '-'))  |
+-----------------------------------------------+
| 2DE4B8ADE59BBD                                |
```

RPAD

```
rpad(varbinary x, bigint y, varchar z)
rpad(varbinary x, double y, varchar z)
```

- 命令说明：将字符串 `x` 右边拼接字符串 `z` 直到长度达到 `y` ，并返回拼接后的字符串。

  如果 `x` 长于 `y` ，则返回值将缩短为 `y` 个字符。
- 返回值类型：VARBINARY
- 示例：

```
select HEX(rpad(CAST('中国' AS VARBINARY), 4.7,'x'));
+-----------------------------------------------+
| HEX(rpad(CAST('中国' AS VARBINARY), 4.7, 'x'))  |
+-----------------------------------------------+
| E4B8ADE59B                                    |
```

TO_BASE64

```
to_base64(varbinary x)
```

- 命令说明：返回字符串 `x` 的BASE64编码形式。
- 返回值类型：VARCHAR
- 示例：

```
select to_base64(CAST('中国' AS VARBINARY));
+---------------------------------------+
| to_base64(CAST('中国' AS varbinary))  |
+---------------------------------------+
| 5Lit5Zu9                              |
```

FROM_BASE64

```
from_base64(varbinary x)
```

- 命令说明：解码BASE64编码的字符串 `x` 并返回结果。
- 返回值类型：VARBINARY
- 示例：

```
select from_base64(to_base64(CAST('abc' AS VARBINARY)));
+--------------------------------------------------+
| from_base64(to_base64(CAST('abc' AS varbinary))) |
+--------------------------------------------------+
| abc                                              |
```

4.11. CAST函数

AnalyticDB MySQL版支持通过 `cast` 函数来转换数据类型。本文介绍如何使用 `cast` 函数。

语法

```
cast(expr AS type)
```

其中：

- `expr`：源数据，如字符串 `'China'`。
- `type`：目标数据类型，例如 `varbinary`。

CAST AS BOOLEAN

```
cast (expr AS boolean)
```

- **命令说明**：将源数据 `expr` 转换为BOOLEAN类型。
- **源数据类型**：
  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
  - BIGINT
  - DOUBLE
  - VARCHAR
  - JSON
- **示例**：

  例如，将INT类型的数据 `1` 转换为BOOLEAN，语句如下：

  ```
  SELECT cast('1' AS boolean);
  ```

  返回结果如下：

  ```
  +----------------------+
  | cast('1' AS boolean) |
  +----------------------+
  |                    1 |
  +----------------------+
  ```

- **注意事项**：
  - 源VARCHAR类型或JSON类型的数据转换为BOOLEAN类型时，会出现如下几种情况：
    - 源数据为true或1，会被转换成1。
    - 源数据为false或0，会被转换成0。
    - 源数据为其他，转换时会报错。

    例如，将VARCHAR类型的数据 `a` 转换为BOOLEAN类型，语句如下：

    ```
    SELECT cast('a' AS boolean);
    ```

    此时会返回如下错误：

    ```
    ERROR 1815 (HY000): [30013, 20210917103441192168188048034539127 63] : Cannot cast 'a' to BOOLEAN
    ```

  - 源DOUBLE类型的数据转换为BOOLEAN类型时，会出现如下几种情况：
    - 非0.0数值会被转换成1。
    - 0.0会被转换成0。

    例如，将源DOUBLE的数据类型 `4.3` 转换为BOOLEAN类型，语句如下：

    ```
    SELECT cast(4.3 AS boolean);
    ```

    返回结果如下：

    ```
    +----------------------+
    | cast(4.3 AS boolean) |
    +----------------------+
    |                    1 |
    +----------------------+
    ```

- 源DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT或BIGINT类型的数据，转换为BOOLEAN类型时，会出现如下几种情况：
  - 非0数值会被转换成1。
  - 0会被转换成0。

  例如，将源INT的数据类型 `5` 转换为BOOLEAN类型，语句如下：

  ```
  SELECT cast(5 AS boolean);
  ```

  返回结果如下：

  ```
  +--------------------+
  | cast(5 AS boolean) |
  +--------------------+
  |                  1 |
  +--------------------+
  ```

CAST AS DECIMAL(m,d)|FLOAT|INT/INTEGER|SMALLINT|TINYINT

```
cast (expr AS decimal(m,d)|float|int/integer|smallint|tinyint)
```

- **命令说明**：将源数据 `expr` 转换为DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT或TINYINT类型。
- **源数据类型**：
  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
  - BIGINT
  - DOUBLE
  - VARCHAR
  - JSON
- **示例**：

  例如，将BIGINT类型的数据 `2001012` 转换为FLOAT类型，语句如下：

  ```
  SELECT cast( 2001012 AS float);
  ```

  返回结果如下：

  ```
  +-------------------------+
  | cast( 2001012 AS float) |
  +-------------------------+
  |              2001012.0 |
  +-------------------------+
  ```

- **注意事项**：
  - 若源类型数值超出了目标类型支持的数值范围，类型转换时会报错。

    例如，将BIGINT类型的数据 `99999999` 转换成SMALLINT类型，语句如下：

    ```
    SELECT cast(99999999 as smallint);
    ```

    由于 `99999999` 不在SMALLINT支持的数值范围内，此时会返回如下错误：

    ```
    ERROR 1815 (HY000): [31002, 2021091417365619216818804803453260208] : Out of range for smallint: 99999999
    ```

- 在如下类型转换的场景中，转换后数据精度会丢失：
  - DOUBLE类型转换为FLOAT类型。

    将源DOUBLE类型的数据 `1.23456789` 转换为FLOAT类型，语句如下：

    ```
    SELECT cast(1.23456789 AS float);
    ```

    返回结果如下：

    ```
    +---------------------------+
    | cast(1.23456789 AS float) |
    +---------------------------+
    |                1.2345679 |
    +---------------------------+
    ```

  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT或TINYINT类型间互相转换。

    例如，将源FLOAT类型的数据 `1.1342` 转换为INT类型，语句如下：

    ```
    SELECT cast(1.1342 AS int);
    ```

    返回结果如下：

    ```
    +---------------------+
    | cast(1.1342 AS int) |
    +---------------------+
    |                   1 |
    +---------------------+
    ```

- 若源VARCHAR类型的数据不是数值，转换为DECIMAL(m,d)或FLOAT类型时会报错。

  例如，将源VARCHAR类型的数据 `China` 转换为DECIMAL(m,d)类型，语句如下：

  ```
  SELECT cast( 'China' AS decimal(5,2));
  ```

  此时会返回如下错误：

  ```
  ERROR 1815 (HY000): [30013, 20210917152730192168188048034539618 57] : Cannot cast VARCHAR 'China' to DECIMAL(5, 2). Value is not a numbe
  r.
  ```

- 若源VARCHAR类型的数据不是数值，转换为INT/INTEGER、SMALLINT或TINYINT类型后的结果均为0。

  例如，将源VARCHAR类型的数据 `China` 转换为SMALLINT类型，语句如下：

  ```
  SELECT cast( 'China' AS smallint);
  ```

  返回结果如下：

  ```
  +-----------------------------+
  | cast( 'China' AS smallint) |
  +-----------------------------+
  |                           0 |
  +-----------------------------+
  ```

- 若源JSON类型的数据不是数值，转换为DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT或TINYINT时会报错。

  例如，将源VARCHAR类型的数据 `[1,2,3]` 先转换为JSON数据，再转换为SMALLINT类型，语句如下：

  ```
  SELECT cast(cast('[1,2,3]' AS json) AS smallint);
  ```

  此时会返回如下错误：

  ```
  ERROR 1815 (HY000): [20034, 20210918141031192168188048034531901 38] : Cannot cast json to smallint
  ```

CAST AS BIGINT

```
cast (expr AS bigint)
```

- **命令说明**：将源数据 `expr` 转换为BIGINT类型。
- **源数据类型**：
  - BOOLEAN
  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
  - DOUBLE
  - DATE、DATETIME、TIMESTAMP、TIME
  - VARCHAR
- **示例**：

- 将源DATE类型的数据 `2021-09-18` 转换为BIGINT类型，语句如下：

  ```
  SELECT cast(date '2021-09-18' AS bigint);
  ```

  返回结果如下：

  ```
  +-----------------------------------+
  | cast(date '2021-09-18' AS bigint) |
  +-----------------------------------+
  |                          20210918 |
  +-----------------------------------+
  ```

- 将源JSON类型的数据 `-1` 转换为BIGINT类型，语句如下：

  ```
  SELECT cast(json '-1' AS bigint);
  ```

  返回结果如下：

  ```
  +---------------------------+
  | cast(json '-1' AS bigint) |
  +---------------------------+
  |                        -1 |
  +---------------------------+
  ```

- **注意事项：**
  - 若源VARCHAR类型的数据不是数值，转换为BIGINT类型后的结果均为0。

    例如，将源VARCHAR类型的数据 `a` 转换为BIGINT，语句如下：

    ```
    SELECT cast('a' AS bigint);
    ```

    返回结果如下：

    ```
    +---------------------+
    | cast('a' AS bigint) |
    +---------------------+
    |                   0 |
    +---------------------+
    ```

  - 在如下类型转换的场景中，转换后数据精度会丢失：
    - DOUBLE类型转换为BIGINT类型。

      将源DOUBLE类型的数据 `1.23456789` 转换为BIGINT类型，语句如下：

      ```
      SELECT cast(1.23456789 AS bigint);
      ```

      返回结果如下：

      ```
      +-----------------------------+
      | cast(1.23456789 AS bigint) |
      +-----------------------------+
      |                           1 |
      +-----------------------------+
      ```

    - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT或TINYINT类型的数据转换为BIGINT类型。

      例如，将DECIMAL类型的数据 `1.1` 转换为BIGINT类型，语句如下：

      ```
      SELECT cast(1.1 AS bigint);
      ```

      返回结果如下：

      ```
      +----------------------+
      | cast(1.1 AS bigint) |
      +----------------------+
      |                    1 |
      +----------------------+
      ```

  - 若源JSON类型的数据不是数值，转换为BIGINT类型时会报错。

    例如，将源JSON类型的数据 `{}` 转换为BIGINT类型，语句如下：

    ```
    SELECT cast(json'{}'AS bigint);
    ```

    此时会返回如下报错：

    ```
    ERROR 1815 (HY000): [30013, 20210918153932192168188048034532050 32] : Cannot cast '{}' to bigint
    ```

CAST AS DOUBLE

```
cast (expr AS double)
```

- **命令说明**：将源数据 `expr` 转换为DOUBLE类型。
- **源数据类型**：
  - BOOLEAN
  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
  - BIGINT
  - DATE、DATETIME、TIMESTAMP、TIME
  - VARCHAR
  - JSON
- **示例**：

  将DATE类型的数据 `2021-09-17` 转换为DOUBLE类型，语句如下：

  ```
  SELECT cast( date '2021-09-17' AS double);
  ```

  返回结果如下：

  ```
  +-----------------------------------+
  | cast( date '2021-09-17' AS double) |
  +-----------------------------------+
  |                       2.0210917E7 |
  +-----------------------------------+
  ```

- **注意事项**：
  - 若源VARCHAR类型的数据不是数值，转换为DOUBLE类型后的结果均为0.0。

    例如，将源VARCHAR类型的数据 `China` 转换为DOUBLE类型，语句如下：

    ```
    SELECT cast( 'China' AS double);
    ```

    返回结果如下：

    ```
    +--------------------------+
    | cast( 'China' AS double) |
    +--------------------------+
    |                      0.0 |
    +--------------------------+
    ```

  - 若源JSON类型的数据不是数值，转换为DOUBLE时会报错。

    例如，将源JSON类型的数据 `{}` 转换为DOUBLE类型，语句如下：

    ```
    SELECT cast(json '{}' AS double);
    ```

    此时会返回如下错误：

    ```
    ERROR 1815 (HY000): [30013, 20210918161722192168188048034532113 59] : Cannot cast '{}' to double
    ```

CAST AS DATE|DATETIME|TIMESTAMP|TIME

```
cast (expr AS date|datetime|timestamp|time)
```

- **命令说明**：将源数据 `expr` 转换为DATE、DATETIME、TIMESTAMP或TIME类型。
- **源数据类型**：
  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
  - BIGINT
  - DOUBLE
  - DATE、DATETIME、TIMESTAMP、TIME
  - VARCHAR
  - JSON
- **示例**：

  例如，将BIGINT类型的数据 `20010122000000` 转换成DATE类型，语句如下：

  ```
  SELECT cast(20010122000000 AS date);
  ```

  返回结果如下：

  ```
  +-------------------------------+
  | cast(20010122000000 AS date) |
  +-------------------------------+
  | 2001-01-22                    |
  +-------------------------------+
  ```

- **注意事项**：

- 若源VARCHAR类型或源BIGINT类型的数据不符合时间类型格式，转换后的结果均为NULL。

  例如，将源VARCHAR类型的数据 `a` 转换为TIME类型，语句如下：

  ```
  SELECT cast('a' AS time);
  ```

  由于格式不匹配会返回NULL，结果如下：

  ```
  +-------------------+
  | cast('a' AS time) |
  +-------------------+
  | NULL              |
  +-------------------+
  ```

- 若目标数据类型包含日期和时间数据，但源数据缺少对应的信息，那么将根据如下规则自动补充数据：
  - 若源数据中缺失时间信息，转换后的时间默认为 `00:00:00` 。

    例如，将TIMSTAMP类型的数据 `2001-1-22` 转换为TIME类型，语句如下：

    ```
    SELECT cast(timestamp '2001-1-22' AS time);
    ```

    由于缺少时间信息，默认返回 `00:00:00` ，结果如下：

    ```
    +-------------------------------------+
    | cast(timestamp '2001-1-22' AS time) |
    +-------------------------------------+
    | 00:00:00                            |
    +-------------------------------------+
    ```

  - 若源数据中缺失日期信息，默认转换后的日期为执行该查询时该客户端的系统日期。

    例如，将TIME类型的数据 `00:00:00` 转换为DATE类型，语句如下：

    ```
    SELECT cast(time '00:00:00' AS date);
    ```

    由于缺少日期信息，默认执行该查询时的系统日期，结果如下：

    ```
    +--------------------------------+
    | cast(time '00:00:00' AS date) |
    +--------------------------------+
    | 2021-09-14                     |
    +--------------------------------+
    ```

CAST AS VARBINARY

```
cast (expr AS varbinary)
```

- **命令说明**：将源数据 `expr` 转换为VARBINARY类型。
- **源数据类型**：
  - BOOLEAN
  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
  - BIGINT
  - DOUBLE
  - DATE、DATETIME、TIMESTAMP、TIME
  - VARCHAR
  - JSON
- **示例**：

  例如，将源VARCHAR类型的数据 `CHINA` 转换为VARBINARY类型，语句如下：

  ```
  SELECT cast('CHINA' AS varbinary);
  ```

  返回结果如下：

  ```
  +-----------------------------------------------------------+
  | cast('CHINA' AS varbinary)                                |
  +-----------------------------------------------------------+
  | 0x4348494E41                                              |
  +-----------------------------------------------------------+
  ```

CAST AS VARCHAR

```
cast (expr AS varchar)
```

- **命令说明**：将源数据 `expr` 转换为VARCHAR类型。
- **源数据类型**：
  - BOOLEAN

- DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
- BIGINT
- DOUBLE
- DATE、DATETIME、TIMESTAMP、TIME
- VARBINARY

  > **说明** 仅集群版本为3.1.4或以上的AnalyticDB MySQL版，支持通过CAST函数将VARBINARY类型转换为VARCHAR类型。

- ARRAY
- MAP
- JSON

- **示例**：

  将源TIMESTAMP类型的数据 `2001-1-22 00:00:00` 转换为VARCHAR类型，语句如下：

```
SELECT cast(timestamp '2001-1-22 00:00:00' AS varchar);
```

  返回结果如下：

```
+-------------------------------------------------+
| cast(timestamp '2001-1-22 00:00:00' AS varchar) |
+-------------------------------------------------+
| 2001-01-22 00:00:00                             |
+-------------------------------------------------+
```

CAST AS ARRAY

```
cast (expr AS array<element_type>)
```

- **命令说明**：将源数据 `expr` 转换为ARRAY数据，其中ARRAY数据由指定数据类型 `<element_type>` 构成。
- **输入值类型**：
  - `expr`：VARCHAR或JSON类型。
  - `<element_type>`：TINYINT、SMALLINT、INT/INTEGER或FLOAT类型。
- **示例**：

  将源JSON类型的数据 `[1,2,3]` 转换ARRAY数据，其中ARRAY数据由INT类型构成，语句如下：

```
SELECT cast( json '[1,2,3]' AS array<int>);
```

  返回结果如下：

```
+--------------------------------------+
| cast( json '[1,2,3]' AS array<int>) |
+--------------------------------------+
| [1,2,3]                              |
+--------------------------------------+
```

- **注意事项**：若源VARCHAR数据或JSON数据不符合ARRAY格式，转换时会报错。

  例如，将源VARCHAR类型的数据 `{}` 转换ARRAY数据，其中ARRAY数据由FLOAT类型构成，语句如下：

```
SELECT cast('{}' AS array<float>);
```

  由于格式不匹配，会提示如下错误：

```
ERROR 1815 (HY000): [30013, 20210918153721192168188048034532046 62] : Value cannot be cast to array(real)
```

CAST AS MAP

```
cast (expr AS map<element_type_1,element_type_2>)
```

- **命令说明**：将源数据 `expr` 转换为MAP数据，该MAP会将 `<element_type_1>` 类型的数据映射为 `<element_type_2>` 类型。
- **输入值类型**：
  - `expr`：VARCHAR类型。
  - `<element_type_1>`：BOOLEAN、DECIMAL、DOUBLE、FLOAT、BIGINT、INT/INTEGER、SMALLINT、TINYINT、VARCHAR类型。
  - `<element_type_2>`：BOOLEAN、DECIMAL、DOUBLE、FLOAT、BIGINT、INT/INTEGER、SMALLINT、TINYINT、VARCHAR、ARRAY、JSON、MAP类型。
- **示例**：

  将源VARCHAR类型的数据 `{"1":"a"}` 转换为MAP数据，该MAP会将一个VARCHAR类型数据映射为VARCHAR，语句如下：

```
SELECT cast('{"1":"a"}' AS map<varchar,varchar>);
```

  返回结果如下：

```
+-------------------------------------------+
| cast('{"1":"a"}' AS map<varchar,varchar>) |
+-------------------------------------------+
| {"1":"a"}                                 |
+-------------------------------------------+
```

- **注意事项**：若源VARCHAR类型的数据不符合MAP格式，转换时会报错。

  例如，将源VARCHAR类型的数据 `[a,b,c]` 转换为MAP类型，语句如下：

  ```
  SELECT cast('[a,b,c]' AS map<varchar,varchar>);
  ```

  此时会返回如下错误：

  ```
  ERROR 1815 (HY000): [30013, 20210918155625192168188048034532078 33] : Value cannot be cast to map(varchar,varchar)
  ```

CAST AS JSON

```
cast (expr AS json)
```

- **命令说明**：将源数据 `expr` 转换为JSON类型。
- **源数据类型**：
  - BOOLEAN
  - DECIMAL(m,d)、FLOAT、INT/INTEGER、SMALLINT、TINYINT
  - BIGINT
  - DOUBLE
  - VARCHAR
  - ARRAY
- **示例**：
  - 将VARCHAR类型的数据 `{}` 转换为JSON类型，语句如下：

    ```
    SELECT cast('{}' AS json);
    ```

    返回结果如下：

    ```
    +--------------------+
    | cast('{}' AS json) |
    +--------------------+
    | {}                 |
    +--------------------+
    ```

  - 将BIGINT类型的数据 `0` 转换成JSON类型，语句如下：

    ```
    SELECT cast( bigint '0' AS json);
    ```

    返回结果如下：

    ```
    +----------------------------+
    | cast( bigint '0' AS json) |
    +----------------------------+
    | 0                          |
    +----------------------------+
    ```

- **注意事项**：若源VARCHAR类型的数据不符合JSON格式，转换时会报错。

  例如，将VARCHAR类型的数据 `{}}` 转换成JSON类型，语句如下：

  ```
  SELECT cast('{}}' AS json);
  ```

  此时会提示如下错误：

  ```
  ERROR 1815 (HY000): [30014, 20210914173352192168188048034532597 05] : Cannot convert '{}}' to JSON
  ```

5.全文检索

5.1. 创建全文索引

本文介绍了云原生数据仓库AnalyticDB MySQL版在创建表时或在已存在的表中添加全文索引的方法。

版本限制

推荐使用内核版本为3.1.4.17及以上的AnalyticDB MySQL集群。

创建表时添加全文索引列

约束条件

- 全文索引仅支持对一列进行设置，如需对多个列创建全文索引，可通过在多个列上单独创建全文索引实现。
- 全文索引只支持varchar类型的列。

语法结构

```
CREATE TABLE [IF NOT EXISTS] table_name
  ({column_name column_type [column_attributes] [ column_constraints ] [COMMENT 'string']
  | table_constraints}
   [, ... ]  )
   table_attribute
   [partition_options]
   [AS] query_expression
   COMMENT 'string'
column_attributes:
   [DEFAULT default_expr]
   [AUTO_INCREMENT]
column_constraints:
   [{NOT NULL|NULL} ]
   [PRIMARY KEY]
table_constraints:
   [{INDEX|KEY} [index_name] (column_name,...)]
   [PRIMARY KEY [index_name] (column_name,...)]
   [CLUSTERED KEY [index_name] (column_name,...)]
   [ANN index [index_name] (col_name,...)] [indexoption]
   [FULLTEXT [INDEX/KEY] [index_name] (col_name,...)] [indexoption]   --定义全文索引
table_attribute:
   DISTRIBUTED BY HASH(column_name,...) | DISTRIBUTED BY BROADCAST
partition_options:
  PARTITION BY
        {VALUE(column_name) | VALUE(date_format(column_name, ?))}
  LIFECYCLE N
```

示例

示例中title列是全文索引，其他列是普通索引。

```
CREATE TABLE fulltext_test (
    id int,
    title varchar,
    body varchar,
    FULLTEXT INDEX title_idx(title),
    PRIMARY KEY (id)
)
DISTRIBUTE BY HASH(id);
mysql> show index from fulltext_test;
+---------------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+-
--------+---------------+
| Table         | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
Comment | Index_comment |
+---------------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+-
--------+---------------+
| fulltext_test |          0 | PRIMARY    |            1 | id          | A         |           0 |     NULL | NULL   |      | BTREE      |
|               |
| fulltext_test |          1 | body_2_idx |            1 | body        | A         |           0 |     NULL | NULL   |      | BTREE      |
|               |
| fulltext_test |          1 | id_0_idx   |            1 | id          | A         |           0 |     NULL | NULL   |      | BTREE      |
|               |
| fulltext_test |          1 | title_idx  |            1 | title       | A         |           0 |     NULL | NULL   |      | FULLTEXT   |
|               |
+---------------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+-
--------+---------------+
4 rows in set (0.04 sec)
```

为已存在的表添加全文索引列

约束条件

- 全文索引仅支持对一列进行设置，如需对多个列创建全文索引，可通过在多个列上单独创建全文索引实现。
- 全文索引只支持varchar类型的列。
- 在已存在的表中添加的全文索引，仅支持查询新写入的数据，查询已经写入数据表的历史数据不能获得有效结果。如果需要对历史数据生效全文索引，需要使用 `BUILD TABLE `表名` force=true;` 强制建立索引。

语法结构

```
ALTER TABLE ADD FULLTEXT [INDEX/KEY] [index_name] (col_name,...)] [indexoption]
```

示例

1. 创建表。

```
CREATE TABLE fulltext_test (
    id int,
    title varchar,
    body varchar,
    FULLTEXT INDEX title_idx(title),
    PRIMARY KEY (id)
)
DISTRIBUTE BY HASH(id)
INDEX_ALL='N';
```

2. 在上表中通过下列语句添加全文索引。

> **说明** 添加全文索引的列不能有普通索引，如果有需要先drop普通索引。

```
ALTER TABLE fulltext_test ADD FULLTEXT INDEX b_idx(body);
```

3. 如需对历史数据建立全文索引，执行下列语句。

```
BUILD TABLE `fulltext_test` force=true;
```

删除全文索引

语法结构

```
ALTER TABLE table_name DROP FULLTEXT INDEX index_name;
```

示例

```
ALTER TABLE fulltext_test DROP FULLTEXT INDEX b_idx;
```

5.2. 通过全文索引查询

本文介绍通过全文检索进行查询的具体方法。

前提条件

已创建包含全文索引列的表，本文以创建fulltext_test表为例说明，包含两个全文索引：t_idx，b_idx。

```
CREATE TABLE fulltext_test (
    id int,
    title varchar,
    body varchar,
    FULLTEXT INDEX t_idx(title),
    FULLTEXT INDEX b_idx(body),
    PRIMARY KEY (id)
)
DISTRIBUTE BY HASH(id);
```

基本查询

```
SELECT *
FROM fulltext_test
WHERE MATCH (title) AGAINST ('杭州');
```

参数说明：

- `match(column_name[, ...])`：在建表语句中创建了全文索引的列名，可以填写一个或多个列名。例如“match(body)”表示只在“body”这一列中进行检索，“match(title, body)”表示在“title”、“body”两列中分别进行检索。
- `against('words')`：进行检索的关键词，例如against（'浙江省杭州市'）表示要检索浙江省杭州市。

多列查询

```
SELECT *
FROM fulltext_test
WHERE MATCH (title, body) AGAINST ('杭州');
```

在匹配多列时，不需要创建多列联合索引，只要多列条件中的每一列具有全文索引，即可进行多列联合全文检索查询，等价于以下查询语句：

```
SELECT *
FROM fulltext_test
WHERE MATCH (title) AGAINST ('杭州')
    OR MATCH (body) AGAINST ('杭州');
```

布尔查询（使用逻辑操作符）

查询时支持如下逻辑操作符，操作符不区分大小写。

- AND：表示操作符两边的关键词都必须出现
- OR：表示操作符两边的关键词出现一个即可
- NOT：表示右侧的关键词不能出现

查询语句：

```
SELECT *
FROM fulltext_test
WHERE MATCH (title) AGAINST ('杭州 AND 浙江');
SELECT *
FROM fulltext_test
WHERE MATCH (title) AGAINST ('杭州 OR 浙江');
```

可以使用括号构造更复杂的布尔查询：

```
SELECT *
FROM fulltext_test
WHERE MATCH (title) AGAINST ('(杭州 OR 浙江) and (教育 or 医疗)');
```

结果集过滤

全文检索会返回所有跟关键词近似的结果。在某些数据量很大的场景中，命中关键词的结果集可能也很大，但是往往只需要取出近似度较高的部分结果，因此AnalyticDB MySQL版提供了结果集过滤的功能。

如下示例中 `where match() against() > 0.9` 表示只取近似度排在前10%的结果，过滤掉了90%的低近似度结果：

```
SELECT *
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9;
```

近似度分数查询/按近似度排序

AnalyticDB MySQL版支持获取结果集的近似度分数，以及按照近似度分数排序的功能。

获取近似度分数的查询语句：

```
SELECT *, MATCH (title) AGAINST ('浙江省杭州市') AS score
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9;
```

默认情况下，返回的结果集将不会按照近似度分数排序，如果需要将结果按照近似度从高到低排序，则需要加上ORDER BY DESC。

按照近似度从高到低排序的查询语句：

```
SELECT *, MATCH (title) AGAINST ('浙江省杭州市') AS score
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9
ORDER BY score DESC;
```

投影中的match(title) against ('浙江省杭州市') 不必和后续查询中的match against语句保持一致，例如，可以通过以下的语句获取match(body) against ('中国') 的近似度分数：

```
SELECT *, MATCH (body) AGAINST ('中国') AS score
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9
ORDER BY score DESC;
```

短语查询/精确匹配

默认情况下，AnalyticDB MySQL版 3.0全文检索会对词语进行分词后，再进行检索。比如“中华人民共和国”被分词为“中华”、“人民”、“共和国”三个词分别检索。在某些特殊情况下，可能需要完全精确的短语匹配，比如只想返回完全精确匹配“中华人民共和国”的结果，而不希望返回分词后的检索结果，则使用短语查询语法。

> 注意
> - 基本查询：分词后再检索，关键词没有双引号。
> - 短语查询：精确匹配，不会做分词处理，关键词有双引号。

基本查询语句：

```
SELECT *, MATCH (title) AGAINST ('中华人民共和国') AS score
FROM fulltext_test
WHERE MATCH (title) AGAINST ('中华人民共和国') > 0.9
ORDER BY score DESC;
```

短语查询语句：

```
SELECT *, MATCH (title) AGAINST ('"中华人民共和国"') AS score
FROM fulltext_test
WHERE MATCH (title) AGAINST ('"中华人民共和国"') > 0.9
ORDER BY score DESC;
```

高亮支持

使用函数高亮

AnalyticDB MySQL版 3.0全文检索功能可以对查询的结果进行高亮，使用函数fulltext_highlight(column)实现。查询语句如下：

```
SELECT MATCH (title) AGAINST ('浙江省杭州市') AS score, fulltext_highlight(title)
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9
ORDER BY score DESC
LIMIT 5;
```

执行结果：

```
score              | fulltext_highlight(title)                                                                                |
+--------------------+------------------------------------------------------------------------------------------+
| 15.57516860961914  | <em>浙江省</em>、<em>杭州市</em>网信办联合约谈“二更食堂”公众号负责人                                                                  |
| 11.763599395751953 | <em>浙江省</em>兰溪<em>市</em>华宇方圆项目正式签约                                                                          |
| 11.269683837890625 | <em>浙江省</em>大学前五名，你知道是那些大学吗?                                                                              |
| 11.153276443481445 | <em>浙江省</em>的985、211大学只有浙大，是否可以说<em>浙江省</em>的大学教育质量差?                                                        |
| 10.928305625915527 | <em>杭州市</em>最适合情侣去的地方
|
```

使用hint自定义高亮

查询语句如下：

```
/*fulltext_highlight_pre_tag=<span>,fulltext_highlight_post_tag=</span>*/
SELECT MATCH (title) AGAINST ('浙江省杭州市') AS score, fulltext_highlight(title)
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9
ORDER BY score DESC
LIMIT 5;
```

如果您自定义的tag中包含等于号或者逗号，需要将整个tag值用中括号包裹，示例如下：

```
/*fulltext_highlight_pre_tag=[<h3 style="color:blue">],fulltext_highlight_post_tag=</h3>*/
SELECT MATCH (title) AGAINST ('浙江省杭州市') AS score, fulltext_highlight(title)
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9
ORDER BY score DESC
LIMIT 5;
```

执行结果：

```
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------
-----+
| score              | fulltext_highlight(title)
|
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------
-----+
| 15.867133140563965 | <h3 style="color:blue">浙江省</h3>、<h3 style="color:blue">杭州市</h3>网信办联合约谈“二更食堂”公众号负责人                                                                  |
| 12.205625534057617 | <h3 style="color:blue">浙江省</h3>兰溪<h3 style="color:blue">市</h3>华宇方圆项目正式签约
|
| 11.646674156188965 | <h3 style="color:blue">浙江省</h3>的985、211大学只有浙大，是否可以说<h3 style="color:blue">浙江省</h3>的大学教育质量差?  |
| 11.338353157043457 | <h3 style="color:blue">浙江省</h3>大学前五名，你知道是那些大学吗?
|
| 11.2699556350708   | <h3 style="color:blue">杭州</h3>GDP一万亿，宁波GDP八千五百亿，那为什么<h3 style="color:blue">浙江省</h3>经济中心在宁波?  |
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------
-----+
```

同一列有多个全文索引时高亮

如果您针对同一个列有多个全文索引，如下查询语句：

```
SELECT MATCH (title) AGAINST ('浙江省杭州市') AS score, fulltext_highlight(title)
FROM fulltext_test
WHERE MATCH (title) AGAINST ('浙江省杭州市') > 0.9
    AND MATCH (title) AGAINST ('中国') > 0.9
ORDER BY score DESC
LIMIT 5;
```

那么最终将同时对“浙江省杭州市”和“中国”进行高亮，结果为：

```
+--------------------+-----------------------------------------------+
| score              | fulltext_highlight(title)                     |
+--------------------+-----------------------------------------------+
| 4.041414260864258  | <em>杭州</em>有哪些<em>中国</em>之最?         |
| 3.8747754096984863 | <em>中国</em><em>杭州</em>旅游区              |
| 3.7213351726531982 | 走遍<em>中国</em>-<em>杭州</em>印象           |
| 3.7213351726531982 | 「<em>杭州</em>篇」<em>中国</em>旅途美食地图  |
| 3.641066789627075  | 湘西在<em>中国</em>的哪个<em>省</em>?         |
+--------------------+-----------------------------------------------+
```

不支持的语法

全文索引过滤条件match() against()不能放在子查询外作为子查询的过滤条件，例如：

```
SELECT *
FROM (
    SELECT id, substr(text, 4) AS text
    FROM tbl
) A
WHERE MATCH (A.text) AGAINST ('浙江省杭州市');
```

5.3. 自定义分词器和自定义词典

本文介绍了如何使用全文检索的分词器和自定义词典。

在大部分场景下，默认分词器（AliNLP）可以获得很好的分词效果，不需要人工进行干预。您也可以自行选择使用IK分词器。在一些特殊场景下，您可以使用自定义词典影响分词结果，获取更贴近业务实践场景的分词结果。AnalyticDB MySQL版 3.0提供灵活的自定义词典能力，您可以像操作普通表一样操作自定义词典，并对新写入的数据实时生效。并且，AnalyticDB MySQL版 3.0中的自定义词典可以应用于索引级别，大大提高了自定义词典使用的灵活性。

> **说明** 全文索引的分词器可以支持英语和其他国语言，具体以使用的分词器的效果评估。

自定义分词器

AnalyticDB MySQL版 3.0全文检索功能支持AliNLP分词器和IK分词器，您可以在创建全文索引时按需指定分词器，不指定的情况下系统默认采用ALiNLP分词器。

自定义分词器示例：

```
CREATE TABLE fulltext_test (
    id int,
    title varchar,
    body varchar,
    FULLTEXT INDEX t_idx(title) WITH ANALYZER alinlp,
    FULLTEXT INDEX b_idx(title) WITH ANALYZER ik,
    PRIMARY KEY (id)
)
DISTRIBUTE BY HASH(id);
```

自定义词典

全文检索功能在构建索引和查询时，都需要对文档或待查询语句进行分词，分词结果直接影响索引构建及查询结果。AliNLP在大部分场景下可以得到理想的分词结果，但在一些特定的场景下，分词结果会造成查询结果差异，并不能满足业务实际需求，自定义词典提供的灵活扩充词库能力，可以很好的解决这个问题。

AnalyticDB MySQL版 3.0的全文检索支持自定义词典，您可以对词典表进行增加和删除以达到维护词典的目的，也可以按需创建多张词典表，以达到不同的应用场景使用不同词典的目的。

创建自定义词典表

AnalyticDB MySQL版 3.0中，具体的创建语法如下：

```
CREATE TABLE `ext_dict` (
  `value` varchar(255) NOT NULL COMMENT '扩展词/停止词值',
  `type` varchar(4) NOT NULL DEFAULT 'main' COMMENT 'main表示扩展词, stop表示停止词(暂不支持停止词)',
  PRIMARY KEY (`value`,`type`)
) COMMENT='用户词典表'
FULLTEXT_DICT = 'Y';
```

约束：

1. 词典只能有两个字段：value和type；
2. value和type字段都为varchar类型，并且不能为空；

3. 词典表主键必须同时包含value和type字段；

字段解释：

1. value字段表示具体的词条内容；
2. type字段用来标记词条类型，比如扩展词和停止词，在目前的应用场景中，只有扩展词（main）是合法的，其他类型的词条可以写入，但不会生效。

AnalyticDB MySQL版对可以创建的全文词典的数量进行了限制。目前一个逻辑库可以创建一个全文词典表，一个物理库中可以创建最多十个全文词典表。

更新自定义词典

词典本身是一张表，读写操作与普通表无异，具体约束为：

1. 词典表不允许执行Online DDL；
2. 不支持update和truncate；
3. 不支持禁用索引；
4. 不支持使用全文索引、自定义词典、自定义分词器；
5. 词典表在被其他索引使用时禁止删除。

插入一个词条：

```
INSERT INTO ext_dict (`value`) VALUES ('China');
```

删除一个词条：

```
DELETE FROM ext_dict WHERE `value` = 'China' AND `type` = 'main';
```

一个词典默认最多允许插入1w条记录，如果需要调整插入记录数，请提交工单。

您插入或者删除词典中的词条后，词典会立刻生效，对于新写入数据表的数据，将使用自定义词典最新的词条进行分词。

使用自定义词典

创建全文索引时，可以指定全文索引使用的自定义词典，以在线创建全文索引为例：

```
ALTER TABLE `test` ADD FULLTEXT INDEX t_idx(`title`) WITH DICT [logical_schema.]table;
```

使用with dict指定使用的自定义词典，指定的自定义词典必须满足以下条件：

1. 是一个自定义词典表；
2. 当前用户具备对应自定义词典表的SELECT权限。

自定义词典中的单个词条如果是中英文混合，例如车牌号前缀 “浙A”，在指定自定义词典时需要同时指定使用AliNLP分词器，例如：

```
ALTER TABLE `test` ADD FULLTEXT INDEX t_idx(`title`) WITH DICT ext_dict WITH ANALYZER alinlp;
```

全文索引可见策略

由于实时构建全文索引的开销比较大，对于实时写入的数据，AnalyticDB MySQL版 3.0采用秒级可见的索引构建策略。如果需要根据业务需求调整索引刷新策略，可联系AnalyticDB MySQL版官方支持进行配置。

6.查询流量控制

云原生数据仓库AnalyticDB MySQL版支持为内部系统查询、用户普通查询、用户ETL（Extract-Transform-Load）类查询三种查询队列设置最大可运行查询数以及最大排队查询数。

查询队列

为了隔离内部系统查询、用户普通查询、用户ETL（Extract-Transform-Load）类查询（例如INSERT INTO SELECT等）三种查询流量，云原生数据仓库AnalyticDB MySQL版接入层将这三种查询的查询队列进行了隔离。

每个查询队列都可设置最大可运行查询数和最大排队查询数：

- 当查询队列里正在执行的查询数量≥最大可运行查询数时，新的查询将进入排队状态；
- 当查询队列里排队状态的查询数量≥最大排队查询数时，新的查询将直接被拒绝；
- 当正在执行的查询结束时，如果有排队状态的查询，会以FIFO的方式，从排队状态的查询中取下一个查询进入执行状态。

> **说明** 查询队列对 `INSERT INTO ... VALUE(...)` 语句不生效。

配置参数

> **说明**
> - 查询队列的参数取值代表单个前端节点的队列大小，不是集群的队列总大小。集群的队列总大小等于单个队列乘以前端节点个数，即 `集群的队列总大小=单个队列×前端节点个数` 。
> - 查询队列的参数配置对全局前端节点同时生效。

查询类型	查询队列	配置参数	配置项
系统查询	ROOT	固定取值，不支持配置。	• 单个前端节点的最大可运行查询数：400 • 单个前端节点的最大排队查询数：500
用户普通查询	NORMAL	XIHE_ENV_QUERY_MAX_CONCURRENT_QUERIES	针对用户普通查询，单个前端节点的最大可运行查询数，参数取值范围为1～20。 假设参数取值为20，集群包含3个前端节点，则每个前端节点的最大可运行查询数为20，集群总的最大可运行查询数为60。
		XIHE_ENV_QUERY_MAX_QUEUED_QUERIES	针对用户普通查询，单个前端节点的最大排队查询数，参数取值范围为1～200。 假设参数取值为200，集群包含3个前端节点，则每个前端节点的最大排队查询数为200，集群总的最大排队查询数为600。
用户ETL查询	LOW	XIHE_ENV_QUERY_ETL_MAX_CONCURRENT_SIZE	针对用户ETL查询，单个前端节点的最大可运行查询数，参数取值范围为1～20。 假设参数取值为10，集群包含3个前端节点，则每个前端节点的最大可运行查询数为10，集群总的最大可运行查询数为30。
		XIHE_ENV_QUERY_ETL_MAX_QUEUED_SIZE	针对用户ETL查询，单个前端节点的最大排队查询数，参数取值范围为1～100。 假设参数取值为100，集群包含3个前端节点，则每个前端节点的最大排队查询数为100，集群总的最大排队查询数为300。

调整查询队列

您可以使用下方的语句，配置单个前端节点的查询队列大小。

> **说明**
> - 查询队列的参数取值代表单个前端节点的队列大小，不是集群的队列总大小。集群的队列总大小等于单个队列乘以前端节点个数，即 `集群的队列总大小=单个队列×前端节点个数` 。
> - 查询队列的参数配置对全局前端节点同时生效。

```
--用户普通查询队列
set adb_config XIHE_ENV_QUERY_MAX_CONCURRENT_QUERIES=20
set adb_config XIHE_ENV_QUERY_MAX_QUEUED_QUERIES=200
--用户ETL查询
set adb_config XIHE_ENV_QUERY_ETL_MAX_CONCURRENT_SIZE=20
set adb_config XIHE_ENV_QUERY_ETL_MAX_QUEUED_SIZE=100
```

上述示例，针对用户普通查询，设置单个前端节点的最大可运行查询数为20，最大排队查询数为200；针对用户ETL查询，设置单个前端节点的最大可运行查询数为20，最大排队查询数为100。

指定查询队列

```
-- 通过Hint指定查询使用LOW队列
/*+coordinator_query_queue=low_priority*/ select * from tbl limit 100;
-- 普通查询使用NORMAL队列
select * from tbl limit 100;
-- ETL查询默认使用LOW队列
insert into dst select * from tbl;
```

执行优先级

云原生数据仓库AnalyticDB MySQL版的计算模块在执行查询时，为了隔离用户普通查询、用户ETL查询，将查询执行线程也进行了隔离。查询执行线程分为两组NORMAL和LOW：

- 内部系统查询、用户普通查询由NORMAL线程组执行。
- 用户ETL查询由LOW线程组执行。

NORMAL线程组相比LOW线程组具有更高的优先级，二者产生资源竞争时，NORMAL线程优先获得CPU资源。

查询类型	执行线程组	优先级
系统查询、普通用户查询	NORMAL	高
用户ETL查询	LOW	低

设置执行优先级

```
-- 通过Hint指定查询使用LOW线程组
/*+direct_low_priority_cpu_queue=true*/ select * from tbl limit 100;
-- 普通查询使用NORMAL线程组
select * from tbl limit 100;
-- ETL查询默认使用LOW线程组
insert into dst select * from tbl;
```

7.常见配置参数

本文为您介绍云原生数据仓库AnalyticDB MySQL版的常见配置参数。

类别	参数	描述	示例	文档链接
新旧集群的切换时间	REPLICATION_SWITCH_TIME_RANGE	新旧集群切换期间，旧集群会有5～10分钟的时间仅支持只读操作。您可以在连接旧集群后，配置REPLICATION_SWITCH_TIME_RANGE来指定切换新旧集群的时间窗口。 说明 如果不配置新旧集群切换的时间窗口，旧集群中的增量数据全部实时同步到新集群后，新旧集群会自动切换。	`SET adb_config REPLICATION_SWITCH_TIME_RANGE=`23:00, 23:30`;`	变更集群配置
IN条件数限制	MAX_IN_ITEMS_COUNT	设置IN条件的个数限制，默认值：2000。	`SET adb_config MAX_IN_ITEMS_COUNT=3000;`	开发类FAQ
查询超时时间	QUERY_TIMEOUT	集群级别为所有查询配置查询的超时时间。单位：毫秒（ms）。	`SET adb_config QUERY_TIMEOUT=1000;`	
		查询级别为单次查询配置查询的超时时间。单位：毫秒（ms）。	`/*+ QUERY_TIMEOUT=1000 */select count(*) from t;`	
INSERT、UPDATE、DELETE超时时间	INSERT_SELECT_TIMEOUT	集群级别修改INSERT、UPDATE和DELETE语句的最大执行时间，默认值：24*3600000。单位：毫秒（ms）。	`SET adb_config INSERT_SELECT_TIMEOUT=3600000;`	约束和限制
		查询级别修改INSERT、UPDATE和DELETE语句的最大执行时间，默认值：24*3600000。单位：毫秒（ms）。	`/*+ INSERT_SELECT_TIMEOUT=3600000 */update customer set customer_name ='adb' where customer_id ='2369';`	
过滤条件不下推	• 内核版本为3.1.4及以上：FILTER_NOT_PUSHDOWN_COLUMNS • 内核版本为3.1.4以下：NO_INDEX_COLUMNS	集群级别关闭特定字段的过滤条件下推能力。	• 内核版本为3.1.4及以上：`SET adb_config FILTER_NOT_PUSHDOWN_COLUMNS=[${database}.${tableName}:${collName}|${col2Name}];` • 内核版本为3.1.4以下：`set adb_config NO_INDEX_COLUMNS=[${tableName}.${collName};${col2Name},${tableName1}.${collName}]`	过滤条件不下推
		查询级别关闭特定字段的过滤条件下推能力。	• 内核版本为3.1.4及以上：`/*+ FILTER_NOT_PUSHDOWN_COLUMNS=[${database}.${tableName}:${collName}|${col2Name}] */` • 内核版本为3.1.4以下：`/*+ NO_INDEX_COLUMNS=[${tableName}.${collName};${col2Name},${tableName1}.${collName}] */`	
查询执行模式	QUERY_TYPE	切换实例的执行模式，取值范围： • interactive • batch 说明 不同系列的集群支持的执行模式不同。	`SET adb_config QUERY_TYPE=interactive;`	查询执行模式（数仓版）
查询队列	XIHE_ENV_QUERY_MAX_CONCURRENT_QUERIES	针对用户普通查询，单个前端节点的最大可运行查询数。取值范围：1～20，默认值为20。	`SET adb_config XIHE_ENV_QUERY_MAX_CONCURRENT_QUERIES=20;`	
	XIHE_ENV_QUERY_MAX_QUEUED_QUERIES	针对用户普通查询，单个前端节点的最大排队查询数。取值范围：1～200，默认值为200。	`SET adb_config XIHE_ENV_QUERY_MAX_QUEUED_QUERIES=20;`	
	XIHE_ENV_QUERY_ETL_MAX_CONCURRENT_SIZE	针对用户ETL查询，单个前端节点的最大可运行查询数。取值范围：1～20。	`SET adb_config XIHE_ENV_QUERY_ETL_MAX_CONCURRENT_SIZE=20;`	

类别	参数	描述	示例	文档链接
	XIHE_ENV_QUERY_ETL_MAX_QUEUED_SIZE	针对用户ETL查询，单个前端节点的最大排队查询数。取值范围：1～100。	`SET adb_config XIHE_ENV_QUERY_ETL_MAX_QUEUED_SIZE =100;`	资源控制
	COORDINATOR_QUERY_QUEUE	查询级别配置查询使用的队列，取值范围： • low_priority • etl	`/*+COORDINATOR_QUERY_QUEUE=low_priority*/ select * from tableName limit 100;`	
执行优先级	DIRECT_LOW_PRIORITY_CPU_QUEUE	查询级别配置执行优先级。	`/*+DIRECT_LOW_PRIORITY_CPU_QUEUE =true*/ select * from tableName limit 100;`	

8.数据存储冷热分离

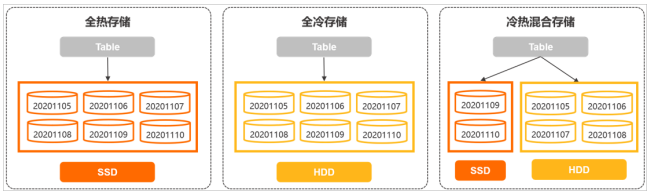
根据业务的使用情况，AnalyticDB MySQL版弹性模式集群版（新版）（3.1.3.3及以上版本）支持表或分区级别的数据存储冷热分离策略。

冷热数据的定义

冷数据指的是访问频次较低的数据，采用低价的HDD存储，满足存储空间的需求。

热数据指的是访问频次较高的数据，采用SSD存储，满足高性能访问的需求。

您可以执行CREATE TABLE语句指定表的冷热存储策略为：全热存储（数据全部存储在SSD）、全冷存储（数据全部存储在HDD）、冷热混合存储（指定一定数量的分区存储在SSD，其余数据存储在HDD）。更多信息可参见CREATE TABLE。



冷热数据的迁移

数据入库开始是在RealTime引擎，经过build会应用冷热存储策略，将数据分为冷、热两部分。其中冷数据放在HDD存储，热数据放在SSD存储。

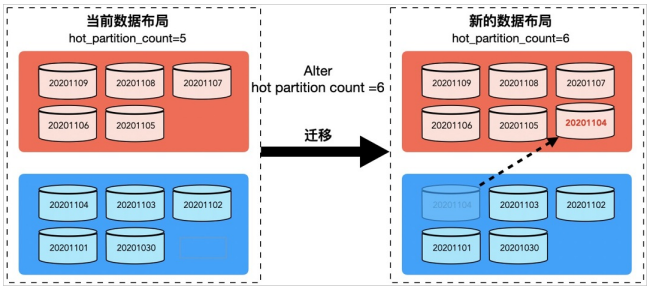
冷热分区布局会按照分区值N的大小降序排列，最大的N个分区为热分区，存储在SSD，其余分区为冷分区，存储在HDD。N的值可以在指定冷热存储策略时定义，更多信息可参见CREATE TABLE。

- 当数据有新增，修改，删除时，会重新调整冷热分区布局。
- 当调整冷热策略时，会重新调整冷热分区布局。您可以执行ALTER TABLE语句修改表的冷热存储策略，更多信息可参见ALTER TABLE。

冷到热的迁移

当前热分区数为N，修改热分区的数量为M，当N<M时，会从冷分区迁移M-N个分区数据到热分区。

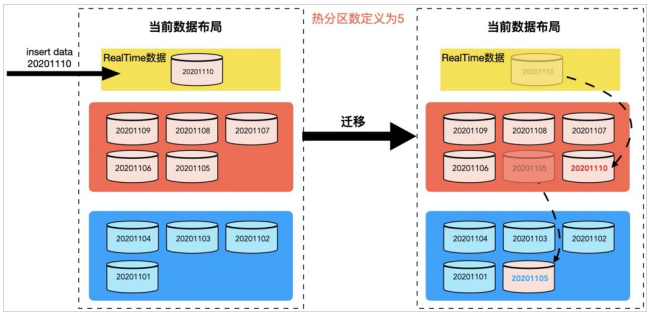
如图所示，当前热分区数为5， 修改热分区数为6后，5<6，所以需要从冷分区迁移一个最大的分区，即20201104到热分区中。



热到冷的迁移

当有新的分区数据写入，对所有分区排序，超过N的旧数据会迁移到冷分区；当前热分区数为N，修改热分区的数量为M，当N>M时，会从热分区迁移N-M个分区数据到冷分区。

如图所示，当新增分区20201110之后，20201110为当前最大的分区，应该放在热分区中，但是当前热分区数已满5，所以需要从热分区中选一个最小的分区20201105迁移到冷分区，并把20201110放在热分区中。



冷热数据存储诊断表

AnalyticDB MySQL版弹性模式集群版（3.1.3.5及以上版本）支持数据的冷热分离存储，用户可以通过查表的方式查询某一张表的冷热数据存储布局情况。如果执行了冷热策略变更语句，也可以通过查表的方式查询当前冷热变更的进度。

查询冷热存储布局

您可以通过查询表 `table_usage` ，来查看当前的冷热数据存储情况。具体使用方式如下：

- 查询所有表的存储状态：

```
select * from information_schema.table_usage
```

- 查询单个表的存储状态：

```
select * from information_schema.table_usage where table_schema='$schema_name' and table_name='$table_name'
```

table_usage表字段信息：

字段名	字段含义描述
table_schema	数据库名
table_name	表名
storage_policy	当前存储策略 • HOT：全热表 • COLD：全冷表 • MIXED：混合表
hot_partition_count	当前存储的热分区数量
cold_partition_count	当前存储的冷分区数
rt_total_size	实时数据总大小（单位：Byte），是rt_data_size和rt_index_size的总和
rt_data_size	实时数据大小（单位：Byte）
rt_index_size	实时数据的主键大小（单位：Byte）
hot_total_size	热分区总数据量（单位：Byte），是hot_data_size和hot_index_size的总和
hot_data_size	热分区数据大小（单位：Byte）
hot_index_size	热分区数据的主键和索引大小（单位：Byte）
cold_total_size	冷分区总数据量（单位：Byte），是cold_data_size和cold_index_size的总和
cold_data_size	冷分区数据大小（单位：Byte）
cold_index_size	冷分区数据的主键和索引大小（单位：Byte）

table_usage表说明：

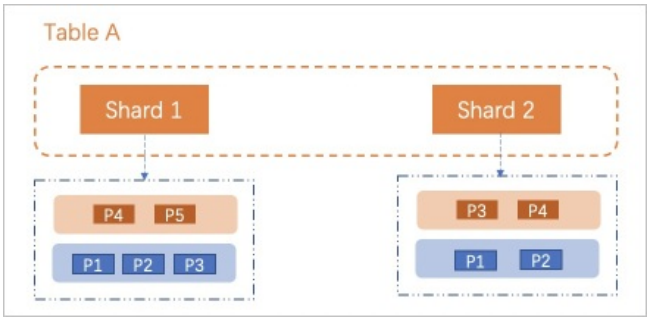
- 如果加载数据之后hot_total_size和cold_total_size都为0，则表示数据还在realtime中，rt_total_size为实际数据的存储，可以通过执行build 指令，将realtime数据转换为分区数据，待build完成后可以看到hot_total_size和cold_total_size的显示。

  build指令：

  ```
  build table $table_name
  ```

- 由于用户定义的hot_partition_count是单个shard内二级分区的热分区存储情况。而这里的hot_partition_count是按照shard union之后的结果，在各shard内分区数据不一致的情况下，可能会出现实际显示的hot_partition_count大于用户定义的hot_partition_count。

  例如：tableA有两个shard（shard1和shard2），并且定义了hot_partition_count=2，此时shard内的数据分布情况如下图。

  shard1：热分区数据为P4，P5，冷分区数据为P1，P2，P3

  shard2：热分区数据为P3，P4，冷分区数据为P1，P2

  则最终计算的实际热分区数为（P4，P5）Union（P3，P4）=（P3，P4，P5），因此实际hot_partition_count=3。

- table_usage是实时更新的，随着insert/update/delete/build的执行，rt_total_size、rt_data_size、rt_index_size、hot_total_size、hot_data_size、hot_index_size、cold_total_size、cold_data_size、cold_index_size会实时变动。

查询冷热变更进度

用户可以通过执行 `ALTER TABLE` 语句修改表的冷热存储策略（详细信息请参见ALTER TABLE）。您可以通过查询表 `storage_policy_modify_progress` 来查看冷热变更进度：

- 查询当前集群中所有参与变更的表的冷热变更进度：

  ```
  select * from information_schema.storage_policy_modify_progress
  ```

- 查询某个表的冷热变更进度：

```
select * from information_schema.storage_policy_modify_progress where table_schema='$schema_name' and table_name='$table_name'
```

`storage_policy_modify_progress` 表字段信息：

字段名	字段含义描述
table_schema	数据库名
table_name	表名
task_id	执行冷热变更任务的ID
source_storage_policy	原存储策略 • HOT：全热表 • COLD：全冷表 • MIXED：混合表
source_hot_partition_count	原热分区数量
dest_storage_policy	目标存储策略 • HOT：全热表 • COLD：全冷表 • MIXED：混合表
dest_hot_partition_count	目标热分区数量
hot_to_cold_partition_count	热到冷变更的分区数量
cold_to_hot_partition_count	冷到热变更的分区数量
hot_to_cold_data_size	热到冷变更的数据量（单位：Byte）
cold_to_hot_data_size	冷到热变更的数据量（单位：Byte）
hot_data_size_before_change	变更前热数据量（单位：Byte）
cold_data_size_before_change	变更前冷数据量（单位：Byte）
hot_data_size_after_change	变更后热数据量（单位：Byte）
cold_data_size_after_change	变更后冷数据量（单位：Byte）
start_time	开始变更时间
update_time	结束变更时间
progress	变更进度（单位：百分比）
status	变更状态 • INIT：还未开始进行变更 • RUNNING：正在进行变更 • FINISH：变更完成

9.API参考

9.1. API概览

云原生数据仓库AnalyticDB MySQL版提供以下API接口。

集群管理

API	描述
CreateDBCluster	调用CreateDBCluster接口创建一个AnalyticDB MySQL版集群。
DescribeDBClusters	调用DescribeDBClusters接口查看集群列表或被RAM授权的集群列表。
ModifyDBCluster	调用ModifyDBCluster接口对AnalyticDB MySQL版集群进行升降配。
DeleteDBCluster	调用DeleteDBCluster接口删除AnalyticDB MySQL版集群。
DescribeDBClusterAttribute	调用DescribeDBClusterAttribute接口查看实例的详细信息。
ModifyDBClusterMaintainTime	调用ModifyDBClusterMaintainTime接口修改实例可维护时间段。
DescribeAutoRenewAttribute	调用DescribeAutoRenewAttribute接口查询包年包月集群自动续费状态。
ModifyAutoRenewAttribute	调用ModifyAutoRenewAttribute接口设置包年包月集群自动续费状态。
ModifyDBClusterDescription	调用ModifyDBClusterDescription接口修改实例的备注名，方便实例的维护管理。
DescribeAvailableResource	调用DescribeAvailableResource接口查询指定可用区资源。
ModifyDBClusterResourceGroup	调用ModifyDBClusterResourceGroup接口将AnalyticDB实例移动到指定资源组。

资源池管理

API	描述
CreateDBResourcePool	调用CreateDBResourcePool创建资源池。该接口仅适用于ADB弹性模式集群版（新版）。
DeleteDBResourcePool	调用DeleteDBResourcePool接口删除资源池。该接口仅适用于ADB弹性模式集群版（新版）。
DescribeDBResourcePool	调用DescribeDBResourcePool接口查询集群的资源池信息。该接口仅适用于ADB弹性模式集群版（新版）。
ModifyDBResourcePool	调用ModifyDBResourcePool接口更新资源池资源。该接口仅适用于ADB弹性模式集群版（新版）。
BindDBResourcePoolWithUser	调用BindDBResourcePoolWithUser接口将资源池与数据库用户进行绑定。该接口仅适用于ADB弹性模式集群版（新版）。
UnbindDBResourcePoolWithUser	调用UnbindDBResourcePoolWithUser接口解绑资源池用户。该接口仅适用于ADB弹性模式集群版（新版）。

弹性计划管理

API	描述
CreateElasticPlan	调用CreateElasticPlan创建分时弹性计划。该接口仅适用于ADB弹性模式集群版（新版）。
DeleteElasticPlan	调用DeleteElasticPlan接口删除分时弹性计划。该接口仅适用于ADB弹性模式集群版（新版）。

API	描述
DescribeElasticDailyPlan	调用DescribeElasticDailyPlan接口查询分时弹性计划日计划执行信息。该接口仅适用于ADB弹性模式集群版（新版）。
DescribeElasticPlan	调用DescribeElasticPlan接口查询弹性计划。该接口仅适用于ADB弹性模式集群版（新版）。
ModifyElasticPlan	调用ModifyElasticPlan修改分时弹性计划。该接口仅适用于ADB弹性模式集群版（新版）。

数据库

API	描述
DescribeTables	调用DescribeTables接口枚举实例指定数据库下所有表列表。
DescribeAllDataSource	调用DescribeAllDataSource接口枚举实例下所有数据库列表、表列表和列列表。
DescribeSchemas	调用DescribeSchemas接口枚举实例下所有数据库列表。
DescribeTableDetail	调用DescribeTableDetail接口表详情，查看分区偏离度。
DescribeProcessList	调用DescribeProcessList接口查看实例正在运行的查询。
DescribeColumns	调用DescribeColumns接口枚举实例下指定表格的列列表。
DescribeTaskInfo	调用DescribeTaskInfo接口查看相关管控任务流进度。
DescribeSQLPlan	调用DescribeSQLPlan接口查询基础信息和计划信息。
DescribeSQLPlanTask	调用DescribeSQLPlanTask查询任务信息。
KillProcess	调用KillProcess接口终止正在进行的任务。
DescribeTablePartitionDiagnose	调用DescribeTablePartitionDiagnose查询二级分区诊断新系列。

日志管理

API	描述
DescribeSlowLogRecord	调用DescribeSlowLogRecords接口查看集群慢日志明细。
DescribeSlowLogTrend	调用DescribeSlowLogTrend接口查看集群慢日志趋势统计情况。
DescribeAuditLogConfig	调用DescribeAuditLogConfig接口查询ADB实例审计日志设置。
ModifyAuditLogConfig	调用ModifyAuditLogConfig接口修改审计日志设置。
DescribeAuditLogRecords	调用DescribeAuditLogRecords接口查看ADB实例的SQL审计日志。

地域管理

API	描述
DescribeRegions	调用DescribeRegions接口查询AnalyticDB for MySQL可选的地域和可用区。

网络管理

API	描述
AllocateClusterPublicConnection	调用AllocateClusterPublicConnection接口为集群申请公网连接地址。
DescribeDBClusterNetInfo	调用DescribeDBClusterNetInfo接口查询集群的网络信息。
ReleaseClusterPublicConnection	调用ReleaseClusterPublicConnection接口释放集群的公网连接地址。
ModifyClusterConnectionString	调用ModifyClusterConnectionString接口修改实例的网络连接地址。

账号管理

API	描述
CreateAccount	调用CreateAccount接口为集群创建账号。
DeleteAccount	调用DeleteAccount接口删除数据库账号。
DescribeAccounts	调用DescribeAccounts接口查询集群的账号信息。
DescribeOperatorPermission	调用DescribeOperatorPermission接口查询集群服务账号的授权详情。
GrantOperatorPermission	调用GrantOperatorPermission接口为集群服务账号授权。
ResetAccountPassword	调用ResetAccountPassword接口重置数据库账号。
RevokeOperatorPermission	调用RevokeOperatorPermission接口撤销集群服务账号权限。
ModifyAccountDescription	调用ModifyAccountDescription接口修改ADB数据库账号的备注信息。
DescribeAllAccounts	调用DescribeAllAccounts接口查询指定集群、指定数据库的账号列表信息或某个指定账号的信息。

标签管理

API	描述
TagResources	调用TagResources接口为AnalyticDB MySQL版集群绑定标签。
ListTagResources	调用ListTagResources接口查询一个或多个AnalyticDB MySQL版集群已绑定的标签列表，或者查询一个或多个标签绑定的AnalyticDB MySQL版集群列表。
UntagResources	调用UntagResources接口将标签从AnalyticDB MySQL版集群中解绑，如果该标签没有绑定到其他集群，则该标签会被删除。

备份恢复

API	描述
DescribeBackups	调用DescribeBackups接口查看集群的备份列表。
DescribeBackupPolicy	调用DescribeBackupPolicy接口查看集群备份设置。
ModifyBackupPolicy	调用ModifyBackupPolicy接口修改ADB实例的备份策略。
ModifyLogBackupPolicy	调用ModifyLogBackupPolicy接口修改日志备份设置。

安全管理

API	描述
DescribeDBClusterAccessWhiteList	调用DescribeDBClusterAccessWhiteList接口查看集群的IP白名单。
ModifyDBClusterAccessWhiteList	调用ModifyDBClusterAccessWhiteList接口修改集群白名单。

监控管理

API	描述
DescribeDBClusterPerformance	调用DescribeDBClusterPerformance接口查看目标集群的性能数据。
DescribeDBClusterResourcePoolPerformance	调用DescribeDBClusterResourcePoolPerformance接口查看AnaluticDB MySQL版弹性模式集群版（新版）的资源池监控信息。
DescribeInclinedTables	调用DescribeInclinedTables接口查看表监控。
DescribeTableStatistics	调用DescribeTableStatistics接口查询目标AnalyticDB MySQL版集群中的表信息统计详情。

运维事件

API	描述
DescribeMaintenanceAction	调用DescribeMaintenanceAction接口查询运维事件的详情。
ModifyMaintenanceAction	调用ModifyMaintenanceAction接口修改待处理运维事件的切换时间。

SQL诊断

API	描述
DescribeDiagnosisRecords	调用DescribeDiagnosisRecords接口查看目标AnalyticDB MySQL版集群中符合指定检索条件的SQL语句摘要信息。
DescribeDiagnosisDimensions	调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL在资源组、数据库名、用户名以及访问源地址等维度下的去重统计信息。
DescribeDownloadRecords	调用DescribeDownloadRecords接口查看目标AnalyticDB MySQL版集群中最近5次的SQL查询下载任务列表。
DownloadDiagnosisRecords	调用DownloadDiagnosisRecords接口下载目标AnalyticDB MySQL版集群中符合指定条件的查询SQL的摘要信息。

实例运行报告

API	描述
DescribeTableAccessCount	调用DescribeTableAccessCount接口查看指定日期内AnalyticDB MySQL版集群下目标表或所有表的被访问次数。
DescribeSqlPattern	调用DescribeSqlPattern接口查看指定日期内AnalyticDB MySQL版集群下的SQL Pattern详情。
DescribeDBClusterHealthReport	调用DescribeDBClusterHealthReport接口查看指定日期内目标AnalyticDB MySQL版集群的健康报告主要性能指标（如CPU使用率、存储磁盘IOPS、存储磁盘IO吞吐等）的平均值和最大值。
DescribeDBClusterForecast	调用DescribeDBClusterForecast接口查看指定日期内目标AnalyticDB MySQL版集群的磁盘水位、QPS和TPS的趋势预测信息。

SQL Pattern

API	描述
DescribeSQLPatterns	调用DescribeSQLPatterns接口查看指定日期内AnalyticDB MySQL版集群下的SQL Pattern列表。
DescribeSQLPatternAttribute	调用DescribeSQLPatternAttribute接口查看AnalyticDB MySQL版集群下指定SQL Pattern的统计信息。
DescribePatternPerformance	调用DescribePatternPerformance接口查看目标SQL Pattern的各指标（如查询时间、平均内存消耗等）在不同时间点的详情。

9.2. 请求结构

本文介绍云原生数据仓库AnalyticDB MySQL版（简称AnalyticDB MySQL版，原分析型数据库MySQL版）API的请求结构。

服务地址

AnalyticDB MySQL版的API服务接入地址见下表。

地域（部署位置）	接入地址（Endpoint）
青岛、北京、杭州、上海、深圳、中国香港、新加坡、弗吉尼亚、硅谷	adb.aliyuncs.com
张家口	adb.cn-zhangjiakou.aliyuncs.com
东京	adb.ap-northeast-1.aliyuncs.com
吉隆坡	adb.ap-southeast-3.aliyuncs.com
悉尼	adb.ap-southeast-2.aliyuncs.com
迪拜	adb.me-east-1.aliyuncs.comm
呼和浩特	adb.cn-huhehaote.aliyuncs.com
孟买	adb.ap-south-1.aliyuncs.com
雅加达	adb.ap-southeast-5.aliyuncs.com
法兰克福	adb.eu-central-1.aliyuncs.com
伦敦	adb.eu-west-1.aliyuncs.com
成都	adb.cn-chengdu.aliyuncs.com

通信协议

支持通过HTTP或HTTPS通道进行请求通信。为了获得更高的安全性，推荐您使用HTTPS通道发送请求。

请求方法

支持HTTP GET方法发送请求，这种方式下请求参数需要包含在请求的URL中。

请求参数

每个请求都需要指定要执行的操作，即Action参数（例如CreateDatabase），以及每个操作都需要包含的公共请求参数和指定操作所特有的请求参数。

字符编码

请求及返回结果都使用UTF-8字符集进行编码。

9.3. 签名机制

云原生数据仓库AnalyticDB MySQL版服务会对每个访问的请求进行身份验证，所以无论使用HTTP还是HTTPS协议提交请求，都需要在请求中包含签名（Signature）信息。

背景信息

云原生数据仓库AnalyticDB MySQL版通过使用Access Key ID和Access Key Secret进行对称加密的方法来验证请求的发送者身份。Access Key ID和Access Key Secret由阿里云官方颁发给访问者（可以通过阿里云官方网站申请和管理），其中Access Key ID用于标识访问者的身份；Access Key Secret是用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密，只有阿里云和用户知道。

用户在访问时，按照下面的方法对请求进行签名处理：

操作步骤

1. 使用请求参数构造规范化的请求字符串（Canonicalized Query String）。

i. 按照参数名称的字典顺序对请求中所有的请求参数（包括文档中描述的公共请求参数和给定了的请求接口的自定义参数，但不能包括公共请求参数中提到Signature参数本身）进行排序。

> **说明** 当使用GET方法提交请求时，这些参数就是请求URL中的参数部分（即URL中 `"?"` 之后由 `"&"` 连接的部分）。

ii. 对每个请求参数的名称和值进行编码。名称和值要使用UTF-8字符集进行URL编码，URL编码的编码规则是：

- 对于字符 A~Z、a~z、0~9以及字符“-”、“_”、“.”、“~”不编码。
- 对于其他字符编码成 `"%XY"` 的格式，其中XY是字符对应ASCII码的16进制表示。比如英文的双引号（”）对应的编码就是%22。
- 对于扩展的UTF-8字符，编码成 `"%XY%ZA…"` 的格式。
- 需要说明的是英文空格（ ）要被编码是%20，而不是加号（+）。

> **说明** 一般支持URL编码的库（比如Java中的 `java.net.URLEncoder` ）都是按照 `"application/x-www-form-urlencoded"` 的MIME类型的规则进行编码的。实现时可以直接使用这类方式进行编码，把编码后的字符串中加号（+）替换成%20、星号（*）替换成%2A、%7E替换回波浪号（~），即可得到上述规则描述的编码字符串。

iii. 对编码后的参数名称和值使用英文等号（=）进行连接。

iv. 再把英文等号连接得到的字符串按参数名称的字典顺序依次使用&符号连接，即得到规范化请求字符串。

2. 使用上一步构造的规范化字符串按照下面的规则构造用于计算签名的字符串：

```
StringToSign= HTTPMethod + "&" + percentEncode("/") + "&" + percentEncode(CanonicalizedQueryString)
```

参数说明：

- HTTPMethod：提交请求用的HTTP方法，比如*GET*。
- percentEncode(“/”)：按照1.b中描述的URL编码规则对字符“/”进行编码得到的值，即*“%2F”*
- percentEncode(CanonicalizedQueryString)：对第1步中构造的规范化请求字符串按1.b中描述的URL编码规则编码后得到的字符串。

3. 按照RFC2104的定义，使用上面的用于签名的字符串计算签名HMAC值。

> **说明** 计算签名时使用的Key就是用户持有的Access Key Secret并加上一个 `"&"` 字符（ASCII:38），使用的哈希算法是SHA1。

4. 按照Base64编码规则把上面的HMAC值编码成字符串，即得到签名值（Signature）。
5. 将得到的签名值作为Signature参数添加到请求参数中，即完成对请求签名的过程。

> **说明** 得到的签名值在作为最后的请求参数值提交给DNS服务器的时候，要和其他参数一样，按照RFC3986的规则进行URL编码）。

以DescribeDBClusters为例，签名前的请求URL为：

```
http://adb.aliyuncs.com/?Timestamp=2013-06-01T10:33:56Z&Format=XML&AccessKeyId=testid&Action=DescribeDBClusters&SignatureMethod=HMAC-SHA1&RegionId=region1&SignatureNonce=NwDAxvLU6tFE0DVb&Version=2014-08-15&SignatureVersion=1.0
```

那么StringToSign就是：

```
GET&%2F&AccessKeyId%3Dtestid&Action%3DDescribeDBClusters&Format%3DXML&RegionId%3Dregion1&SignatureMethod%3DHMAC-SHA1&SignatureNonce%3DNwDAxvLU6tFE0DVb&SignatureVersion%3D1.0&Timestamp%3D2013-06-01T10%253A33%253A56Z&Version%3D2014-08-15
```

假如使用的Access Key Id是“testid”，Access Key Secret是“testsecret”，用于计算HMAC的Key就是“testsecret&”，则计算得到的签名值是： `uRpHwaSEt3J+6KQD//svCh/x+pI=` 。

签名后的请求URL为（注意增加了Signature参数）：

```
http://adb.aliyuncs.com/?Timestamp=2013-06-01T10%3A33%3A56Z&Format=XML&AccessKeyId=testid&Action=DescribeDBClusters&SignatureMethod=HMAC-SHA1&RegionId=region1&SignatureNonce=NwDAxvLU6tFE0DVb&SignatureVersion=1.0&Version=2014-08-15&Signature=BIPOMlu8LXBeZtLQkJTw6iFvwlE%3D
```

9.4. 公共参数

公共参数是指所有接口调用都需要用到的参数，包含公共请求参数和公共返回参数。

公共请求参数

公共请求参数是指每个接口都需要使用到的请求参数。

名称	类型	是否必须	描述
Format	String	否	返回值的类型，支持JSON与XML，默认为JSON。
Version	String	是	API版本号，为日期形式：YYYY-MM-DD，本版本对应的版本号为2019-03-15。
AccessKeyId	String	是	阿里云颁发给用户的访问服务所用的密钥ID。

名称	类型	是否必须	描述
Signature	String	是	签名结果串。关于签名的计算方法，请参见签名机制
SignatureMethod	String	是	签名方式，目前仅支持HMAC-SHA1。
Timestamp	String	是	请求的时间戳。日期格式按照ISO8601标准表示，并需要使用UTC时间，格式为YYYY-MM-DDThh:mm:ssZ。例如，2013-08-15T12:00:00Z为北京时间2013年8月15日20点0分0秒。
SignatureVersion	String	是	签名算法版本，目前的版本是1.0。
SignatureNonce	String	是	唯一随机数，用于防止网络重放攻击。在不同请求间要使用不同的随机数值。

9.5. RAM资源授权

描述

您通过云账号创建的云原生数据仓库AnalyticDB MySQL版实例，都是该账号自己拥有的资源。默认情况下，账号对自己的资源拥有完整的操作权限。

通过使用阿里云的访问控制RAM（Resource Access Management）服务，您可以将您云账号下分析型数据库MySQL版资源的访问及管理权限授予RAM中的子用户。

目前，可以在RAM中进行授权的资源类型只有dbinstance，即最细粒度为实例级别。在通过RAM进行授权时，资源的描述方式如下：

请求参数

资源类型	授权策略中的资源描述方式
dbcluster	acs:adb:$regionid:$accountid:dbcluster/acs:adb:::dbcluster/

参数说明：

- $regionid：地域ID，可以用*代替。
- $accountid：云账号的数字ID，可以用*代替。

> 说明 云原生数据仓库AnalyticDB MySQL版仅支持针对账号下所有集群授权，不支持集群级别的授权，即不支持acs:adb:::dbcluster/pc-xxxxxxx。

示例

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "adb:Describe*"
      ],
      "Effect": "Allow",
      "Resource": [
        "acs:adb:cn-hangzhou:12345678901234:dbcluster/*"
      ]
    },
    {
      "Action": "adb:Describe*",
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

9.6. AnalyticDB MySQL服务关联角色

本文为您介绍云原生数据仓库MySQL版（简称ADB MySQL版）服务关联角色（AliyunServiceRoleForAnalyticDBForMySQL）的应用场景以及如何删除服务关联角色。

背景信息

ADB MySQL版服务关联角色（AliyunServiceRoleForAnalyticDBForMySQL）是在某些情况下，为了完成ADB MySQL版自身的某个功能，需要获取其他云服务的访问权限，而提供的RAM角色。更多关于服务关联角色的信息请参见服务关联角色。

AliyunServiceRoleForAnalyticDBForMySQL介绍

角色名称：AliyunServiceRoleForAnalyticDBForMySQL

角色权限策略：AliyunServiceRolePolicyForAnalyticDBForMySQL

权限说明：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:GetProject",
        "log:ListProject",
        "log:GetCursorTime",
        "log:GetLogs",
        "log:GetHistograms",
        "log:GetContextLogs",
        "log:GetLogStoreLogs",
        "log:GetLogStoreHistogram",
        "log:GetLogStore",
        "log:ListLogStores",
        "log:GetConfig",
        "log:ListConfig",
        "log:GetShipperStatus",
        "log:GetCheckPoint",
        "log:HeartBeat",
        "log:UpdateCheckPoint",
        "log:PostLogStoreLogs",
        "log:CreateConsumerGroup",
        "log:UpdateConsumerGroup",
        "log:DeleteConsumerGroup",
        "log:ListConsumerGroup",
        "log:ConsumerGroupUpdateCheckPoint",
        "log:ConsumerGroupHeartBeat",
        "log:GetConsumerGroupCheckPoint",
        "log:CreateExport",
        "log:GetExport",
        "log:ListExport",
        "log:UpdateExport",
        "log:DeleteExport",
        "log:CreateJob",
        "log:GetJob",
        "log:ListJobs",
        "log:UpdateJob",
        "log:DeleteJob",
        "log:GetCursor",
        "log:PullLogs",
        "log:GetCursorOrData",
        "log:ListShards",
        "dts:CreateSynchronizationJob",
        "dts:ConfigureSynchronizationJob",
        "dts:DescribeSynchronizationJobStatus",
        "dts:StartSynchronizationJob",
        "dts:DeleteSynchronizationJob",
        "dts:DescribeSynchronizationJobs",
        "vpc:DescribeVpcAttribute",
        "ecs:CreateSecurityGroup",
        "ecs:AuthorizeSecurityGroup",
        "ecs:AuthorizeSecurityGroupEgress",
        "ecs:DeleteSecurityGroup",
        "ecs:CreateNetworkInterface"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "ads.aliyuncs.com"
        }
      }
    }
  ]
}
```

删除服务关联角色

如果您需要删除AliyunServiceRoleForAnalyticDBForMySQL（服务关联角色），需要先释放依赖这个服务角色的所有集群。

- 删除ADB MySQL版集群具体操作请参见删除集群。
- 删除服务关联角色具体操作请参见删除服务关联角色。

9.7. 资源池管理

9.7.1. CreateDBResourcePool

调用CreateDBResourcePool接口为AnalyticDB MySQL版集群创建资源池。

使用说明

该接口仅适用于AnalyticDB MySQL弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreateDBResourcePool	系统规定参数。取值：**CreateDBResourcePool**。
DBClusterId	String	是	am-bp11q28kvl688****	集群ID。
PoolName	String	是	test	资源池名称，长度不能超过64byte。
QueryType	String	否	interactive	查询类型，取值说明： • **etl**：批处理模式。 • **interactive**：交互查询模式。 • **default_type**：默认查询模式。 默认值为default_type。
NodeNum	Integer	否	2	节点数，默认节点数0，填写的节点数需小于等于资源名称为USER_DEFAULT的节点数。 **说明** 可通过DescribeDBResourcePool接口查看资源名称为USER_DEFAULT的节点数。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=CreateDBResourcePool
&DBClusterId=am-bp11q28kvl688****
&PoolName=test
&QueryType=interactive
&NodeNum=2
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<CreateDBResourcePoolResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
</CreateDBResourcePoolResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.7.2. DeleteDBResourcePool

调用DeleteDBResourcePool接口删除资源池。该接口仅适用于ADB弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteDBResourcePool	系统规定参数，取值：DeleteDBResourcePool。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
PoolName	String	是	realtime	资源池名称。默认资源池USER_DEFAULT不可做删除操作。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteDBResourcePool
&DBClusterId=am-bp1xxxxxxxx47
&PoolName=realtime
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.7.3. DescribeDBResourcePool

调用DescribeDBResourcePool接口查询AnalyticDB MySQL版集群的资源池信息。

使用说明

该接口仅适用于弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDBResourcePool	系统规定参数。取值：**DescribeDBResourcePool**。
DBClusterId	String	是	am-bp11q28kvl688****	集群ID。
PoolName	String	是	USER_DEFAULT	资源池名称。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
PoolsInfo	Array of PoolInfo		资源池信息列表。
QueryType	String	default_type	查询类型，取值说明： • **etl**：批查询模式。 • **interactive**：交互查询模式。 • **default_type**：默认查询模式。 说明 详情请参见查询执行模式。
UpdateTime	String	2022-03-09 16:57:35.241	更新时间。
PoolName	String	USER_DEFAULT	资源池名称。
CreateTime	String	2022-03-09 16:57:35.241	创建时间。
PoolUsers	String	testb,testc	资源池绑定账号列表。
NodeNum	Integer	2	节点数量。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDBResourcePool
&DBClusterId=am-bp11q28kvl688****
&PoolName=USER_DEFAULT
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDBResourcePoolResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
    <PoolsInfo>
        <QueryType>default_type</QueryType>
        <UpdateTime>2022-03-09 16:57:35.241</UpdateTime>
        <PoolName>USER_DEFAULT</PoolName>
        <CreateTime>2022-03-09 16:57:35.241</CreateTime>
        <PoolUsers>testb,testc</PoolUsers>
        <NodeNum>2</NodeNum>
    </PoolsInfo>
</DescribeDBResourcePoolResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "PoolsInfo" : {
    "QueryType" : "default_type",
    "UpdateTime" : "2022-03-09 16:57:35.241",
    "PoolName" : "USER_DEFAULT",
    "CreateTime" : "2022-03-09 16:57:35.241",
    "PoolUsers" : "testb,testc",
    "NodeNum" : 2
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.7.4. ModifyDBResourcePool

调用ModifyDBResourcePool接口更新资源池资源。该接口仅适用于ADB弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ModifyDBResourcePool	系统规定参数，取值： ModifyDBResourcePool。
DBClusterId	String	是	am-bpxxxxxxxx47	实例ID。
PoolName	String	是	test	资源池名称。
QueryType	String	否	interactive	查询类型： • etl：批处理模式。 • interactive：交互查询模式。 • default_type：默认查询模式。 默认值default_type，详情请参见查询执行模式。参数QueryType NodeNum至少传一个
NodeNum	Integer	否	2	节点数，单节点对应配置16核64GB，默认资源池USER_DEFAULT不可更新该信息。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ModifyDBResourcePool
&DBClusterId=am-bpxxxxxxxx47
&PoolName=test
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.7.5. BindDBResourcePoolWithUser

调用BindDBResourcePoolWithUser接口将资源池与数据库用户进行绑定。该接口仅适用于ADB弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	BindDBResourcePoolWithUser	系统规定参数，取值：BindDBResourcePoolWithUser。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
PoolName	String	是	realtime	资源池名称。
PoolUser	String	是	testa	需要绑定资源池的数据库账号。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=BindDBResourcePoolWithUser
&DBClusterId=am-bp1xxxxxxxx47
&PoolName=realtime
&PoolUser=testa
&<公共请求参数>
```

正常返回示例

XML 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

JSON 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.7.6. UnbindDBResourcePoolWithUser

调用UnbindDBResourcePoolWithUser接口解绑资源池用户。该接口仅适用于ADB弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	UnbindDBResourcePoolWithUser	系统规定参数，取值： UnbindDBResourcePoolWithUser。
DBClusterId	String	是	am-bpxxxxxxxx47	集群ID。
PoolName	String	是	test	资源池名称。默认资源池USER_DEFAULT不可做解绑操作。
PoolUser	String	是	testb	资源池用户。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=UnbindDBResourcePoolWithUser
&DBClusterId=am-bpxxxxxxxx47
&PoolName=test
&PoolUser=testb
&<公共请求参数>
```

正常返回示例

XML 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

JSON 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.8. 弹性计划管理

9.8.1. CreateElasticPlan

调用CreateElasticPlan创建分时弹性计划。该接口仅适用于ADB弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreateElasticPlan	系统规定参数，取值：CreateElasticPlan。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
ElasticPlanEnable	Boolean	是	true	弹性计划生效开关：true/false，默认开关打开。
ElasticPlanEndDay	String	是	2020-11-11	弹性计划结束日期，如：2020-11-11。
ElasticPlanName	String	是	test	弹性计划名称。
ElasticPlanNodeNum	Integer	是	2	弹性计划节点数。
ElasticPlanStartDay	String	是	2020-10-10	弹性计划开始日期，如：2020-10-10。
ElasticPlanTimeEnd	String	是	10:00:00	弹性计划结束时间，支持整点，如：10:00:00，可延伸至第二天，但不能超过24小时。
ElasticPlanTimeStart	String	是	08:00:00	弹性计划开始时间，支持整点，如：08:00:00。
ElasticPlanWeeklyRepeat	String	是	1,2,3,4,5	执行弹性计划的周期。按周配置 0～6表示周日到周六，多个日期间用 `,` 分割，如：1,2,3,4,5。
ResourcePoolName	String	是	realtime	资源池名称。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreateElasticPlan
&DBClusterId=am-bp1xxxxxxxx47
&ElasticPlanEnable=true
&ElasticPlanEndDay=2020-11-11
&ElasticPlanName=test
&ElasticPlanNodeNum=2
&ElasticPlanStartDay=2020-10-10
&ElasticPlanTimeEnd=10:00:00
&ElasticPlanTimeStart=08:00:00
&ElasticPlanWeeklyRepeat=1,2,3,4,5
&ResourcePoolName=realtime
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.8.2. DeleteElasticPlan

调用DeleteElasticPlan接口删除分时弹性计划。该接口仅适用于ADB弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteElasticPlan	系统规定参数，取值： DeleteElasticPlan。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
ElasticPlanName	String	是	test	弹性计划名称。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteElasticPlan
&DBClusterId=am-bp1xxxxxxxx47
&ElasticPlanName=test
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.8.3. DescribeElasticDailyPlan

调用DescribeElasticDailyPlan接口查询AnalyticDB MySQL集群分时弹性计划日的计划信息列表。

使用说明

该接口仅适用于AnalyticDB MySQL弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeElasticDailyPlan	系统规定参数。取值：**DescribeElasticDailyPlan**。
DBClusterId	String	是	am-bp11q28kvl688****	集群ID。
ElasticPlanName	String	是	realtimeplan	弹性计划名称，长度不能超过64byte。
ResourcePoolName	String	是	test	执行日计划的资源池名称，长度不能超过64byte。
ElasticDailyPlanDay	String	是	2020-10-16	日计划执行日期，格式：YYYY-MM-DD。
ElasticDailyPlanStatusList	String	是	1,2,3,4	日计划执行状态列表，取值说明： • **1**：未执行 • **2**：执行中 • **3**：执行成功 • **4**：执行失败 同时查询多个状态用逗号（`,`）分割。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
ElasticDailyPlanList	Array of ElasticDailyPlanInfo		弹性计划日计划信息列表。
Status	Integer	4	日计划执行状态，取值说明： • **1**：未执行 • **2**：执行中 • **3**：执行成功 • **4**：执行失败
Day	String	2020-10-16	日计划日期。
ResourcePoolName	String	test	执行日计划的资源池名称。
StartTs	String	2020-10-16 16:00:00	开始时间。
PlanEndTs	String	2020-10-16 16:00:00	计划结束时间。
PlanStartTs	String	2020-10-16 15:00:00	计划开始时间。
ElasticNodeNum	Integer	1	分时弹性节点数。
EndTs	String	2020-10-16 16:00:00	结束时间。

名称	类型	示例值	描述
PlanName	String	realtimeplan	弹性计划名称。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeElasticDailyPlan
&DBClusterId=am-bp11q28kv1688****
&ElasticPlanName=realtimeplan
&ResourcePoolName=test
&ElasticDailyPlanDay=2020-10-16
&ElasticDailyPlanStatusList=1,2,3,4
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeElasticDailyPlanResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
    <ElasticDailyPlanList>
        <Status>4</Status>
        <Day>2020-10-16</Day>
        <ResourcePoolName>test</ResourcePoolName>
        <StartTs>2020-10-16 16:00:00</StartTs>
        <PlanEndTs>2020-10-16 16:00:00</PlanEndTs>
        <PlanStartTs>2020-10-16 15:00:00</PlanStartTs>
        <ElasticNodeNum>1</ElasticNodeNum>
        <EndTs>2020-10-16 16:00:00</EndTs>
        <PlanName>realtimeplan</PlanName>
    </ElasticDailyPlanList>
</DescribeElasticDailyPlanResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "ElasticDailyPlanList" : {
    "Status" : 4,
    "Day" : "2020-10-16",
    "ResourcePoolName" : "test",
    "StartTs" : "2020-10-16 16:00:00",
    "PlanEndTs" : "2020-10-16 16:00:00",
    "PlanStartTs" : "2020-10-16 15:00:00",
    "ElasticNodeNum" : 1,
    "EndTs" : "2020-10-16 16:00:00",
    "PlanName" : "realtimeplan"
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.8.4. DescribeElasticPlan

调用DescribeElasticPlan接口查询弹性计划。该接口仅适用于ADB弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeElasticPlan	系统规定参数，取值：DescribeElasticPlan。
DBClusterId	String	是	am-bpxxxxxxxx47	集群ID。
ElasticPlanEnable	Boolean	是	true	弹性计划生效开关。
ElasticPlanName	String	是	realtime	弹性计划名称。
ResourcePoolName	String	是	test	资源池名称。

返回数据

名称	类型	示例值	描述
ElasticPlanList	Array of ElasticPlanInfo		弹性计划列表。
ElasticNodeNum	Integer	1	弹性计划节点数量。
Enable	Boolean	true	弹性计划是否生效。
EndDay	String	2020-11-16	计划结束日期，设置该字段值时才会有对应返回字段信息。
EndTime	String	16:00:00	计划结束时间。
PlanName	String	realtime	计划名称。
ResourcePoolName	String	test	资源池名称。
StartDay	String	2020-10-30	计划开始日期，设置该字段值时才会有对应返回字段信息。
StartTime	String	15:00:00	计划开始时间。
WeeklyRepeat	String	3,4,5,6	弹性计划按周执行的周期。0～6表示周日到周六，多个日期间用`,`分割。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeElasticPlan
&DBClusterId=am-bpxxxxxxxx47
&ElasticPlanEnable=true
&ElasticPlanName=realtime
&ResourcePoolName=test
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<ElasticPlanList>
    <WeeklyRepeat>3,4,5,6</WeeklyRepeat>
    <EndTime>16:00:00</EndTime>
    <StartTime>15:00:00</StartTime>
    <ElasticNodeNum>1</ElasticNodeNum>
    <Enable>true</Enable>
    <ResourcePoolName>test</ResourcePoolName>
    <EndDay>2020-11-16</EndDay>
    <StartDay>2020-10-30</StartDay>
    <PlanName>realtime</PlanName>
</ElasticPlanList>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "ElasticPlanList": {
        "WeeklyRepeat": "3,4,5,6",
        "EndTime": "16:00:00",
        "StartTime": "15:00:00",
        "ElasticNodeNum": 1,
        "Enable": true,
        "ResourcePoolName": "test",
        "EndDay": "2020-11-16",
        "StartDay": "2020-10-30",
        "PlanName": "realtime"
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.8.5. ModifyElasticPlan

调用ModifyElasticPlan接口修改AnalyticDB MySQL版集群的分时弹性计划。

使用说明

该接口仅适用于AnalyticDB MySQL版弹性模式集群版（新版）。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ModifyElasticPlan	系统规定参数。取值：**ModifyElasticPlan**。
DBClusterId	String	是	am-bp1rqvm70uh2v****	集群ID。
ElasticPlanName	String	是	realtime	弹性计划名称。
ResourcePoolName	String	是	test	资源池名称。
ElasticPlanNodeNum	Integer	否	1	弹性计划节点数。
ElasticPlanTimeStart	String	否	8:00:00	弹性计划开始时间，支持整点。
ElasticPlanTimeEnd	String	否	10:00:00	弹性计划结束时间，支持整点，可延伸至第二天，与弹性计划开始时间的间隔不能超过24小时。

名称	类型	是否必选	示例值	描述
ElasticPlanWeeklyRepeat	String	否	1,2,3,4,5	执行弹性计划的周期。按周配置0～6表示周日到周六，多个日期间用逗号（,）分割 。
ElasticPlanStartDay	String	否	2022-03-18	弹性计划开始日期。
ElasticPlanEndDay	String	否	2022-03-18	弹性计划结束日期。
ElasticPlanEnable	Boolean	是	true	弹性计划生效开关。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=ModifyElasticPlan
&DBClusterId=am-bp1rqvm70uh2v****
&ElasticPlanName=realtime
&ResourcePoolName=test
&ElasticPlanNodeNum=1
&ElasticPlanTimeStart=08:00:00
&ElasticPlanTimeEnd=10:00:00
&ElasticPlanWeeklyRepeat=1,2,3,4,5
&ElasticPlanStartDay=2022-03-18
&ElasticPlanEndDay=2022-03-18
&ElasticPlanEnable=true
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ModifyElasticPlanResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
</ModifyElasticPlanResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9. 数据库

9.9.1. DescribeTables

调用DescribeTables接口枚举实例指定数据库下所有表列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeTables	系统规定参数，取值：DescribeTables。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
SchemaName	String	是	adb_demo	数据库名。

返回数据

名称	类型	示例值	描述
Items	Array of Table		表列表。
Table			
DBClusterId	String	am-bp1xxxxxxxx47	实例ID。
SchemaName	String	adb_demo	数据库名。
TableName	String	test	表名。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeTables
&DBClusterId=am-bp1xxxxxxxx47
&SchemaName=adb_demo
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<Items>
    <Table>
        <TableName>adb_test</TableName>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Table>
    <Table>
        <TableName>customer</TableName>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Table>
    <Table>
        <TableName>test</TableName>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Table>
</Items>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "Items": {
        "Table": [
            {
                "TableName": "adb_test",
                "DBClusterId": "am-bp1xxxxxxxx47",
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "customer",
                "DBClusterId": "am-bp1xxxxxxxx47",
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "DBClusterId": "am-bp1xxxxxxxx47",
                "SchemaName": "adb_demo"
            }
        ]
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.2. DescribeAllDataSource

调用DescribeAllDataSource接口枚举实例下所有数据库列表、表列表和列列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeAllDataSource	系统规定参数，取值：DescribeAllDataSource。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
SchemaName	String	是	adb_demo	数据库名。
TableName	String	是	test	表名。

返回数据

名称	类型	示例值	描述
Columns	Array of Column		列列表。
Column			
AutoIncrementColumn	Boolean	true	是否自增。
ColumnName	String	id	列名
DBClusterId	String	am-bp1xxxxxxxx47	实例ID。
PrimaryKey	Boolean	false	是否主键。
SchemaName	String	adb_demo	数据库名。

名称	类型	示例值	描述
TableName	String	test	表名。
Type	String	bigint	列数据类型。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
Schemas	Array of Schema		数据库列表。
Schema			
DBClusterId	String	am-bp1xxxxxxxx47	实例ID。
SchemaName	String	adb_demo	数据库名。
Tables	Array of Table		表列表。
Table			
DBClusterId	String	am-bp1xxxxxxxx47	实例ID。
SchemaName	String	adb_demo	数据库名。
TableName	String	test	表名。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeAllDataSource
&DBClusterId=am-bp1xxxxxxxx47
&SchemaName=adb_demo
&TableName=test
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<Tables>
    <Table>
        <TableName>adb_oss_import_test</TableName>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Table>
    <Table>
        <TableName>customer</TableName>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Table>
    <Table>
        <TableName>test</TableName>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Table>
</Tables>
<Columns>
    <Column>
        <TableName>test</TableName>
        <ColumnName>id</ColumnName>
        <Type>bigint</Type>
        <AutoIncrementColumn>true</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
    <Column>
        <TableName>test</TableName>
        <ColumnName>name</ColumnName>
        <Type>varchar</Type>
        <AutoIncrementColumn>false</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
    <Column>
        <TableName>test</TableName>
        <ColumnName>value</ColumnName>
        <Type>int</Type>
        <AutoIncrementColumn>false</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
    <Column>
        <TableName>test</TableName>
        <ColumnName>ts</ColumnName>
        <Type>timestamp</Type>
        <AutoIncrementColumn>false</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
</Columns>
<Schemas>
    <Schema>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>test_adb</SchemaName>
    </Schema>
    <Schema>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>a123</SchemaName>
    </Schema>
    <Schema>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>test</SchemaName>
    </Schema>
    <Schema>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Schema>
    <Schema>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>test_db</SchemaName>
    </Schema>
</Schemas>
```

JSON 格式


```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "Tables": {
        "Table": [
            {
                "TableName": "adb_oss_import_test",
                "DBClusterId": "am-bplxxxxxxxx47",
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "customer",
                "DBClusterId": "am-bplxxxxxxxx47",
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "DBClusterId": "am-bplxxxxxxxx47",
                "SchemaName": "adb_demo"
            }
        ]
    },
    "Columns": {
        "Column": [
            {
                "TableName": "test",
                "ColumnName": "id",
                "Type": "bigint",
                "AutoIncrementColumn": true,
                "DBClusterId": "am-bplxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "ColumnName": "name",
                "Type": "varchar",
                "AutoIncrementColumn": false,
                "DBClusterId": "am-bplxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "ColumnName": "value",
                "Type": "int",
                "AutoIncrementColumn": false,
                "DBClusterId": "am-bplxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "ColumnName": "ts",
                "Type": "timestamp",
                "AutoIncrementColumn": false,
                "DBClusterId": "am-bplxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            }
        ]
    },
    "Schemas": {
        "Schema": [
            {
                "DBClusterId": "am-bplxxxxxxxx47",
                "SchemaName": "test_adb"
            },
            {
                "DBClusterId": "am-bplxxxxxxxx47",
                "SchemaName": "a123"
            },
            {
                "DBClusterId": "am-bplxxxxxxxx47",
                "SchemaName": "test"
            },
            {
                "DBClusterId": "am-bplxxxxxxxx47",
                "SchemaName": "adb_demo"
            },
```

```
            {
                "DBClusterId": "am-bp1xxxxxxxx47",
                "SchemaName": "test_db"
            }
        ]
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.3. DescribeSchemas

调用DescribeSchemas接口枚举实例下所有数据库列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeSchemas	系统规定参数。取值：**DescribeSchemas**。
DBClusterId	String	是	am-bp1xxxxxxxx47	集群ID。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
Items	Array of Schema		数据库列表。
Schema			
DBClusterId	String	am-bp1xxxxxxxx47	集群ID。
SchemaName	String	adb_demo	数据库名。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeSchemas
&DBClusterId=am-bp1xxxxxxxx47
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeSchemasResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
    <Items>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <SchemaName>adb_demo</SchemaName>
    </Items>
</DescribeSchemasResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "Items" : [ {
    "DBClusterId" : "am-bp1xxxxxxxx47",
    "SchemaName" : "adb_demo"
  } ]
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.4. DescribeTableDetail

调用DescribeTableDetail接口表详情，查看分区偏离度。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeTableDetail	系统规定参数，取值：DescribeTableDetail。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
SchemaName	String	是	adb_demo	数据库名。
TableName	String	是	test	表名。

返回数据

名称	类型	示例值	描述
AvgSize	Long	0	平均分区行数。
Items	Array of Shard		分区列表。
Shard			
Id	Integer	1	分区号（只返回分区名数字部分）。
Size	Long	0	表大小，行数。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeTableDetail
&DBClusterId=am-bp1xxxxxxxx47
&SchemaName=adb_demo
&TableName=test
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<AvgSize>0</AvgSize>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<Items>
    <Shard>
        <Size>0</Size>
        <Id>1</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>2</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>3</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>4</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>5</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>6</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>7</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>8</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>9</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>10</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>11</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>12</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>13</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>14</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>15</Id>
    </Shard>
    <Shard>
        <Size>0</Size>
        <Id>16</Id>
    </Shard>
</Items>
```

`JSON` 格式

```
{
    "AvgSize": 0,
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "Items": {
        "Shard": [
            {
                "Size": 0,
                "Id": 1
            },
            {
                "Size": 0,
                "Id": 2
            },
            {
                "Size": 0,
                "Id": 3
            },
            {
                "Size": 0,
                "Id": 4
            },
            {
                "Size": 0,
                "Id": 5
            },
            {
                "Size": 0,
                "Id": 6
            },
            {
                "Size": 0,
                "Id": 7
            },
            {
                "Size": 0,
                "Id": 8
            },
            {
                "Size": 0,
                "Id": 9
            },
            {
                "Size": 0,
                "Id": 10
            },
            {
                "Size": 0,
                "Id": 11
            },
            {
                "Size": 0,
                "Id": 12
            },
            {
                "Size": 0,
                "Id": 13
            },
            {
                "Size": 0,
                "Id": 14
            },
            {
                "Size": 0,
                "Id": 15
            },
            {
                "Size": 0,
                "Id": 16
            }
        ]
    }
}
```


错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.5. DescribeProcessList

调用DescribeProcessList接口查看AnalyticDB MySQL集群正在运行的查询。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeProcessList	系统规定参数。取值：**DescribeProcessList**。
DBClusterId	String	是	am-bp11q28kvl688****	集群ID。
ShowFull	Boolean	否	True	是否展示完整的SQL语句。取值说明： • **True**：展示完整的SQL语句。 • **False**：只展示SQL语句的前100个字符。 说明 默认值False。
RunningTime	Integer	否	5	按运行时间过滤，展示大于该运行时间的查询。单位：秒。
User	String	否	test	按用户名过滤。
Keyword	String	否	SELECT	过滤关键字，目前仅支持过滤**SELECT**。
Order	String	否	[{ "Field":"Time","Type":"Desc" }, { "Field":"User", "Type":"Asc" }]	按指定字段排序，JSON格式，`[{"Field":"Time","Type":"Desc" },{ "Field":"User", "Type":"Asc" }]`，取值说明： • **Field**：需要排序的字段名，支持Time，User，Host，DB字段。 • **Type**：排序类型，**Desc**为降序，**Asc**为升序。
PageSize	Integer	否	30	每页记录数，取值： • **30**（默认值） • **50** • **100**
PageNumber	Integer	否	1	页码，取值为：大于0且不超过Integer数据类型的最大值，默认值为1。

返回数据

名称	类型	示例值	描述
TotalCount	String	1	总记录数。
PageSize	String	30	总页数。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
PageNumber	String	1	页码。
Items	Array of Process		任务列表。
Process			
StartTime	String	2020-11-19T02:48:15Z	任务的开始时间，UTC时间，格式：yyyy-MM-ddTHH:mm:ssZ。

名称	类型	示例值	描述
Time	Integer	11	查询任务已运行时间。单位：秒。
ProcessId	String	20201119104815192168149242031510 0****	任务的唯一标识，KILL PROCESS时使用。
Host	String	192.168.XX.XX:12308	发起查询的IP地址。
DB	String	adb_demo	数据库名。
Command	String	SELECT	命令类型，仅支持SELECT类型。
User	String	test	用户名。
Id	Integer	49104	工作线程ID。
Info	String	select * from sbtest1,sbtest2,sbtest3,sbtest4	正在运行的SQL语句，默认返回前100字符。当传入参数ShowFull为True时，显示全文。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeProcessList
&DBClusterId=am-bp11q28kv1688****
&ShowFull=True
&RunningTime=5
&User=test
&Keyword=SELECT
&Order=[ { "Field":"Time","Type":"Desc" },  { "Field":"User", "Type":"Asc" }]
&PageSize=30
&PageNumber=1
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeProcessListResponse>
    <TotalCount>1</TotalCount>
    <PageSize>30</PageSize>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
    <PageNumber>1</PageNumber>
    <Items>
        <StartTime>2020-11-19T02:48:15Z</StartTime>
        <Time>11</Time>
        <ProcessId>20201119104815192168149242031510 0****</ProcessId>
        <Host>192.168.XX.XX:12308</Host>
        <DB>adb_demo</DB>
        <Command>SELECT</Command>
        <User>test</User>
        <Id>49104</Id>
        <Info>select * from sbtest1,sbtest2,sbtest3,sbtest4</Info>
    </Items>
</DescribeProcessListResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TotalCount" : 1,
  "PageSize" : 30,
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "PageNumber" : 1,
  "Items" : {
    "StartTime" : "2020-11-19T02:48:15Z",
    "Time" : 11,
    "ProcessId" : "20201119104815192168149242031510****",
    "Host" : "192.168.XX.XX:12308",
    "DB" : "adb_demo",
    "Command" : "SELECT",
    "User" : "test",
    "Id" : 49104,
    "Info" : "select * from sbtest1,sbtest2,sbtest3,sbtest4"
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.6. DescribeColumns

调用DescribeColumns接口枚举实例下指定表格的列列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeColumns	系统规定参数，取值：DescribeColumns。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
SchemaName	String	是	adb_demo	数据库名。
TableName	String	是	test	表名。

返回数据

名称	类型	示例值	描述
Items	Array of Column		列列表。
Column			
AutoIncrementColumn	Boolean	true	是否自增。
ColumnName	String	id	列名。
DBClusterId	String	am-bp1xxxxxxxx47	实例ID。
PrimaryKey	Boolean	false	是否主键。
SchemaName	String	adb_demo	数据库名。
TableName	String	test	表名。

名称	类型	示例值	描述
Type	String	bigint	列数据类型。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeColumns
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<Items>
    <Column>
        <TableName>test</TableName>
        <ColumnName>id</ColumnName>
        <Type>bigint</Type>
        <AutoIncrementColumn>true</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
    <Column>
        <TableName>test</TableName>
        <ColumnName>name</ColumnName>
        <Type>varchar</Type>
        <AutoIncrementColumn>false</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
    <Column>
        <TableName>test</TableName>
        <ColumnName>value</ColumnName>
        <Type>int</Type>
        <AutoIncrementColumn>false</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
    <Column>
        <TableName>test</TableName>
        <ColumnName>ts</ColumnName>
        <Type>timestamp</Type>
        <AutoIncrementColumn>false</AutoIncrementColumn>
        <DBClusterId>am-bp1xxxxxxxx47</DBClusterId>
        <PrimaryKey>false</PrimaryKey>
        <SchemaName>adb_demo</SchemaName>
    </Column>
</Items>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "Items": {
        "Column": [
            {
                "TableName": "test",
                "ColumnName": "id",
                "Type": "bigint",
                "AutoIncrementColumn": true,
                "DBClusterId": "am-bp1xxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "ColumnName": "name",
                "Type": "varchar",
                "AutoIncrementColumn": false,
                "DBClusterId": "am-bp1xxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "ColumnName": "value",
                "Type": "int",
                "AutoIncrementColumn": false,
                "DBClusterId": "am-bp1xxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            },
            {
                "TableName": "test",
                "ColumnName": "ts",
                "Type": "timestamp",
                "AutoIncrementColumn": false,
                "DBClusterId": "am-bp1xxxxxxxxx47",
                "PrimaryKey": false,
                "SchemaName": "adb_demo"
            }
        ]
    }
}
```


错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.7. DescribeTaskInfo

调用DescribeTaskInfo接口查看相关管控任务流进度。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeTaskInfo	系统规定参数，取值：DescribeTaskInfo。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
TaskId	Integer	是	225685759	任务ID。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
TaskInfo	Struct		任务信息。
BeginTime	String	2020-01-07T07:39:56Z	开始时间，格式：yyyy-MM-ddTHH:mm:ssZ。
FinishTime	String	2020-01-07T08:08:50Z	结束时间，格式：yyyy-MM-ddTHH:mm:ssZ。
Progress	String	100	进度，单位：%。
Status	String	Finished	状态： • Waiting：等待中 • Running：执行中 • Finished：已结束 • Failed：失败 • Closed：已关闭 • Cancel：已取消 • Retry：重试 • Pause：暂停 • Stop：中断
TaskId	Integer	225685759	任务ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeTaskInfo
&DBClusterId=am-bp1xxxxxxxx47
&TaskId=225685759
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<TaskInfo>
    <Status>Finished</Status>
    <Progress>100</Progress>
    <TaskId>225685759</TaskId>
    <FinishTime>2020-01-07T08:08:50Z</FinishTime>
    <BeginTime>2020-01-07T07:39:56Z</BeginTime>
</TaskInfo>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "TaskInfo": {
        "Status": "Finished",
        "Progress": 100,
        "TaskId": 225685759,
        "FinishTime": "2020-01-07T08:08:50Z",
        "BeginTime": "2020-01-07T07:39:56Z"
    },
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.8. DescribeSQLPlan

调用DescribeSQLPlan接口查询目标SQL，例如查询语句或ETL（Extract Transform Load）任务语句的计划信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeSQLPlan	系统规定参数。取值：**DescribeSQLPlan**。
DBClusterId	String	是	am-********** **********	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
ProcessId	String	是	20210527160443172016166249034 5*******	任务ID。 说明 您可以调用DescribeProcessList接口查看目标SQL的任务ID。

返回数据

名称	类型	示例值	描述
RequestId	String	22D6DEF0-CBC7-4388-A41C-D5FD62******	请求ID。
StageList	Array of SqlPlanStage		计划阶段的信息列表。
State	String	FINISHED	该计划阶段的最终执行状态，取值范围如下： • FINISHED：完成。 • CANCELED：取消。 • ABORTED：中止。 • FAILED：失败。
CPUTimeAvg	Long	5984	该阶段的 `CPU Time` 在各个计算节点的平均值，单位为毫秒（ms）。
CPUTimeMax	Long	5984	该阶段的 `CPU Time` 在各个计算节点的最大值，单位为毫秒（ms）。
OperatorCost	Long	5984	该阶段内部总的算子耗时，等同于该阶段的CPU Time，可用于判断查询哪些部分消耗了较多的计算资源。单位为毫秒（ms）。
ScanTimeMax	Long	0	带数据扫描算子的阶段在各个存储节点读取数据耗时的最大值，单位为毫秒（ms）。
InputSizeMax	Long	173	阶段在各个计算节点的输入数据量的最大值，单位为Byte。
StageId	Integer	1	阶段ID。
ScanSizeMax	Long	0	带数据扫描算子的阶段在各个存储节点的扫描数据量的最大值，单位为Byte。
CPUTimeMin	Long	47	该阶段的 `CPU Time` 在各个计算节点的最小值，单位为毫秒（ms）。
ScanTimeMin	Long	0	带数据扫描算子的阶段在各个存储节点读取数据耗时的最小值，单位为毫秒（ms）。

名称	类型	示例值	描述
ScanSizeMin	Long	0	带数据扫描算子的阶段在各个存储节点的扫描数据量的最小值，单位为Byte。
InputSizeMin	Long	173	阶段在各个计算节点的输入数据量的最小值，单位为Byte。
PeakMemory	Long	74208	执行目标SQL时的峰值内存，单位为Byte。
ScanTimeAvg	Long	0	带数据扫描算子的阶段在各个存储节点读取数据耗时的平均值，单位为毫秒（ms）。
ScanSizeAvg	Long	0	带数据扫描算子的阶段在各个存储节点的扫描数据量的平均值，单位为Byte。
InputSizeAvg	Long	173	阶段在各个计算节点的输入数据量的平均值，单位为Byte。
OriginInfo	String	{\"queryId\":\"20210527_160443_10581_hdhzr\",\"session\":{\"queryId\":\"20210527_160443_10581_hdhzr\",\"hasSharedStage\":false,\"parentId\":0}}	SQL Plan的详细原始数据。
Detail	Object		目标SQL的详细执行信息。
SQL	String	INSERT OVERWRITE INTO hdfs_import_external\nSELECT *\nFROM adb_hdfs_import_source	SQL语句。
OutputSize	Long	9	目标SQL语句的总输出数据量，单位为Byte。
State	String	FINISHED	目标SQL语句的最终执行状态，取值范围如下： • FINISHED：完成。 • FAILED：失败。
OutputRows	Long	1	目标SQL的总输出行数。
User	String	test_acc	提交目标SQL语句的用户名。
StartTime	String	1622102683243	目标SQL语句的执行开始时间，格式为Unix时间戳，单位为毫秒。
TotalStage	Long	4	目标SQL中包含的总阶段个数。
QueuedTime	Long	0	执行目标SQL的排队时间，单位为毫秒（ms）。
TotalTime	Long	2340	目标SQL的执行总耗时，单位为毫秒（ms）。
TotalTask	Long	4	目标SQL中包含的总任务个数。
Database	String	adb_demo	执行目标SQL的数据库名称。
PeakMemory	Long	441802	执行目标SQL时的峰值内存，单位为Byte。
ClientIP	String	172.16.***.***	提交目标SQL语句的客户端IP地址。
PlanningTime	Long	86	执行目标SQL时生成执行计划的时间，单位为毫秒（ms）。
CPUTime	Long	6100	执行目标SQL时，算子处理数据单元的总耗时，是多服务器多线程上的累计值。单位为毫秒（ms）。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeSQLPlan
&DBClusterId=am-****************
&ProcessId=20210527160443172016162490345*******
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeSQLPlanResponse>
    <RequestId>22D6DEF0-CBC7-4388-A41C-D5FD62******</RequestId>
    <StageList>
        <ScanSizeMin>0</ScanSizeMin>
        <ScanSizeAvg>0</ScanSizeAvg>
        <StageId>0</StageId>
        <PeakMemory>264960</PeakMemory>
        <CPUTimeMax>47</CPUTimeMax>
        <ScanTimeMin>0</ScanTimeMin>
        <ScanTimeAvg>0</ScanTimeAvg>
        <InputSizeMax>1023</InputSizeMax>
        <InputSizeMin>1023</InputSizeMin>
        <InputSizeAvg>1023</InputSizeAvg>
        <ScanSizeMax>0</ScanSizeMax>
        <State>FINISHED</State>
        <OperatorCost>47</OperatorCost>
        <ScanTimeMax>0</ScanTimeMax>
        <CPUTimeMin>47</CPUTimeMin>
        <CPUTimeAvg>47</CPUTimeAvg>
    </StageList>
    <StageList>
        <ScanSizeMin>0</ScanSizeMin>
        <ScanSizeAvg>0</ScanSizeAvg>
        <StageId>1</StageId>
        <PeakMemory>74208</PeakMemory>
        <CPUTimeMax>5984</CPUTimeMax>
        <ScanTimeMin>0</ScanTimeMin>
        <ScanTimeAvg>0</ScanTimeAvg>
        <InputSizeMax>173</InputSizeMax>
        <InputSizeMin>173</InputSizeMin>
        <InputSizeAvg>173</InputSizeAvg>
        <ScanSizeMax>0</ScanSizeMax>
        <State>FINISHED</State>
        <OperatorCost>5984</OperatorCost>
        <ScanTimeMax>0</ScanTimeMax>
        <CPUTimeMin>5984</CPUTimeMin>
        <CPUTimeAvg>5984</CPUTimeAvg>
    </StageList>
    <StageList>
        <ScanSizeMin>0</ScanSizeMin>
        <ScanSizeAvg>0</ScanSizeAvg>
        <StageId>2</StageId>
        <PeakMemory>102634</PeakMemory>
        <CPUTimeMax>18</CPUTimeMax>
        <ScanTimeMin>0</ScanTimeMin>
        <ScanTimeAvg>0</ScanTimeAvg>
        <InputSizeMax>173</InputSizeMax>
        <InputSizeMin>173</InputSizeMin>
        <InputSizeAvg>173</InputSizeAvg>
        <ScanSizeMax>0</ScanSizeMax>
        <State>FINISHED</State>
        <OperatorCost>18</OperatorCost>
        <ScanTimeMax>0</ScanTimeMax>
        <CPUTimeMin>18</CPUTimeMin>
        <CPUTimeAvg>18</CPUTimeAvg>
    </StageList>
    <StageList>
        <ScanSizeMin>36</ScanSizeMin>
        <ScanSizeAvg>36</ScanSizeAvg>
        <StageId>3</StageId>
        <PeakMemory>0</PeakMemory>
        <CPUTimeMax>51</CPUTimeMax>
        <ScanTimeMin>10</ScanTimeMin>
        <ScanTimeAvg>10</ScanTimeAvg>
        <InputSizeMax>36</InputSizeMax>
        <InputSizeMin>36</InputSizeMin>
```

```
        <InputSizeAvg>36</InputSizeAvg>
        <ScanSizeMax>36</ScanSizeMax>
        <State>FINISHED</State>
        <OperatorCost>51</OperatorCost>
        <ScanTimeMax>10</ScanTimeMax>
        <CPUTimeMin>51</CPUTimeMin>
        <CPUTimeAvg>51</CPUTimeAvg>
    </StageList>
    <OriginInfo>{"queryId":"20210527_160443_10581_hdhzr","session":{"queryId":"20210527_160443_10581_hdhzr","hasSharedStage":false,"parentI
d":0}}</OriginInfo>
    <Detail>
        <TotalTask>4</TotalTask>
        <OutputSize>9</OutputSize>
        <User>test_acc</User>
        <TotalStage>4</TotalStage>
        <PeakMemory>441802</PeakMemory>
        <StartTime>1622102683243</StartTime>
        <ClientIP>172.16.***.***</ClientIP>
        <SQL>INSERT OVERWRITE INTO hdfs_import_external
SELECT *
FROM adb_hdfs_import_source</SQL>
        <CPUTime>6100</CPUTime>
        <QueuedTime>0</QueuedTime>
        <State>FINISHED</State>
        <Database>adb_demo</Database>
        <TotalTime>2340</TotalTime>
        <PlanningTime>86</PlanningTime>
        <OutputRows>1</OutputRows>
    </Detail>
</DescribeSQLPlanResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "22D6DEF0-CBC7-4388-A41C-D5FD62******",
  "StageList" : [ {
    "ScanSizeMin" : 0,
    "ScanSizeAvg" : 0,
    "StageId" : 0,
    "PeakMemory" : 264960,
    "CPUTimeMax" : 47,
    "ScanTimeMin" : 0,
    "ScanTimeAvg" : 0,
    "InputSizeMax" : 1023,
    "InputSizeMin" : 1023,
    "InputSizeAvg" : 1023,
    "ScanSizeMax" : 0,
    "State" : "FINISHED",
    "OperatorCost" : 47,
    "ScanTimeMax" : 0,
    "CPUTimeMin" : 47,
    "CPUTimeAvg" : 47
  }, {
    "ScanSizeMin" : 0,
    "ScanSizeAvg" : 0,
    "StageId" : 1,
    "PeakMemory" : 74208,
    "CPUTimeMax" : 5984,
    "ScanTimeMin" : 0,
    "ScanTimeAvg" : 0,
    "InputSizeMax" : 173,
    "InputSizeMin" : 173,
    "InputSizeAvg" : 173,
    "ScanSizeMax" : 0,
    "State" : "FINISHED",
    "OperatorCost" : 5984,
    "ScanTimeMax" : 0,
    "CPUTimeMin" : 5984,
    "CPUTimeAvg" : 5984
  }, {
    "ScanSizeMin" : 0,
    "ScanSizeAvg" : 0,
    "StageId" : 2,
    "PeakMemory" : 102634,
    "CPUTimeMax" : 18,
    "ScanTimeMin" : 0,
    "ScanTimeAvg" : 0,
    "InputSizeMax" : 173,
    "InputSizeMin" : 173,
```

```
    "InputSizeAvg" : 173,
    "ScanSizeMax" : 0,
    "State" : "FINISHED",
    "OperatorCost" : 18,
    "ScanTimeMax" : 0,
    "CPUTimeMin" : 18,
    "CPUTimeAvg" : 18
  }, {
    "ScanSizeMin" : 36,
    "ScanSizeAvg" : 36,
    "StageId" : 3,
    "PeakMemory" : 0,
    "CPUTimeMax" : 51,
    "ScanTimeMin" : 10,
    "ScanTimeAvg" : 10,
    "InputSizeMax" : 36,
    "InputSizeMin" : 36,
    "InputSizeAvg" : 36,
    "ScanSizeMax" : 36,
    "State" : "FINISHED",
    "OperatorCost" : 51,
    "ScanTimeMax" : 10,
    "CPUTimeMin" : 51,
    "CPUTimeAvg" : 51
  } ],
  "OriginInfo" : "{\"queryId\":\"20210527_160443_10581_hdhzr\",\"session\":{\"queryId\":\"20210527_160443_10581_hdhzr\",\"hasSharedStage\":
false,\"parentId\":0}}",
  "Detail" : {
    "TotalTask" : 4,
    "OutputSize" : 9,
    "User" : "test_acc",
    "TotalStage" : 4,
    "PeakMemory" : 441802,
    "StartTime" : 1622102683243,
    "ClientIP" : "172.16.***.***",
    "SQL" : "INSERT OVERWRITE INTO hdfs_import_external\nSELECT *\nFROM adb_hdfs_import_source",
    "CPUTime" : 6100,
    "QueuedTime" : 0,
    "State" : "FINISHED",
    "Database" : "adb_demo",
    "TotalTime" : 2340,
    "PlanningTime" : 86,
    "OutputRows" : 1
  }
}
```


错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.9. DescribeSQLPlanTask

调用DescribeSQLPlanTask查询任务信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeSQLPlanTask	系统规定参数，取值：DescribeSQLPlanTask。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
ProcessId	String	是	20190724144530172021111 1570315107****	查询任务ID。
StageId	String	是	1785135913****	任务所属阶段。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
TaskList	Array of SqlPlanTask		任务列表。
ElapsedTime	Long	10	任务从创建到结束的时间差。单位：ms。
InputRows	Long	105	任务的输入数据行数。
InputSize	Long	3763	任务的输入数据量。单位：Byte。
OperatorCost	Long	3	任务在某个节点的算子总耗时，是个多线程累加值，可以用于判断计算是否存在长尾。单位：ms。
OutputRows	Long	105	任务的输出数据行数。
OutputSize	Long	945	任务的输出数据量。单位：Byte。
PeakMemory	Long	898576	任务在某个节点的峰值内存。单位：Byte。
ScanCost	Long	0	带数据源的任务的扫描数据耗时。单位：ms。
ScanRows	Long	0	带数据源的任务的扫描数据行数。
ScanSize	Long	0	带数据源的任务的扫描数据量。单位：Byte。
State	String	FINISHED	任务最终执行状态： • FINISHED：完成。 • CANCELED：取消。 • ABORTED：中止。 • FAILED：失败。
TaskId	Integer	198877623	任务ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeSQLPlanTask
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<TaskList>
    <OutputSize>945</OutputSize>
    <TaskId>198877623</TaskId>
    <InputSize>3763</InputSize>
    <PeakMemory>898576</PeakMemory>
    <ElapsedTime>10</ElapsedTime>
    <InputRows>105</InputRows>
    <ScanRows>0</ScanRows>
    <ScanCost>0</ScanCost>
    <State>FINISHED</State>
    <OutputRows>105</OutputRows>
    <OperatorCost>3</OperatorCost>
    <ScanSize>0</ScanSize>
</TaskList>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "TaskList": {
        "OutputSize": 945,
        "TaskId": 198877623,
        "InputSize": 3763,
        "PeakMemory": 898576,
        "ElapsedTime": 10,
        "InputRows": 105,
        "ScanRows": 0,
        "ScanCost": 0,
        "State": "FINISHED",
        "OutputRows": 105,
        "OperatorCost": 3,
        "ScanSize": 0
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.10. KillProcess

调用KillProcess接口终止正在进行的任务。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	KillProcess	系统规定参数，取值：KillProcess。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
ProcessId	String	是	20201119104815192168149242031510 0****	需要被终止任务的唯一标识，调用DescribeProcessList返回。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=KillProcess
&DBClusterId=am-bp1xxxxxxxx47
&ProcessId=20201119104815192168149242031510****
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.11. DescribeTablePartitionDiagnose

调用DescribeTablePartitionDiagnose接口返回表分区是否合理的诊断结果。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeTablePartitionDiagnose	系统规定参数，取值：DescribeTablePartitionDiagnose。
RegionId	String	是	cn-hangzhou	地域ID。
DBClusterId	String	是	am-bpxxxxxxxx47	实例ID。
PageSize	Integer	否	30	每页记录数，取值： • 30； • 50； • 100； 默认值：30。
PageNumber	Integer	否	1	页码，取值为：大于0且不超过Integer数据类型的最大值，默认值为1。

返回数据

名称	类型	示例值	描述
TotalCount	Integer	1	总记录数。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
PageSize	Integer	123	总页数。
PageNumber	Integer	34	页码。
DBClusterId	String	rm-uf6wjk5xxxxxxxxxx	实例ID。
SuggestMaxRecordsPerPartition	Long	640000000	建议每个二级分区的最大行数。
SuggestMinRecordsPerPartition	Long	64000000	建议每个二级分区的最小行数。
Items	Array of TablePartitionDiagnose		表统计信息。
TableName	String	user	表名。
PartitionDetail	String	[20210110, 20210113,20210123]	不合理分区。
SchemaName	String	test_db	数据库名。
PartitionNumber	Integer	34	分区个数。

示例

请求示例

```
http(s)://[Endpoint]/?AccessKeyId=xxxx
&Action=DescribeTablePartitionDiagnose
&RegionId=cn-hangzhou
&DBClusterId=am-bpxxxxxxxx47
&PageSize=30
&PageNumber=1
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<TotalCount>1</TotalCount>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<PageSize>123</PageSize>
<PageNumber>34</PageNumber>
<DBClusterId>rm-uf6wjk5xxxxxxxxxx</DBClusterId>
<SuggestMaxRecordsPerPartition>640000000</SuggestMaxRecordsPerPartition>
<SuggestMinRecordsPerPartition>64000000</SuggestMinRecordsPerPartition>
<Items>
    <TableName>user</TableName>
    <PartitionDetail>[20210110, 20210113,20210123]</PartitionDetail>
    <SchemaName>test_db</SchemaName>
    <PartitionNumber>34</PartitionNumber>
</Items>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TotalCount" : 1,
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "PageSize" : 123,
  "PageNumber" : 34,
  "DBClusterId" : "rm-uf6wjk5xxxxxxxxxx",
  "SuggestMaxRecordsPerPartition" : 640000000,
  "SuggestMinRecordsPerPartition" : 64000000,
  "Items" : [ {
    "TableName" : "user",
    "PartitionDetail" : "[20210110, 20210113,20210123]",
    "SchemaName" : "test_db",
    "PartitionNumber" : 34
  } ]
}
```

错误码

HttpCode	错误码	错误信息	描述
500	InternalError	An error occurred while processing your request.	系统内部错误，请稍后重试
503	ServiceUnavailable	An error occurred while processing your request.	系统暂时不可用，请稍后重试

访问错误中心查看更多错误码。

9.9.12. DescribeLoadTasksRecords

调用DescribeLoadTasksRecords接口查看异步导入导出任务的详情。

更多关于异步提交导入导出任务的详情，请参见异步提交导入导出任务。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeLoadTasksRecords	系统规定参数。取值为**DescribeLoadTasksRecords**。

名称	类型	是否必选	示例值	描述
DBClusterId	String	是	am-bp2590j****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
StartTime	String	是	2021-05-18T06:00:00Z	查询开始时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。 说明 建议查询开始时间设置为30天内的任意时间点。
EndTime	String	是	2021-05-18T06:30:00Z	查询结束时间，需晚于查询开始时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
DBName	String	否	adb_demo	导入导出任务所涉及的数据库名称。
PageSize	Integer	否	30	每页记录数，取值范围如下： • **30**（默认值）。 • **50**。 • **100**。
PageNumber	Integer	否	1	页码，取值为：大于0且不超过Integer数据类型的最大值，默认值为1。
Order	String	否	[{"Field":"CreateTime", "Type":"desc"}]	按指定字段对任务进行升序或降序排列。参数值需为JSON字符串类型，例如：`[{"Field":"CreateTime", "Type":"desc"}]` 说明 • `Field` 表示需要排序的字段名，支持的字段为：`State`、`CreateTime`、`DBName`、`ProcessID`、`UpdateTime`、`JobName`和`ProcessRows`。 • `Type` 表示排序类型，取值范围为 `Desc`（降序）或 `Asc`（升序），取值不区分大小写。
State	String	否	FINISH	需要查询的异步导入或导出任务的状态，取值范围如下： • **INIT**：任务初始化。 • **RUNNING**：任务进行中。 • **FINISH**：任务执行成功。 • **FAILED**：任务执行失败。

返回数据

名称	类型	示例值	描述
TotalCount	String	1	任务总数。
PageSize	String	30	本页记录数。
RequestId	String	C60B05DB-2B77-421A-98E9-92C20E******	请求ID。
PageNumber	String	1	页码。
DBClusterId	String	am-bp2590j****	集群ID。
LoadTasksRecords	Array of LoadTaskRecord		任务信息列表。
Sql	String	insert overwrite into courses_external_table\nselect * from courses	异步导入导出任务中使用的SQL语句详情。

名称	类型	示例值	描述
State	String	FINISH	任务状态。
CreateTime	String	2021-05-18 18:47:27.0	任务开始时间，精确到毫秒，格式为*yyyy-MM-ddTHH:mm:ss.SSSZ*。
DBName	String	adb_demo	导入导出任务涉及的数据库名称。
ProcessID	String	20210518184727172016166249034 53******	进程ID。
UpdateTime	String	2021-05-18 18:47:31.0	任务状态的更新时间，精确到毫秒，格式为*yyyy-MM-ddTHH:mm:ss.SSSZ*。
JobName	String	20210518184727172016166249034 53******	任务ID。
ProcessRows	Long	6	异步导入导出任务所处理的数据行数。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?&Action=DescribeLoadTasksRecords
&DBClusterId=am-bp2590j****
&StartTime=2021-05-18T06:00:00Z
&EndTime=2021-05-18T06:30:00Z
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeLoadTasksRecordsResponse>
    <TotalCount>1</TotalCount>
    <PageSize>30</PageSize>
    <RequestId>C60B05DB-2B77-421A-98E9-92C20E******</RequestId>
    <PageNumber>1</PageNumber>
    <DBClusterId>am-bp2590j****</DBClusterId>
    <LoadTasksRecords>
        <Sql>insert overwrite into courses_external_table\nselect * from courses</Sql>
        <State>FINISH</State>
        <CreateTime>2021-05-18 18:47:27.0</CreateTime>
        <DBName>adb_demo</DBName>
        <ProcessID>20210518184727172016166249034 53******</ProcessID>
        <UpdateTime>2021-05-18 18:47:31.0</UpdateTime>
        <JobName>20210518184727172016166249034 53******</JobName>
        <ProcessRows>6</ProcessRows>
    </LoadTasksRecords>
</DescribeLoadTasksRecordsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TotalCount" : 1,
  "PageSize" : 30,
  "RequestId" : "C60B05DB-2B77-421A-98E9-92C20E******",
  "PageNumber" : 1,
  "DBClusterId" : "am-bp2590j****",
  "LoadTasksRecords" : {
    "Sql" : "insert overwrite into courses_external_table\\nselect * from courses",
    "State" : "FINISH",
    "CreateTime" : "2021-05-18 18:47:27.0",
    "DBName" : "adb_demo",
    "ProcessID" : "20210518184727172016166249034 53******",
    "UpdateTime" : "2021-05-18 18:47:31.0",
    "JobName" : "20210518184727172016166249034 53******",
    "ProcessRows" : 6
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.9.13. DescribeConnectionCountRecords

调用DescribeConnectionCountRecords接口查询目标AnalyticDB MySQL版集群当前的连接数和用户基本信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeConnectionCountRecords	系统规定参数。取值：**DescribeConnectionCountRecords**。
DBClusterId	String	是	am-bp1jj9xqft1po****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。

返回数据

名称	类型	示例值	描述
RequestId	String	562C7F89-FBE6-4A04-AAAA-7EBC25******	请求ID。
DBClusterId	String	am-bp1jj9xqft1po****	集群ID。
AccessIpRecords	Array of AccessIps		客户端IP信息。
AccessIp	String	42.120.XX.XX	客户端IP。
Count	Long	1	连接数。
UserRecords	Array of Users		用户信息列表。
User	String	test	用户名。
Count	Long	1	连接数。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeConnectionCountRecords
&DBClusterId=am-bp1jj9xqft1po****
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeConnectionCountRecordsResponse>
    <RequestId>562C7F89-FBE6-4A04-AAAA-7EBC25******</RequestId>
    <DBClusterId>am-bp1jj9xqft1po****</DBClusterId>
    <AccessIpRecords>
        <AccessIp>42.120.XX.XX</AccessIp>
        <Count>1</Count>
    </AccessIpRecords>
    <UserRecords>
        <User>test</User>
        <Count>1</Count>
    </UserRecords>
</DescribeConnectionCountRecordsResponse>
```


`JSON` 格式


```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "562C7F89-FBE6-4A04-AAAA-7EBC25******",
  "DBClusterId" : "am-bp1jj9xqft1po****",
  "AccessIpRecords" : {
    "AccessIp" : "42.120.XX.XX",
    "Count" : 1
  },
  "UserRecords" : {
    "User" : "test",
    "Count" : 1
  }
}
```


错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.10. 日志管理

9.10.1. DescribeSlowLogRecords

调用DescribeSlowLogRecords接口查看AnalyticDB MySQL版集群的慢日志明细。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeSlowLogRecords	系统规定参数。取值：**DescribeSlowLogRecords**。
DBClusterId	String	是	am-***************	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
StartTime	String	是	2021-05-20T16:00:00Z	查询开始时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
EndTime	String	是	2021-05-27T16:00:00Z	查询结束时间，格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。 说明 查询结束时间需晚于查询开始时间，且与查询开始时间间隔小于7天。
DBName	String	否	adb_demo	数据库名称。

名称	类型	是否必选	示例值	描述
PageSize	Integer	否	30	每页记录数，取值为**30**（默认值）、**50**或**100**。
PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。默认值为**1**。
ProcessID	String	否	20210527160443172016166249034 53******	进程ID。
Order	String	否	[{"Field":"ExecutionStartTime","Type":"Desc"},{"Field":"ScanRows","Type":"Asc"}]	根据指定字段进行排序，格式为JSON，是一个有序JSON数组，按输入数组的顺序进行复合排序，包含 `Field` 和 `Type` 两个字段，例如 `[{"Field":"ExecutionStartTime","Type":"Desc"},{"Field":"ScanRows","Type":"Asc"}]` 。其中： • `Field` 表示需要排序的字段名，支持如下取值： ◦ **HostAddress**：连接数据库的客户端地址。 ◦ **UserName**：用户名。 ◦ **ExecutionStartTime**：目标SQL的执行开始时间。 ◦ **QueryTime**：目标SQL执行时长。 ◦ **PeakMemoryUsage**：执行目标SQL语句时的峰值内存。 ◦ **ScanRows**：带数据源的任务的扫描数据行数。 ◦ **ScanSize**：扫描的数据量。 ◦ **ScanTime**：扫描数据总耗时。 ◦ **PlanningTime**：执行计划生成耗时。 ◦ **WallTime**：查询中的所有算子在各个节点CPU Time的累加值。 ◦ **ProcessID**：进程ID。 • `Type` 表示排序类型，支持如下取值： ◦ **Desc**：降序。 ◦ **Asc**：升序。
Range	String	否	[{"Field":"ScanSize","Min":"1000000","Max":"10000000"},{"Field":"QueryTime","Min":"1000","Max":"10000"}]	根据指定字段的最大值（ `Max` ）和最小值（ `Min` ）进行范围过滤，格式为JSON格式，是一个JSON数组，例如 `[{"Field":"ScanSize","Min":"1000000","Max":"10000000"},{"Field":"QueryTime","Min":"1000","Max":"10000"}]` 。 其中 `Field` 字段表示需要限制范围的字段，支持如下取值： • **ScanSize**：扫描的数据量，单位：KB。 • **QueryTime**：执行时长，单位：毫秒（ms）。 • **PeakMemoryUsage**：执行目标SQL语句时的峰值内存，单位：KB。 说明 `Min` 表示查询范围最小值（左值），`Max` 表示查询范围最大值（右值），数据类型都为String。
State	String	否	SUCCESSED	查询状态。

返回数据

名称	类型	示例值	描述
TotalCount	String	100	总记录数。
PageSize	String	30	本页记录数。
RequestId	String	D7559209-7EC3-41E1-8F78-156990******	请求ID。

名称	类型	示例值	描述
PageNumber	String	1	页码。
DBClusterId	String	am-**********	集群ID。
Items	Array of SlowLogRecord		慢日志明细列表。
SlowLogRecord			
HostAddress	String	172.16.***.***	连接数据库的客户端地址。
ScanTime	Long	10	扫描数据总耗时，是多个TableScanNode在多个节点上的累加值，单位：毫秒。
SQLText	String	INSERT OVERWRITE INTO hdfs_import_external\nSELECT *\nFROM adb_hdfs_import_source	SQL语句详情。
OutputSize	String	0.009	任务的输出数据量。单位：Byte。
PeakMemoryUsage	String	431.447	执行目标SQL语句时的峰值内存，单位：KB。
State	String	SUCCESSED	SQL语句的执行状态。
WallTime	Long	6100	查询中的所有算子在各个节点CPU Time的累加值，单位：毫秒（ms）。
ScanSize	String	0.035	扫描的数据量，单位：KB。
ExecutionStartTime	String	2021-05-27T08:04:43Z	执行开始时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
QueryTime	Long	2344	执行时长，单位：毫秒（ms）。
ReturnRowCounts	Long	1	返回行数。
ScanRows	Long	3	带数据源的任务的扫描数据行数。
ParseRowCounts	Long	0	解析行数。
DBName	String	adb_demo	数据库名称。
PlanningTime	Long	12	执行计划生成耗时，单位：毫秒（ms）。
QueueTime	Long	0	查询执行前的排队时间，单位：毫秒（ms）。
UserName	String	test	用户名。
ProcessID	String	20210527160443172016166249034 53******	进程ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeSlowLogRecords
&DBClusterId=am-**********
&StartTime=2021-05-20T16:00:00Z
&EndTime=2021-05-27T16:00:00Z
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeSlowLogRecordsResponse>
    <TotalCount>1</TotalCount>
    <RequestId>D7559209-7EC3-41E1-8F78-156990******</RequestId>
    <PageSize>30</PageSize>
    <PageNumber>1</PageNumber>
    <DBClusterId>am-****************</DBClusterId>
    <Items>
        <SlowLogRecord>
            <OutputSize>0.009</OutputSize>
            <UserName>test</UserName>
            <PeakMemoryUsage>431.447</PeakMemoryUsage>
            <ExecutionStartTime>2021-05-27T08:04:43Z</ExecutionStartTime>
            <ParseRowCounts>0</ParseRowCounts>
            <QueryTime>2344</QueryTime>
            <ScanTime>10</ScanTime>
            <HostAddress>172.16.***.***</HostAddress>
            <SQLText>INSERT OVERWRITE INTO hdfs_import_external
SELECT *
FROM adb_hdfs_import_source</SQLText>
            <WallTime>6100</WallTime>
            <ScanRows>3</ScanRows>
            <State>SUCCESSED</State>
            <ReturnRowCounts>1</ReturnRowCounts>
            <PlanningTime>86</PlanningTime>
            <DBName>adb_demo</DBName>
            <QueueTime>0</QueueTime>
            <ProcessID>20210527160443172016166249034536******</ProcessID>
            <ScanSize>0.035</ScanSize>
        </SlowLogRecord>
    </Items>
</DescribeSlowLogRecordsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TotalCount" : 1,
  "RequestId" : "D7559209-7EC3-41E1-8F78-156990******",
  "PageSize" : 30,
  "PageNumber" : 1,
  "DBClusterId" : "am-****************",
  "Items" : {
    "SlowLogRecord" : [ {
      "OutputSize" : "0.009",
      "UserName" : "test",
      "PeakMemoryUsage" : "431.447",
      "ExecutionStartTime" : "2021-05-27T08:04:43Z",
      "ParseRowCounts" : 0,
      "QueryTime" : 2344,
      "ScanTime" : 10,
      "HostAddress" : "172.16.***.***",
      "SQLText" : "INSERT OVERWRITE INTO hdfs_import_external\nSELECT *\nFROM adb_hdfs_import_source",
      "WallTime" : 6100,
      "ScanRows" : 3,
      "State" : "SUCCESSED",
      "ReturnRowCounts" : 1,
      "PlanningTime" : 86,
      "DBName" : "adb_demo",
      "QueueTime" : 0,
      "ProcessID" : "20210527160443172016166249034536******",
      "ScanSize" : "0.035"
    } ]
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.10.2. DescribeSlowLogTrend

调用DescribeSlowLogTrend接口查看集群慢日志趋势统计情况。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeSlowLogTrend	系统规定参数。取值：**DescribeSlowLogTrend**。
DBClusterId	String	是	am-uf6wjk5xxxxxxxxx	实例ID。
StartTime	String	是	2019-06-01T16:00:00Z	查询开始时间。格式：yyyy-MM-ddTHH:mm:ssZ（UTC时间）。
EndTime	String	是	2019-06-20T16:00:00Z	查询结束时间，需要大于查询开始时间，且与查询开始时间间隔小于7天。 格式：yyyy-MM-ddTHH:mm:ssZ（UTC时间）。
DBName	String	否	test_db	数据库名称。

返回数据

名称	类型	示例值	描述
EndTime	String	2019-06-20T16:00:00Z	查询结束时间，需要大于查询开始时间，且与查询开始时间间隔小于7天。 格式：yyyy-MM-ddTHH:mm:ssZ（UTC时间）。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
StartTime	String	2019-06-01T16:00:00Z	查询开始时间。格式：yyyy-MM-ddTHH:mm:ssZ（UTC时间）。
DBClusterId	String	am-uf6wjk5xxxxxxxxx	实例ID。
Items	Array of SlowLogTrendItem		慢SQL趋势数据列表。
SlowLogTrendItem			
Key	String	AnalyticDB_SlowLogTrend	参数，固定为AnalyticDB_SlowLogTrend。
Unit	String	%	此参数为预留字段。数据单位。
Series	Array of SeriesItem		性能值列表。
SeriesItem			
Values	String	[["2019-05-06T05:17:46.487Z", 5], ["2019-05-06T05:18:20.784Z", 5]]	数组格式。
Name	String	slow_log_trend	性能值名称。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeSlowLogTrend
&DBClusterId=am-uf6wjk5xxxxxxxxx
&StartTime=2019-06-01T16:00:00Z
&EndTime=2019-06-20T16:00:00Z
&DBName=test_db
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeSlowLogTrendResponse>
    <EndTime>2019-06-20T16:00:00Z</EndTime>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
    <StartTime>2019-06-01T16:00:00Z</StartTime>
    <DBClusterId>am-uf6wjk5xxxxxxxxxx</DBClusterId>
    <Items>
        <Key>AnalyticDB_SlowLogTrend</Key>
        <Unit> %</Unit>
        <Series>
            <Values>[ [ "2019-05-06T05:17:46.487Z", 5 ], [ "2019-05-06T05:18:20.784Z", 5 ] ]</Values>
            <Name>slow_log_trend</Name>
        </Series>
    </Items>
</DescribeSlowLogTrendResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "EndTime" : "2019-06-20T16:00:00Z",
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "StartTime" : "2019-06-01T16:00:00Z",
  "DBClusterId" : "am-uf6wjk5xxxxxxxxxx",
  "Items" : [ {
    "Key" : "AnalyticDB_SlowLogTrend",
    "Unit" : " %",
    "Series" : [ {
      "Values" : "[ [ \"2019-05-06T05:17:46.487Z\", 5 ], [ \"2019-05-06T05:18:20.784Z\", 5 ] ]",
      "Name" : "slow_log_trend"
    } ]
  } ]
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.10.3. DescribeAuditLogConfig

调用DescribeAuditLogConfig接口查询AnalyticDB MySQL版集群的SQL审计日志设置。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
DBClusterId	String	是	am-t4nj8619bz2w3****	集群ID。
RegionId	String	是	cn-hangzhou	地域ID。您可通过接口DescribeRegions查看可用的地域ID。
Action	String	是	DescribeAuditLogConfig	系统规定参数。取值：**DescribeAuditLogConfig**。

返回数据

名称	类型	示例值	描述
RequestId	String	F0983B43-B2EC-536A-8791-142B5CF1E9B6	请求ID。

名称	类型	示例值	描述
AuditLogStatus	String	on	SQL审计的开启状态。取值说明： • **on**：开启SQL审计。 • **off**：关闭SQL审计。
DBClusterId	String	am-t4nj8619bz2w3****	集群ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeAuditLogConfig
&DBClusterId=am-t4nj8619bz2w3****
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeAuditLogConfigResponse>
    <RequestId>F0983B43-B2EC-536A-8791-142B5CF1E9B6</RequestId>
    <AuditLogStatus>on</AuditLogStatus>
    <DBClusterId>am-t4nj8619bz2w3****</DBClusterId>
</DescribeAuditLogConfigResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "F0983B43-B2EC-536A-8791-142B5CF1E9B6",
  "AuditLogStatus" : "on",
  "DBClusterId" : "am-t4nj8619bz2w3****"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.10.4. ModifyAuditLogConfig

调用ModifyAuditLogConfig接口修改AnalyticDB MySQL版集群的SQL审计日志设置。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
DBClusterId	String	是	am-t4nj8619bz2w3****	集群ID。
RegionId	String	是	cn-hangzhou	地域ID，您可通过接口DescribeRegions查看可用的地域ID。
AuditLogStatus	String	是	on	SQL审计的开启状态。取值说明： • **on**：开启SQL审计。 • **off**：关闭SQL审计。
Action	String	是	ModifyAuditLogConfig	系统规定参数。取值：**ModifyAuditLogConfig**。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
UpdateSucceed	Boolean	true	是否已更新SQL审计的开启状态。取值说明： • **true**：状态已更新。 • **false**：状态更新失败。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=ModifyAuditLogConfig
&DBClusterId=am-t4nj8619bz2w3****
&RegionId=cn-hangzhou
&AuditLogStatus=on
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ModifyAuditLogConfigResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
    <UpdateSucceed>true</UpdateSucceed>
</ModifyAuditLogConfigResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "UpdateSucceed" : true
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.10.5. 查询集群的SQL审计日志

调用DescribeAuditLogRecords接口查询AnalyticDB MySQL版集群的SQL审计日志。

使用说明

调用本接口查看AnalyticDB MySQL集群的SQL审计日志前，需要开启SQL审计。您可以调用DescribeAuditLogConfig接口查询SQL审计的开启状态，如未开启，可调用ModifyAuditLogConfig接口开启SQL审计。

仅当SQL审计状态为开启时，才能查询到SQL审计日志，且只支持查询30天内的SQL审计日志。如果中途关闭了SQL审计，再次打开时，仅能查询到再次打开后的SQL审计日志。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
DBClusterId	String	是	am-t4nj8619bz2w3****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。

名称	类型	是否必选	示例值	描述
RegionId	String	是	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。
StartTime	String	是	2022-01-23T02:18Z	查询开始时间，UTC时间，格式为：yyyy-MM-ddTHH:mmZ。 说明 仅当SQL审计状态为开启时，才能查询到SQL审计日志，且只支持查询30天内的SQL审计日志。如果中途关闭了SQL审计，再次打开时，仅能查询到再次打开后的SQL审计日志。
EndTime	String	是	2022-01-23T22:18Z	查询结束时间，UTC时间，格式为：yyyy-MM-ddTHH:mmZ。 说明 • 查询结束时间需晚于查询开始时间。 • 查询开始时间与查询结束时间的间隔不能超过24小时。
DBName	String	否	adb_demo	执行SQL的数据库名称。
QueryKeyword	String	否	adb	对查询包含目标关键字的SQL进行查询。
SqlType	String	否	SELECT	SQL类型，取值： • DELETE • SELECT • UPDATE • INSERT_INTO_SELECT • ALTER • DROP • INSERT 说明 每次仅允许传入一种类型查询，若该参数为空，默认查询SELECT类型。
Succeed	String	否	true	目标SQL是否执行成功，取值说明： • true：执行成功。 • false：执行失败。
HostAddress	String	否	100.104.XX.XX:43908	执行目标SQL的客户端IP地址和端口号。
OrderType	String	否	asc	按SQL执行时间进行正序或倒序排序，取值说明： • asc：正序。 • desc：倒序。
User	String	否	test_user	执行目标SQL的用户名。

名称	类型	是否必选	示例值	描述

Order	String	否	[{"Field":"ExecuteTime","Type":"Desc"},{"Field":"HostAddress","Type":"Asc"}]	根据指定字段进行排序，格式为JSON，是一个有序JSON数组，按输入数组的顺序进行复合排序，包含Field和Type两个字段。
				• Field表示需要排序的字段名，取值说明：
				  ○ HostAddress：连接数据库的客户端地址。
				  ○ Succeed：目标SQL是否执行成功。
				  ○ TotalTime：目标SQL的执行总耗时。
				  ○ DBName：执行目标SQL的数据库名称。
				  ○ SQLType：SQL类型。
				  ○ User：执行目标SQL的用户名。
				  ○ ExecuteTime：目标SQL的执行开始时间。
				• Type表示排序类型，取值说明：
				  ○ Desc：降序。
				  ○ Asc：升序。

PageSize	Integer	否	10	每页记录数，取值： • **10** • **30** • **50** • **100** **说明** 本参数不填写时，默认为10。
PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。默认值为**1**。
Action	String	是	DescribeAuditLogRecords	系统规定参数。取值：**DescribeAuditLogRecords**。

返回数据

名称	类型	示例值	描述
TotalCount	String	1	总记录数。
PageSize	String	10	每页记录数。
RequestId	String	8A564B7F-8C00-43C0-8EC5-919FBB70573	请求ID。
PageNumber	String	1	页码。
DBClusterId	String	am-t4nj8619bz2w3****	集群ID。
Items	Array of SlowLogRecord		SQL审计日志详情。
HostAddress	String	100.104.XX.XX:43908	执行目标SQL的客户端IP地址和端口号。
Succeed	String	true	目标SQL是否执行成功。取值说明： • **true**：执行成功。 • **false**：执行失败。
SQLText	String	SELECT * FROM tb_courses	SQL语句详情。
TotalTime	String	216	目标SQL的执行时长，单位：毫秒（ms）。
ConnId	String	无	此参数无效。
DBName	String	adb_test	执行目标SQL的数据库名称。

名称	类型	示例值	描述
SQLType	String	SELECT	目标SQL的类型。
ExecuteTime	String	2022-01-23 16:05:08	目标SQL的执行开始时间，本地时间，格式为：yyyy-MM-dd HH:mm:ss。
ProcessID	String	20210608175202172016166249034536239 0	任务ID。
User	String	test_user	执行目标SQL的用户名。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?DBClusterId=am-t4nj8619bz2w3****
&RegionId=cn-hangzhou
&StartTime=2022-01-23T02:18Z
&EndTime=2022-01-23T22:18Z
&DBName=adb_demo
&QueryKeyword=adb
&SqlType=SELECT
&Succeed=true
&HostAddress=100.104.XX.XX:43908
&OrderType=asc
&User=test_user
&Order=[{"Field":"ExecuteTime","Type":"Desc"},{"Field":"HostAddress","Type":"Asc"}]
&PageSize=10
&PageNumber=1
&Action=DescribeAuditLogRecords
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeAuditLogRecordsResponse>
    <TotalCount>1</TotalCount>
    <PageSize>10</PageSize>
    <RequestId>8A564B7F-8C00-43C0-8EC5-919FBB70573</RequestId>
    <PageNumber>1</PageNumber>
    <DBClusterId>am-t4nj8619bz2w3****</DBClusterId>
    <Items>
        <HostAddress>100.104.XX.XX:43908</HostAddress>
        <Succeed>true</Succeed>
        <SQLText>SELECT * FROM tb_courses</SQLText>
        <TotalTime>216</TotalTime>
        <ConnId>无</ConnId>
        <DBName>adb_test</DBName>
        <SQLType>SELECT</SQLType>
        <ExecuteTime>2022-01-23 16:05:08</ExecuteTime>
        <ProcessID>20210608175202172016166249034536239 0</ProcessID>
        <User>test_user</User>
    </Items>
</DescribeAuditLogRecordsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TotalCount" : 1,
  "PageSize" : 10,
  "RequestId" : "8A564B7F-8C00-43C0-8EC5-919FBB70573",
  "PageNumber" : 1,
  "DBClusterId" : "am-t4nj8619bz2w3****",
  "Items" : {
    "HostAddress" : "100.104.XX.XX:43908",
    "Succeed" : true,
    "SQLText" : "SELECT * FROM tb_courses",
    "TotalTime" : 216,
    "ConnId" : "无",
    "DBName" : "adb_test",
    "SQLType" : "SELECT",
    "ExecuteTime" : "2022-01-23 16:05:08",
    "ProcessID" : 2.0210608175202173E35,
    "User" : "test_user"
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.11. 账号管理

9.11.1. CreateAccount

调用CreateAccount接口创建账号。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreateAccount	系统规定参数，取值： CreateAccount。
DBClusterId	String	是	am-bp1r053byu48p****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
AccountName	String	是	test_accout	数据库账号名。
AccountPassword	String	是	Test_accout1	数据库账号名的密码。 • 密码由大写字母、小写字母、数字以及特殊符号组成。 • 特殊符号包含：（!）、（@）、（#）、（$）、（%）、（^）、（&）、（*）（()）、（_）、（+）、（-）、（=） • 密码长度为8字符~32个字符。
AccountDescription	String	否	数据库测试账号	账号描述。 • 不能以 `http://` 或者 `https://` 开头； • 长度不超过256个字符。
AccountType	String	否	Normal	账号类型，取值说明： • **Normal**：普通账号。 • **Super**：高权限账号。

返回数据

名称	类型	示例值	描述
TaskId	Integer	1564657730	任务ID。
RequestId	String	2FED790E-FB61-4721-8C1C-07C627FA5A19	请求ID。
DBClusterId	String	am-bp1r053byu48p****	集群ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=CreateAccount
&DBClusterId=am-bp1r053byu48p****
&AccountName=test_accout
&AccountPassword=Test_accout1
&AccountDescription=数据库测试账号
&AccountType=Normal
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<CreateAccountResponse>
    <TaskId>1564657730</TaskId>
    <RequestId>2FED790E-FB61-4721-8C1C-07C627FA5A19</RequestId>
    <DBClusterId>am-bp1r053byu48p****</DBClusterId>
</CreateAccountResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TaskId" : 1564657730,
  "RequestId" : "2FED790E-FB61-4721-8C1C-07C627FA5A19",
  "DBClusterId" : "am-bp1r053byu48p****"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.11.2. DeleteAccount

调用DeleteAccount接口删除数据库账号。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteAccount	系统规定参数。取值：DeleteAccount。
DBClusterId	String	是	rm-uf6wjk5xxxxxxxxxx	实例ID。
AccountName	String	是	test_accout	数据库账号名。

名称	类型	是否必选	示例值	描述
AccountType	String	否	Normal	• Normal：普通账号。 • Super：高权限账号。

返回数据

名称	类型	示例值	描述
RequestId	String	2FED790E-FB61-4721-8C1C-07C627FA5A19	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteAccount
&AccountName=test_accout
&DBClusterId=rm-uf6wjk5xxxxxxxxxx
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<RequestId>2FED790E-FB61-4721-8C1C-07C627FA5A19</RequestId>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "2FED790E-FB61-4721-8C1C-07C627FA5A19"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.11.3. DescribeAccounts

调用DescribeAccounts接口查询集群的账号信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeAccounts	系统规定参数，取值： DescribeAccounts。
DBClusterId	String	是	rm-uf6wjk5xxxxxxxxxx	实例ID。
AccountName	String	否	test_accout	数据库账号名。
AccountType	String	否	Normal	• Normal：普通账号。 • Super：高权限账号。

返回数据

名称	类型	示例值	描述
AccountList	Array of DBAccount		数据库账号列表。
DBAccount			
AccountDescription	String	测试数据库账号	账号备注。
AccountName	String	test1	账号名。
AccountStatus	String	Available	账号状态。 • Creating：创建中。 • Available：可用。 • Deleting：删除中。
AccountType	String	Normal	• Normal：普通账号。 • Super：高权限账号。
RequestId	String	64E37E6F-C363-41F3-867A-70EF5DC60EA4	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeAccounts
&DBClusterId=rm-uf6wjk5xxxxxxxxxx
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>64E37E6F-C363-41F3-867A-70EF5DC60EA4</RequestId>
<AccountList>
    <DBAccount>
        <AccountDescription>测试数据库账号</AccountDescription>
        <AccountStatus>Available</AccountStatus>
        <AccountType>Normal</AccountType>
        <AccountName>test1</AccountName>
    </DBAccount>
</AccountList>
```

`JSON` 格式

```
{
    "RequestId":"64E37E6F-C363-41F3-867A-70EF5DC60EA4",
    "AccountList":{
        "DBAccount":[
            {
                "AccountDescription":"测试数据库账号",
                "AccountStatus":"Available",
                "AccountType":"Normal",
                "AccountName":"test1"
            }
        ]
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.11.4. DescribeOperatorPermission

调用DescribeOperatorPermission接口查询集群服务账号的授权详情。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeOperatorPermission	系统规定参数，取值：DescribeOperatorPermission。
DBClusterId	String	是	rm-uf6wjk5xxxxxxxxxx	实例ID。

返回数据

名称	类型	示例值	描述
CreatedTime	String	2019-05-20T05:41:19Z	授权生效时间。
DBClusterId	String	rm-uf6wjk5xxxxxxxxxx	实例ID。
ExpiredTime	String	2019-05-20T07:41:19Z	授权过期时间。
Privileges	String	Control,Data	已获得的授权类型：Control|Data。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeOperatorPermission
&DBClusterId=rm-uf6wjk5xxxxxxxxxx
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<CreatedTime>2019-05-20T05:41:19Z</CreatedTime>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<Privileges>Control,Data</Privileges>
<ExpiredTime>2019-05-20T07:41:19Z</ExpiredTime>
<DBClusterId>rm-uf6wjk5xxxxxxxxxx</DBClusterId>
```

`JSON` 格式

```
{
    "CreatedTime": "2019-05-20T05:41:19Z",
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "Privileges": "Control,Data",
    "ExpiredTime": "2019-05-20T07:41:19Z",
    "DBClusterId": "rm-uf6wjk5xxxxxxxxxx"
}
```

错误码

访问错误中心查看更多错误码。

9.11.5. GrantOperatorPermission

调用GrantOperatorPermission接口为集群服务账号授权。

当您在使用AnalyticDB for MySQL集群过程中需要阿里云技术支持时，如果技术支持过程中需要对您的集群进行操作。您需要授权AnalyticDB for MySQL集群的服务账号，技术支持人员才可以通过服务账号提供技术支持服务。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GrantOperatorPermission	系统规定参数，取值：GrantOperatorPermission。
DBClusterId	String	是	rm-uf6wjk5xxxxxxxxxx	实例ID。
ExpiredTime	String	是	2019-05-20T05:41:19Z	权限过期时间。
Privileges	String	是	Control	授权类型：Control|Data。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GrantOperatorPermission
&DBClusterId=rm-uf6wjk5xxxxxxxxxx
&ExpiredTime=2019-05-20T05:41:19Z
&Privileges=Control
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

访问错误中心查看更多错误码。

9.11.6. ResetAccountPassword

调用ResetAccountPassword接口重置数据库账号。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ResetAccountPassword	系统规定参数。取值：ResetAccountPassword。
DBClusterId	String	是	rm-uf6wjk5xxxxxxxxxx	实例ID。
AccountName	String	是	test_accout	数据库账号名。
AccountPassword	String	是	Test_accout1	数据库账号名密码。 • 密码由大写字母、小写字母、数字以及特殊符号组成。 • 特殊符号包含：（!）、（@）、（#）、（$）、（%）、（^）、（&）、（*）（()）、（_）、（+）、（-）、（=） • 密码长度为8字符~32个字符。

名称	类型	是否必选	示例值	描述
AccountType	String	否	Normal	1. Normal：普通账号； 2. Super：高权限账号。

返回数据

名称	类型	示例值	描述
TaskId	Integer	1564657730	任务ID。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
DBClusterId	String	rm-uf6wjk5xxxxxxxxxx	实例ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ResetAccountPassword
&AccountName=test_accout
&AccountPassword=Test_accout1
&DBClusterId=rm-uf6wjk5xxxxxxxxxx
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<TaskId>1564657730</TaskId>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<DBClusterId>rm-uf6wjk5xxxxxxxxxx</DBClusterId>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TaskId" : 1564657730,
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
  "DBClusterId" : "rm-uf6wjk5xxxxxxxxxx"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.11.7. RevokeOperatorPermission

调用RevokeOperatorPermission接口撤销集群服务账号权限。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	RevokeOperatorPermission	系统规定参数，取值：RevokeOperatorPermission。
DBClusterId	String	是	rm-uf6wjk5xxxxxxxxxx	实例ID。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=RevokeOperatorPermission
&DBClusterId=rm-uf6wjk5xxxxxxxxxx
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

访问错误中心查看更多错误码。

9.11.8. ModifyAccountDescription

调用ModifyAccountDescription接口修改ADB数据库账号的备注信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ModifyAccountDescription	系统规定参数，取值：ModifyAccountDescription。
AccountDescription	String	是	AccDesc	修改账号备注： • 以中文、英文字母开头。 • 可以包含中文、英文字符、数字、”_”、”-”。 • 不能以 `http://` 、 `https://` 开头。 • 长度为2-256个字符。
AccountName	String	是	testacc	账号名。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ModifyAccountDescription
&AccountDescription=AccDesc
&AccountName=testacc
&DBClusterId=am-bp1xxxxxxxx47
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.11.9. DescribeAllAccounts

调用DescribeAllAccounts接口查询指定集群、指定数据库的账号列表信息或某个指定账号的信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeAllAccounts	系统规定参数，取值：DescribeAllAccounts。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。

返回数据

名称	类型	示例值	描述
AccountList	Array of AccountInfo		账号列表。
User	String	rdsdt_dts_adb	账号名。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeAllAccounts
&DBClusterId=am-bp1xxxxxxxx47
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<AccountList>
    <User>rdsdt_dts_adb</User>
</AccountList>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "AccountList": {
        "User": "rdsdt_dts_adb"
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.12. 标签管理

9.12.1. TagResources

调用TagResources接口为AnalyticDB MySQL版集群绑定标签。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	TagResources	系统规定参数，取值：**TagResources**。
RegionId	String	是	cn-hangzhou	地域ID，您可通过接口DescribeRegions查看可用的地域ID。
ResourceId.N	RepeatList	是	am-bp1gfds6a32s9****	需要绑定标签的集群ID。一次最多为50个集群同时绑定标签。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL版集群的详情，包括集群ID。
ResourceType	String	是	ALIYUN::ADB::CLUSTER	集群类型，取值：**ALIYUN::ADB::CLUSTER**。
Tag.N.Key	String	是	testkey1	标签键。一次可添加多个标签，每个集群最多可添加20对标签。
Tag.N.Value	String	是	testvalue1	标签键对应的标签值。一次可添加多个标签，每个集群最多可添加20对标签。

返回数据

名称	类型	示例值	描述
RequestId	String	863D51B7-5321-41D8-A0B6-A088B0450EFD	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=TagResources
&RegionId=cn-hangzhou
&ResourceId.1=am-bp1gfds6a32s9****
&ResourceType=ALIYUN::ADB::CLUSTER
&Tag.1.Value=testkey1
&Tag.1.Key=testvalue1
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>863D51B7-5321-41D8-A0B6-A088B0450EFD</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "863D51B7-5321-41D8-A0B6-A088B0450EFD"
}
```

错误码

访问错误中心查看更多错误码。

9.12.2. ListTagResources

调用ListTagResources查询一个或多个AnalyticDB MySQL集群已绑定的标签列表，或者查询一个或多个标签绑定的AnalyticDB MySQL集群列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListTagResources	系统规定参数。取值：**ListTagResources**。
RegionId	String	是	cn-hangzhou	地域ID，您可通过接口DescribeRegions查看可用的地域ID。
ResourceType	String	是	cluster	资源类型定义。唯一取值：**cluster**。
NextToken	String	否	212db86sca4384811e0b5e8707ec21345	用来返回更多结果。第一次查询不需要提供这个参数，如果一次查询没有返回全部结果，则可在后续查询中传入前一次返回的Token继续查询。
ResourceId.N	String	否	am-bp1l20nxxxxxxxxxx	集群ID。支持同时传入多个集群ID，N的取值范围为：1~50。 说明 ResourceId.N参数和Tag.N.Key参数至少传入一个。
Tag.N.Key	String	否	testkey1	标签键。支持同时传入多个标签键，不允许传入空字符串。N的取值范围为：1~20。 说明 ResourceId.N参数和Tag.N.Key参数至少传入一个。
Tag.N.Value	String	否	testvalue1	标签键对应的标签值。支持同时传入多个标签值，允许传入空字符串。N的取值范围为：1~20。

返回数据

名称	类型	示例值	描述
NextToken	String	212db86sca4384811e0b5e8707ec21345	用来返回更多结果。如果一次查询没有返回全部结果，则可在后续查询中传入前一次返回的token继续查询。
RequestId	String	184DE106-CB2C-4DD2-B57F-396652E6C8F8	请求ID。
TagResources	Array of TagResource		查询到的集群和标签的信息。
TagResource			
ResourceType	String	cluster	资源类型，cluster即云原生数据仓库AnalyticDB MySQL集群。
TagValue	String	testvalue1	标签键对应的标签值。
ResourceId	String	am-bp1l20nxxxxxxxxxx	集群ID。
TagKey	String	testkey1	标签键。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=ListTagResources
&RegionId=cn-hangzhou
&ResourceType=cluster
&NextToken=212db86sca4384811e0b5e8707ec21345
&ResourceId=["am-bp1l20nxxxxxxxxxx"]
&Tag=[{"Key":"testkey1","Value":"testvalue1"}]
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ListTagResourcesResponse>
    <NextToken>212db86sca4384811e0b5e8707ec21345</NextToken>
    <RequestId>184DE106-CB2C-4DD2-B57F-396652E6C8F8</RequestId>
    <TagResources>
        <ResourceType>cluster</ResourceType>
        <TagValue>testvalue1</TagValue>
        <ResourceId>am-bp1l20nxxxxxxxxxx</ResourceId>
        <TagKey>testkey1</TagKey>
    </TagResources>
</ListTagResourcesResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "NextToken" : "212db86sca4384811e0b5e8707ec21345",
  "RequestId" : "184DE106-CB2C-4DD2-B57F-396652E6C8F8",
  "TagResources" : {
    "ResourceType" : "cluster",
    "TagValue" : "testvalue1",
    "ResourceId" : "am-bp1l20nxxxxxxxxxx",
    "TagKey" : "testkey1"
  }
}
```

错误码

访问错误中心查看更多错误码。

9.12.3. UntagResources

调用UntagResources接口将标签从AnalyticDB MySQL集群中解绑，如果该标签没有绑定到其他集群，则该标签会被删除。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	UntagResources	系统规定参数，取值：UntagResources。
RegionId	String	是	cn-hangzhou	地域ID，您可通过接口DescribeRegions查看可用的地域ID。
ResourceId.N	RepeatList	是	am-xxxxx	第n个需要删除标签的集群ID。每次最多为50个集群同时删除标签，各个集群ID后面的数字n必须不同，且必须是从1开始的连续整数。例如：ResourceId.1=am-xxxxx，ResourceId.2=am-XXXXX。
ResourceType	String	是	ALIYUN::ADB::CLUSTER	集群类型，取值：`ALIYUN::ADB::CLUSTER`。
TagKey.N	RepeatList	否	a	标签键。每次最多删除20对标签，各个标签对的数字n必须不同，且必须是从1开始的连续整数。

名称	类型	是否必选	示例值	描述
All	Boolean	否	false	是否全部删除，仅当TagKey.n为空时有效。取值范围： • true • false 默认值为：false。

返回数据

名称	类型	示例值	描述
RequestId	String	2D69A58F-345C-4FDE-88E4-BF5189484043	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=UntagResources
&RegionId=cn-hangzhou
&ResourceId.1=am-xxxxx
&ResourceType=ALIYUN::ADB::CLUSTER
&&TagKey.1=a
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>2D69A58F-345C-4FDE-88E4-BF5189484043</RequestId>
```

`JSON` 格式

```
{
    "RequestId": "2D69A58F-345C-4FDE-88E4-BF5189484043"
}
```

错误码

访问错误中心查看更多错误码。

9.13. 备份恢复

9.13.1. DescribeBackups

调用DescribeBackups接口查看集群的备份列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeBackups	系统规定参数，取值： DescribeBackups。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。
EndTime	String	是	2011-06-15T16:00Z	查询结束时间，需要大于查询开始时间，格式：yyyy-MM-ddTHH:mmZ。
StartTime	String	是	2011-06-01T16:00Z	查询开始时间，格式：yyyy-MM-ddTHH:mmZ。
BackupId	String	否	327329803	备份集ID。

名称	类型	是否必选	示例值	描述
PageSize	Integer	否	30	每页记录数，取值： • 30； • 50； • 100 默认值：30。
PageNumber	Integer	否	1	页码，取值：大于0且不超过Integer的最大值。 默认值：1。

返回数据

名称	类型	示例值	描述
Items	Array of Backup		备份集列表。
Backup			
BackupEndTime	String	2011-05-30T03:29:00Z	本次备份结束时间，格式：YYYY-MM-DDTHH:mm:ssZ。
BackupId	String	327329803	备份集ID。
BackupMethod	String	Snapshot	备份方法。 Snapshot：快照备份
BackupSize	Integer	2167808	备份文件大小，单位：Byte。
BackupStartTime	String	2011-05-30T03:29:00Z	本次备份开始时间，格式：YYYY-MM-DDTHH:mm:ssZ。
BackupType	String	FullBackup	备份类型。取值： • FullBackup：全量备份。 • IncrementalBackup：增量备份。
DBClusterId	String	am-bp1xxxxxxxx47	实例ID。
PageNumber	String	10	页码。
PageSize	String	30	本页备份集个数。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
TotalCount	String	300	总记录数。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeBackups
&DBClusterId=am-bp1xxxxxxxx47
&EndTime=2011-06-15T16:00Z
&StartTime=2011-06-01T16:00Z
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<TotalCount>300</TotalCount>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<PageSize>30</PageSize>
<PageNumber>10</PageNumber>
<Items>
    <Backup>
        <BackupMethod>Snapshot</BackupMethod>
        <BackupEndTime>2011-05-30T03:29:00Z</BackupEndTime>
        <BackupSize>2167808</BackupSize>
        <DBClusterId>am-bp1xxxxxxxx47 </DBClusterId>
        <BackupStartTime>2011-05-30T03:29:00Z</BackupStartTime>
        <BackupId>327329803</BackupId>
        <BackupType>FullBackup</BackupType>
    </Backup>
</Items>
```

`JSON` 格式

```
{
    "TotalCount": 300,
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "PageSize": 30,
    "PageNumber": 10,
    "Items": {
        "Backup": {
            "BackupMethod": "Snapshot",
            "BackupEndTime": "2011-05-30T03:29:00Z",
            "BackupSize": 2167808,
            "DBClusterId": "am-bp1xxxxxxxx47",
            "BackupStartTime": "2011-05-30T03:29:00Z",
            "BackupId": 327329803,
            "BackupType": "FullBackup"
        }
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.13.2. DescribeBackupPolicy

调用DescribeBackupPolicy接口查看集群备份设置。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeBackupPolicy	系统规定参数，取值： DescribeBackupPolicy。
DBClusterId	String	是	am-bp1xxxxxxxx47	实例ID。

返回数据

名称	类型	示例值	描述
BackupRetentionPeriod	Integer	7	数据备份保留天数。
EnableBackupLog	String	true	是否开启。 • true：开启。 • false：关闭。

名称	类型	示例值	描述
LogBackupRetentionPeriod	Integer	7	日志备份保留天数。
PreferredBackupPeriod	String	Tuesday,Friday	数据备份周期，多个取值用英文逗号（,）隔开。取值： • Monday：周一； • Tuesday：周二； • Wednesday：周三； • Thursday：周四； • Friday：周五； • Saturday：周六； • Sunday：周日。
PreferredBackupTime	String	15:00Z-16:00Z	数据备份时间，格式：HH:mmZ-HH:mmZ。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeBackupPolicy
&DBClusterId=am-bp1xxxxxxxx47
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<PreferredBackupPeriod>Tuesday,Friday</PreferredBackupPeriod>
<LogBackupRetentionPeriod>7</LogBackupRetentionPeriod>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<PreferredBackupTime>15:00Z-16:00Z</PreferredBackupTime>
<EnableBackupLog>true</EnableBackupLog>
<BackupRetentionPeriod>7</BackupRetentionPeriod>
```

`JSON` 格式

```
{
    "PreferredBackupPeriod": "Tuesday,Friday",
    "LogBackupRetentionPeriod": 7,
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "PreferredBackupTime": "15:00Z-16:00Z",
    "EnableBackupLog": true,
    "BackupRetentionPeriod": 7
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.13.3. ModifyBackupPolicy

调用ModifyBackupPolicy接口修改AnalyticDB MySQL版实例的备份策略。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ModifyBackupPolicy	系统规定参数。取值：**ModifyBackupPolicy**。
DBClusterId	String	是	am-bp1xxxxxxxx47	集群ID。

名称	类型	是否必选	示例值	描述
PreferredBackupTime	String	是	00:00Z-01:00Z	执行全量备份的开始时间，格式为HH:mmZ-HH:mmZ（UTC时间）。 说明 时间范围限制为1小时。
PreferredBackupPeriod	String	是	Monday,Wednesday,Friday,Sunday	全量备份周期，多个取值用英文逗号（,）隔开。取值说明： • Monday：周一。 • Tuesday：周二。 • Wednesday：周三。 • Thursday：周四。 • Friday：周五。 • Saturday：周六。 • Sunday：周日。 说明 为保证数据安全，最少选择两个。
BackupRetentionPeriod	String	否	30	全量备份保留天数，取值范围为：7~730。 说明 本参数不填写，默认为7天。
EnableBackupLog	String	否	Enable	是否开启日志（实时）备份。取值说明： • Enable：开启。 • Disable：关闭。 说明 本参数不填写，默认开启。
LogBackupRetentionPeriod	Integer	否	30	日志（实时）数据备份保留天数，取值范围为：7~730。 说明 本参数不填写，默认为7天。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ModifyBackupPolicy
&DBClusterId=am-bp1xxxxxxxx47
&PreferredBackupTime=00:00Z-01:00Z
&PreferredBackupPeriod=Monday,Wednesday,Friday,Sunday
&BackupRetentionPeriod=30
&EnableBackupLog=Enable
&LogBackupRetentionPeriod=30
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ModifyBackupPolicyResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
</ModifyBackupPolicyResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.13.4. ModifyLogBackupPolicy

调用ModifyLogBackupPolicy接口修改日志备份设置。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ModifyLogBackupPolicy	系统规定参数。取值：**ModifyLogBackupPolicy**。
DBClusterId	String	是	am-bp1xxxxxxxx47	集群ID。
EnableBackupLog	String	是	Enable	是否开启日志备份。取值说明： • **Enable**：开启。 • **Disable**：关闭。
LogBackupRetentionPeriod	String	否	30	数据备份保留天数，取值范围：7~730。 说明 此参数不填写，默认为7天。

返回数据

名称	类型	示例值	描述
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ModifyLogBackupPolicy
&DBClusterId=am-bp1xxxxxxxx47
&EnableBackupLog=Enable
&LogBackupRetentionPeriod=30
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ModifyLogBackupPolicyResponse>
    <RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
</ModifyLogBackupPolicyResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1AD222E9-E606-4A42-BF6D-8A4442913CEF"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.14. 安全管理

9.14.1. DescribeDBClusterAccessWhiteList

调用DescribeDBClusterAccessWhiteList接口查看集群的IP白名单。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDBClusterAccessWhiteList	系统规定参数，取值： DescribeDBClusterAccessWhiteList。
DBClusterId	String	是	am-uf6wjk5xxxxxxxxx	实例ID。

返回数据

名称	类型	示例值	描述
Items	Array of IPArray		IP白名单分组列表。
IPArray			
DBClusterIPArrayAttribute	String	hidden	白名单分组属性，默认为空。 说明 控制台不显示带有“hidden”属性的分组，带有该标签的分组通常是用于访问DTS、DRDS服务的。
DBClusterIPArrayName	String	test	IP白名单分组名称。 • IP白名单分组名称以小写字母开头、以数字或小写字母结尾，可包含小写字母、数字及'_'，长度为2~32。 • 单个实例最多支持50个白名单分组。
SecurityIPList	String	127.0.0.1	IP白名单分组下的IP列表，最多1000个，以英文逗号（,）隔开。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeDBClusterAccessWhiteList
&DBClusterId=rm-uf6wjk5xxxxxxxxx
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<Items>
    <IPArray>
        <DBClusterIPArrayName>test</DBClusterIPArrayName>
        <SecurityIPList>127.0.0.1</SecurityIPList>
        <DBClusterIPArrayAttribute>hidden</DBClusterIPArrayAttribute>
    </IPArray>
</Items>
```

`JSON` 格式

```
{
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "Items": {
        "IPArray": {
            "DBClusterIPArrayName": "test",
            "SecurityIPList": "127.0.0.1",
            "DBClusterIPArrayAttribute": "hidden"
        }
    }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.14.2. ModifyDBClusterAccessWhiteList

调用ModifyDBClusterAccessWhiteList接口修改集群白名单。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ModifyDBClusterAccessWhiteList	系统规定参数，取值： ModifyDbClusterAccessWhiteList。
DBClusterId	String	是	rm-uf6wjk5xxxxxxxxxx	实例ID。
SecurityIps	String	是	10.23.12.24	集群的IP白名单，多个IP地址之间以英文逗号（,）隔开，不可重复，最多500个。支持如下两种格式： • IP地址形式，例如：10.23.12.24。 • CIDR形式，例如：10.23.12.24/24（无类域间路由，24表示了地址中前缀的长度，范围为1~32）。 ModifyMode值为Delete时，此参数值可以是空，其他情况下必须有非空值。
DBClusterIPArrayName	String	否	test	需要修改的IP白名单分组名称，默认操作“Default”分组。 IP白名单分组名称以小写字母开头、以数字或小写字母结尾，可包含小写字母、数字及'_'，长度为2~32。 单个实例最多支持50个白名单分组。
DBClusterIPArrayAttribute	String	否	hidden	白名单分组属性，默认为空。 控制台不显示带有“hidden”属性的分组，带有该标签的分组通常是用于访问DTS、DRDS服务的。

名称	类型	是否必选	示例值	描述
ModifyMode	String	否	Cover	修改白名单的方式。取值： • Cover：覆盖原IP白名单； • Append：追加IP； • Delete：删除IP。 默认值：Cover。

返回数据

名称	类型	示例值	描述
DBClusterId	String	rm-uf6wjk5xxxxxxxxxx	实例ID。
RequestId	String	D0CEC6AC-7760-409A-A0D5-E6CD8660E9CC	请求ID。
TaskId	Integer	1564657730	任务ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ModifyDBClusterAccessWhiteList
&DBClusterId=rm-uf6wjk5xxxxxxxxxx
&SecurityIps=10.23.12.24
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<TaskId>1564657730</TaskId>
<RequestId>D0CEC6AC-7760-409A-A0D5-E6CD8660E9CC</RequestId>
<DBClusterId>rm-uf6wjk5xxxxxxxxxx</DBClusterId>
```

`JSON` 格式

```
{
    "TaskId": 1564657730,
    "RequestId": "D0CEC6AC-7760-409A-A0D5-E6CD8660E9CC",
    "DBClusterId": "rm-uf6wjk5xxxxxxxxxx"
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.15. 监控管理

9.15.1. DescribeDBClusterPerformance

调用DescribeDBClusterPerformance接口查看目标集群的性能数据。

根据性能参数查看某个集群、某时间段范围内的性能监控数据。采集粒度为30秒。支持对慢SQL进行查询，提供SQL查询时长、扫描行数、扫描量等监控内容。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDBClusterPerformance	系统规定参数。取值：**DescribeDBClusterPerformance**。

名称	类型	是否必选	示例值	描述
DBClusterId	String	是	am-************	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL版集群的详情，包括集群ID。

Key	String	否	AnalyticDB_CPU	需要查询的性能指标名称，多个值之间用英文逗号（,）分隔。支持查询如下性能指标：
				• CPU
				  ◦ **AnalyticDB_CPU**：CPU平均使用率。
				• 连接数
				  ◦ **AnalyticDB_Connections**：数据库连接数。
				• 写入
				  ◦ **AnalyticDB_TPS**：写入TPS。
				  ◦ **AnalyticDB_InsertRT**：写入响应时间。
				  ◦ **AnalyticDB_InsertBytes**：写入吞吐量。
				• 更新
				  ◦ **AnalyticDB_UpdateRT**：更新响应时间。
				• 删除
				  ◦ **AnalyticDB_DeleteRT**：删除响应时间。
				• 查询
				  ◦ **AnalyticDB_QPS**：查询QPS。
				  ◦ **AnalyticDB_QueryRT**：查询响应时间。
				  ◦ **AnalyticDB_QueryWaitTime**：查询等待耗时。
				• 磁盘
				  ◦ **AnalyticDB_IO**：磁盘IO吞吐。
				  ◦ **AnalyticDB_IO_UTIL**：IO Util。
				  ◦ **AnalyticDB_IO_WAIT**：IO Wait。
				  ◦ **AnalyticDB_IOPS**：磁盘IOPS。
				  ◦ **AnalyticDB_DiskUsage**：磁盘用量。
				  ◦ **AnalyticDB_HotDataDiskUsage**：热数据磁盘用量。
				  ◦ **AnalyticDB_ColdDataDiskUsage**：冷数据磁盘用量。
				说明 若该参数留空，默认返回上述所有性能指标的值。

StartTime	String	是	2021-05-03T15:00Z	查询开始时间。格式为*yyyy-MM-ddTHH:mmZ*（UTC时间）。
EndTime	String	是	2021-05-03T15:01Z	查询结束时间。格式为*yyyy-MM-ddTHH:mmZ*（UTC时间）。 说明 查询结束时间需晚于开始时间，且开始时间和结束时间的时间间隔不能超过两天。

返回数据

名称	类型	示例值	描述
EndTime	String	2021-05-03T15:01:00Z	查询结束时间。格式为*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
RequestId	String	25B56BC7-4978-40B3-9E48-4B7067******	请求ID。
StartTime	String	2021-05-03T15:00:00Z	查询开始时间。格式为*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
DBClusterId	String	am-************	集群ID。
Performances	Array of PerformanceItem		集群性能指标列表。

名称	类型	示例值	描述
Key	String	AnalyticDB_CPU	性能指标名称。
Unit	String	%	数据单位。
Series	Array of SeriesItem		性能数据列表。
Values	Array of String	["2021-05-03T15:00:12.72Z", "0.1250"], ["2021-05-03T15:00:42.739Z", "0.3125"]	不同时间点下的性能值详情。
Name	String	worker_avg_cpu_used	性能值名称。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDBClusterPerformance
&DBClusterId=am-************
&Key=AnalyticDB_CPU
&StartTime=2021-06-01T15:00Z
&EndTime=2012-06-18T15:00:00Z
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDBClusterPerformanceResponse>
    <RequestId>25B56BC7-4978-40B3-9E48-4B7067******</RequestId>
    <EndTime>2021-05-03T15:01:00Z</EndTime>
    <DBClusterId>am-bp**************</DBClusterId>
    <StartTime>2021-05-03T15:00:00Z</StartTime>
    <Performances>
        <Series>
            <Values>
                <0>2021-05-03T15:00:12.72Z</0>
                <1>0.1875</1>
            </Values>
            <Values>
                <0>2021-05-03T15:00:42.739Z</0>
                <1>0.1250</1>
            </Values>
            <Name>worker_avg_cpu_used</Name>
        </Series>
        <Series>
            <Values>
                <0>2021-05-03T15:00:12.72Z</0>
                <1>0.1250</1>
            </Values>
            <Values>
                <0>2021-05-03T15:00:42.739Z</0>
                <1>0.3125</1>
            </Values>
            <Name>executor_avg_cpu_used</Name>
        </Series>
        <Unit>%</Unit>
        <Key>AnalyticDB_CPU</Key>
    </Performances>
</DescribeDBClusterPerformanceResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "25B56BC7-4978-40B3-9E48-4B7067******",
  "EndTime" : "2021-05-03T15:01:00Z",
  "DBClusterId" : "am-bp**************",
  "StartTime" : "2021-05-03T15:00:00Z",
  "Performances" : [ {
    "Series" : [ {
      "Values" : [ [ "2021-05-03T15:00:12.72Z", "0.1875" ], [ "2021-05-03T15:00:42.739Z", "0.1250" ] ],
      "Name" : "worker_avg_cpu_used"
    }, {
      "Values" : [ [ "2021-05-03T15:00:12.72Z", "0.1250" ], [ "2021-05-03T15:00:42.739Z", "0.3125" ] ],
      "Name" : "executor_avg_cpu_used"
    } ],
    "Unit" : "%",
    "Key" : "AnalyticDB_CPU"
  } ]
}
```

错误码

访问错误中心查看更多错误码。

9.15.2. DescribeDBClusterResourcePoolPerformance

调用DescribeDBClusterResourcePoolPerformance接口查看AnalyticDB MySQL版弹性模式集群版（新版）的资源池监控信息。

> **说明** 您还可以在控制台上以图表的形式查看AnalyticDB MySQL版弹性模式集群版（新版）资源池监控信息。更多详细信息，请参见查看监控信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDBClusterResourcePoolPerformance	系统规定参数。取值：**DescribeDBClusterResourcePoolPerformance**。
DBClusterId	String	是	am-****************	集群ID。 **说明** 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。

名称	类型	是否必选	示例值	描述
Key	String	否	AnalyticDB_RP_CPU	资源池数据监控指标。允许同时输入多个指标进行查询，多个指标间用英文逗号（,）分隔。支持监控如下指标： • **AnalyticDB_RP_CPU**：CPU平均使用率，单位：%。 • **AnalyticDB_RP_RT**：查询响应时间（RT），单位：毫秒（ms）。 • **AnalyticDB_RP_QPS**：每秒查询率（QPS），单位：数值。 • **AnalyticDB_RP_WaitTime**：查询等待时间，单位：毫秒（ms）。 • **AnalyticDB_RP_OriginalNode**：资源池基础节点数。 • **AnalyticDB_RP_ActualNode**：资源池分时弹性实际弹出的节点数（即执行扩容计划时实际增加的节点数）。 • **AnalyticDB_RP_PlanNode**：资源池分时弹性计划弹出的节点数。 • **AnalyticDB_RP_TotalNode**：资源池拥有的总节点数，总节点数=基础节点数+分时弹性计划中实际生效的节点数。 说明 • 若该参数留空，默认返回所有指标的监控数据详情。 • 更多关于资源池弹性计划的信息，请参见资源弹性扩容。
ResourcePools	String	否	TEST_POOL	目标资源池名称。允许同时输入多个资源池名称进行查询，多个名称间用英文逗号（,）分隔。 说明 • 输入的资源池名称不区分大小写，例如 `USER_DEFAULT` 与 `user_default` 表示的是同一个资源池。 • 若该参数留空，将返回 `USER_DEFAULT` 即默认资源池的监控信息。
StartTime	String	是	2021-06-10T07:00Z	资源池数据监控开始时间。格式为*yyyy-MM-ddTHH:mmZ*（UTC时间）。 说明 仅支持查看最近2天的资源池监控信息。
EndTime	String	是	2021-06-10T07:01Z	资源池数据监控结束时间，结束时间需晚于开始时间。格式为*yyyy-MM-ddTHH:mmZ*（UTC时间）。

返回数据

名称	类型	示例值	描述
EndTime	String	2021-06-10T07:01:00Z	资源池数据监控结束时间，格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
RequestId	String	C7EDB8E4-9769-4233-88C7-DCA4C9******	请求ID。
StartTime	String	2021-06-10T07:00:00	资源池数据监控开始时间，格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
DBClusterId	String	am-****************	集群ID。
Performances	Array of PerformanceItem		以监控指标为维度的数据详情列表。
Key	String	AnalyticDB_RP_CPU	资源池数据监控指标。

名称	类型	示例值	描述
Unit	String	%	监控指标的单位。
ResourcePoolPerformances	Array of ResourcePoolPerformancesItem		以资源池为维度的监控数据列表。
ResourcePoolName	String	test_pool	资源池名称。
ResourcePoolSeries	Array of ResourcePoolSeriesItem		资源池监控数据的序列列表。
Values	Array of String	["2021-06-10T07:00:22.601Z","0.0000"],["2021-06-10T07:00:52.62Z","0.0312"]	不同时间点下的资源池监控项数值。
Name	String	cpu	监控指标名称。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDBClusterResourcePoolPerformance
&DBClusterId=am-****************
&StartTime=2021-06-10T07:00Z
&EndTime=2021-06-10T07:01Z
&公共请求参数
```

正常返回示例

`XML` **格式**

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDBClusterResourcePoolPerformanceResponse>
    <RequestId>C7EDB8E4-9769-4233-88C7-DCA4C9******</RequestId>
    <EndTime>2021-06-10T07:01:00Z</EndTime>
    <StartTime>2021-06-10T07:00:00Z</StartTime>
    <Performances>
        <ResourcePoolPerformances>
            <ResourcePoolSeries>
                <Values>
                    <0>2021-06-10T07:00:22.601Z</0>
                    <1>0.0000</1>
                </Values>
                <Values>
                    <0>2021-06-10T07:00:52.62Z</0>
                    <1>0.0312</1>
                </Values>
                <Name>cpu</Name>
            </ResourcePoolSeries>
            <ResourcePoolName>test_pool</ResourcePoolName>
        </ResourcePoolPerformances>
        <Unit>%</Unit>
        <Key>AnalyticDB_RP_CPU</Key>
    </Performances>
</DescribeDBClusterResourcePoolPerformanceResponse>
```

`JSON` **格式**

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "C7EDB8E4-9769-4233-88C7-DCA4C9******",
  "EndTime" : "2021-06-10T07:01:00Z",
  "StartTime" : "2021-06-10T07:00:00Z",
  "Performances" : [ {
    "ResourcePoolPerformances" : [ {
      "ResourcePoolSeries" : [ {
        "Values" : [ [ "2021-06-10T07:00:22.601Z", "0.0000" ], [ "2021-06-10T07:00:52.62Z", "0.0312" ] ],
        "Name" : "cpu"
      } ],
      "ResourcePoolName" : "test_pool"
    } ],
    "Unit" : "%",
    "Key" : "AnalyticDB_RP_CPU"
  } ]
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.15.3. DescribeInclinedTables

调用DescribeInclinedTables接口查看表监控。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeInclinedTables	系统规定参数，取值：DescribeInclinedTables。
DBClusterId	String	是	am-bpxxxxxxxx47	实例ID。
TableType	String	是	FactTable	表类型： • FactTable 分区表 • DimensionTable 维度表
Order	String	否	[{ "Field":"Name", "Type":"Asc" }]	按指定字段排序，json格式：JSON数组，有序，按顺序输入排列字段和升降序类型： 如： `[ { "Field":"Name", "Type":"Asc" } ]` 其中Field表示需要排序的字段名，目前仅支持对"Name" 字段排序。 Type表示排序类型 Desc为降序，Asc为升序 均不区分大小写
PageSize	Integer	否	30	每页记录数，取值： • 30； • 50； • 100； 默认值：30。

名称	类型	是否必选	示例值	描述
PageNumber	Integer	否	1	页码，取值为：大于0且不超过Integer数据类型的最大值，默认值为1。

返回数据

名称	类型	示例值	描述
Items	Array of Table		表列表。
Table			
IsIncline	Boolean	true	是否倾斜，反应表内分区分布是否倾斜。
Name	String	test	表名。
Schema	String	adb_demo	数据库名。
Size	Long	2	表行数。
Type	String	FactTable	表类型： • FactTable 分区表 • DimensionTable 维度表
PageNumber	String	1	页数。
PageSize	String	30	总页数。
RequestId	String	1AD222E9-E606-4A42-BF6D-8A4442913CEF	请求ID。
TotalCount	String	1	总记录数。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeInclinedTables
&DBClusterId=am-bpxxxxxxxx47
&TableType=FactTable
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<TotalCount>1</TotalCount>
<PageSize>30</PageSize>
<RequestId>1AD222E9-E606-4A42-BF6D-8A4442913CEF</RequestId>
<PageNumber>1</PageNumber>
<Items>
    <Table>
        <Type>FactTable</Type>
        <IsIncline>true</IsIncline>
        <Size>2</Size>
        <Schema>adb_demo</Schema>
        <Name>test</Name>
    </Table>
</Items>
```

`JSON` 格式

```
{
    "TotalCount": 1,
    "PageSize": 30,
    "RequestId": "1AD222E9-E606-4A42-BF6D-8A4442913CEF",
    "PageNumber": 1,
    "Items": {
        "Table": {
            "Type": "FactTable",
            "IsIncline": true,
            "Size": 2,
            "Schema": "adb_demo",
            "Name": "test"
        }
    }
}
```


错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.15.4. DescribeTableStatistics

调用DescribeTableStatistics接口查询目标AnalyticDB MySQL版集群中的表信息统计详情。

> **说明** 更多关于表信息统计的详情，请参见监控信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeTableStatistics	系统规定参数。取值：**DescribeTableStatistics**。
DBClusterId	String	是	am-********** > **说明** 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。	集群ID。
PageSize	Integer	否	30	每页记录数，取值为**30**（默认值）、**50**或**100**。
PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。默认值为**1**。

Order	String	否	[{"Field":"TableName","Type":"Asc"}]	根据指定字段进行排序，格式为JSON，是一个有序JSON数组，按输入数组的顺序进行复合排序，包含 `Field` 和 `Type` 两个字段，例如 `[{ "Field":"TableName", "Type":"Asc" }]`。其中：
				• `Field` 表示需要排序的字段名，当前仅支持对 `TableName` 字段进行排序。
				• `Type` 表示排序类型，支持如下取值（取值均不区分大小写）：
				  ◦ **Desc**：降序。
				  ◦ **Asc**：升序。

返回数据

名称	类型	示例值	描述
TotalCount	String	1	总记录数。

名称	类型	示例值	描述
PageSize	String	30	本页记录数。
RequestId	String	4C4433FF-5D3A-4C3E-A19C-6D93B2******	请求ID。
PageNumber	String	1	页数。
DBClusterId	String	am-****************	集群ID。
Items	Array of TableStatisticRecords		表统计信息列表。
TableStatisticRecords			
SchemaName	String	test_schema	数据库名称。
TableName	String	test_table	表名称。
RowCount	Long	3	表行数。
DataSize	Long	15592	表数据量，单位：Byte。
IndexSize	Long	3076	索引数据量，单位：Byte。
PrimaryKeyIndexSize	Long	16340	主键索引数据量，单位：Byte。
PartitionCount	Long	1	分区数。
ColdDataSize	Long	0	冷数据总量，单位：Byte。 说明 仅内核版本为3.1.3.4或以上集群才会返回该参数。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeTableStatistics
&DBClusterId=am-****************
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeTableStatisticsResponse>
    <TotalCount>1</TotalCount>
    <RequestId>4C4433FF-5D3A-4C3E-A19C-6D93B2******</RequestId>
    <PageSize>30</PageSize>
    <PageNumber>1</PageNumber>
    <DBClusterId>am-****************</DBClusterId>
    <Items>
        <TableStatisticRecords>
            <TableName>test_table</TableName>
            <ColdDataSize>0</ColdDataSize>
            <DataSize>15592</DataSize>
            <PrimaryKeyIndexSize>16340</PrimaryKeyIndexSize>
            <IndexSize>3076</IndexSize>
            <RowCount>3</RowCount>
            <PartitionCount>1</PartitionCount>
            <SchemaName>test_schema</SchemaName>
        </TableStatisticRecords>
    </Items>
</DescribeTableStatisticsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TotalCount" : 1,
  "RequestId" : "4C4433FF-5D3A-4C3E-A19C-6D93B2******",
  "PageSize" : 30,
  "PageNumber" : 1,
  "DBClusterId" : "am-****************",
  "Items" : {
    "TableStatisticRecords" : [ {
      "TableName" : "test_table",
      "ColdDataSize" : 0,
      "DataSize" : 15592,
      "PrimaryKeyIndexSize" : 16340,
      "IndexSize" : 3076,
      "RowCount" : 3,
      "PartitionCount" : 1,
      "SchemaName" : "test_schema"
    } ]
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
404	InvalidDBCluster.NotFound	The DBClusterId provided does not exist in our records.	您指定的 DBClusterId 不存在，请确认 DBClusterId 值是否正确。

访问错误中心查看更多错误码。

9.16. 运维事件

9.16.1. DescribeMaintenanceAction

调用DescribeMaintenanceAction接口查询运维事件的详情。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeMaintenanceAction	系统规定参数。取值：**DescribeMaintenanceAction**。
Region	String	是	cn-hangzhou	地域。取值如下： • 运维事件所在地域的地域ID，如 `cn-hangzhou` 。您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。 • 您也可以输入 `all` 查看当前账号下所有地域下的所有运维事件，当 `Region` 设置为 `all` 时， `TaskType` 也必须设置为 `all` 。
TaskType	String	是	rds_apsaradb_upgrade	运维事件的类型，取值如下： • **rds_apsaradb_upgrade**：数据库软件升级。 • **all**：查看当前账号下所有地域下的所有运维事件，当 `Region` 设置为 `all` 时， `TaskType` 也必须设置为 `all` 。
IsHistory	Integer	否	1	指定返回待处理或历史运维事件的详情，取值如下： • **0**：返回待处理运维事件的详情。 • **1**：返回历史运维事件的详情。 若该参数留空，默认返回待处理运维事件的详情。
PageSize	Integer	否	30	每页记录数，取值为**30**（默认值）、**50**或**100**。

名称	类型	是否必选	示例值	描述
PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。默认值为1。
RegionId	String	是	cn-hangzhou	运维事件所在地域的地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。

返回数据

名称	类型	示例值	描述
PageNumber	Integer	1	页码。
RequestId	String	E774C8A9-8819-4A09-9E91-07C078******	请求ID。
PageSize	Integer	30	每页记录数。
TotalRecordCount	Integer	2	总记录数。
Items	Array of Items		事件列表。

Status	String	SUCCESS	事件状态。
			• 若您设置 `IsHistory` 为0，则返回待处理运维事件状态的详情，取值范围如下：
			  ◦ **WAITING_MODIFY**：等待用户设置运维事件开始时间。
			  ◦ **WAITING**：运维事件正在等待处理。
			  ◦ **PROCESSING**：运维事件正在处理中（不支持修改处于该状态下事件的切换时间）。
			• 若您设置 `IsHistory` 为1，则返回历史运维事件状态的详情，取值范围如下：
			  ◦ **SUCCESS**：事件结束且执行成功。
			  ◦ **FAILED**：事件结束但执行失败。
			  ◦ **CANCEL**：事件已取消。

Deadline	String	2021-07-04T15:59:59Z	运维事件执行时间可调整范围的最晚时间，格式为 `yyyy-MM-ddTHH:mm:ssZ`（UTC时间）。
PrepareInterval	String	02:00:00	运维事件切换之前所需的准备时间，格式为 `HH:mm:ss`。
DBType	String	analyticdb	数据库的引擎类型。
StartTime	String	2021-07-03T04:00:00Z	后台执行该运维事件的时间点，格式为 `yyyy-MM-ddTHH:mm:ssZ`（UTC时间）。
TaskType	String	rds_apsaradb_upgrade	运维事件的类型。
DBVersion	String	3.0	数据库引擎版本号。
DBClusterId	String	am-**********	运维事件所涉及集群的集群ID。
ModifiedTime	String	2021-07-03T06:33:00Z	修改运维事件切换时间的时间点，格式为 `yyyy-MM-ddTHH:mm:ssZ`（UTC时间）。
Region	String	cn-hangzhou	运维事件所在地域的地域ID。

名称	类型	示例值	描述
ResultInfo	String	autoCancel	运维事件的执行结果。 **说明** 仅当 `Status`（状态）取值为**FAILED**（运维事件结束但执行失败）或**CANCEL**（事件已取消）时，返回该参数。
CreatedTime	String	2021-06-30T02:44:27Z	运维事件的创建时间，格式为 `yyyy-MM-ddTHH:mm:ssZ`（UTC时间）。
Id	Integer	11111	运维事件的ID。
SwitchTime	String	2021-07-03T06:00:00Z	运维事件预计的切换时间，格式为 `yyyy-MM-ddTHH:mm:ssZ`（UTC时间）。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeMaintenanceAction
&Region=cn-hangzhou
&TaskType=rds_apsaradb_upgrade
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeMaintenanceActionResponse>
    <TotalRecordCount>2</TotalRecordCount>
    <RequestId>E774C8A9-8819-4A09-9E91-07C078******</RequestId>
    <PageSize>30</PageSize>
    <PageNumber>1</PageNumber>
    <Items>
        <Status>SUCCESS</Status>
        <CreatedTime>2021-06-30T02:44:27Z</CreatedTime>
        <Deadline>2021-07-04T15:59:59Z</Deadline>
        <StartTime>2021-07-03T04:00:00Z</StartTime>
        <DBClusterId>am-****************</DBClusterId>
        <DBType>analyticdb</DBType>
        <DBVersion>3.0</DBVersion>
        <ModifiedTime>2021-07-03T06:33:00Z</ModifiedTime>
        <TaskType>rds_apsaradb_upgrade</TaskType>
        <PrepareInterval>02:00:00</PrepareInterval>
        <Region>cn-hangzhou</Region>
        <Id>11111</Id>
        <SwitchTime>2021-07-03T06:00:00Z</SwitchTime>
    </Items>
    <Items>
        <Status>CANCEL</Status>
        <CreatedTime>2021-06-25T06:32:45Z</CreatedTime>
        <Deadline>2021-06-29T15:59:59Z</Deadline>
        <StartTime>2021-06-26T16:10:00Z</StartTime>
        <DBClusterId>am-****************</DBClusterId>
        <DBType>analyticdb</DBType>
        <DBVersion>3.0</DBVersion>
        <ModifiedTime>2021-06-26T18:40:00Z</ModifiedTime>
        <TaskType>rds_apsaradb_upgrade</TaskType>
        <PrepareInterval>02:00:00</PrepareInterval>
        <Region>cn-hangzhou</Region>
        <Id>22222</Id>
        <ResultInfo>autoCancel</ResultInfo>
        <SwitchTime>2021-06-26T18:10:00Z</SwitchTime>
    </Items>
</DescribeMaintenanceActionResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "TotalRecordCount" : 2,
  "RequestId" : "E774C8A9-8819-4A09-9E91-07C078******",
  "PageSize" : 30,
  "PageNumber" : 1,
  "Items" : [ {
    "Status" : "SUCCESS",
    "CreatedTime" : "2021-06-30T02:44:27Z",
    "Deadline" : "2021-07-04T15:59:59Z",
    "StartTime" : "2021-07-03T04:00:00Z",
    "DBClusterId" : "am-****************",
    "DBType" : "analyticdb",
    "DBVersion" : 3,
    "ModifiedTime" : "2021-07-03T06:33:00Z",
    "TaskType" : "rds_apsaradb_upgrade",
    "PrepareInterval" : "02:00:00",
    "Region" : "cn-hangzhou",
    "Id" : 11111,
    "SwitchTime" : "2021-07-03T06:00:00Z"
  }, {
    "Status" : "CANCEL",
    "CreatedTime" : "2021-06-25T06:32:45Z",
    "Deadline" : "2021-06-29T15:59:59Z",
    "StartTime" : "2021-06-26T16:10:00Z",
    "DBClusterId" : "am-****************",
    "DBType" : "analyticdb",
    "DBVersion" : 3,
    "ModifiedTime" : "2021-06-26T18:40:00Z",
    "TaskType" : "rds_apsaradb_upgrade",
    "PrepareInterval" : "02:00:00",
    "Region" : "cn-hangzhou",
    "Id" : 22222,
    "ResultInfo" : "autoCancel",
    "SwitchTime" : "2021-06-26T18:10:00Z"
  } ]
}
```

错误码

访问错误中心查看更多错误码。

9.16.2. ModifyMaintenanceAction

调用ModifyMaintenanceAction接口修改待处理运维事件的切换时间。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ModifyMaintenanceAction	系统规定参数。取值：**ModifyMaintenanceAction**。
Ids	String	是	11111	目标待处理运维事件的ID。支持输入多个ID来批量修改切换时间，多个ID间用英文逗号（,）隔开。 **说明** • 您可以调用DescribeMaintenanceAction接口查看目标待处理运维事件的详情，包括ID。 • 仅待处理运维事件支持修改切换时间，历史运维事件不支持修改切换时间。关于待处理运维事件和历史运维事件的状态详情，请参见DescribeMaintenanceAction。
SwitchTime	String	是	2021-07-09T22:00:00Z	后台将在指定时间点执行目标待处理运维事件所对应的操作。格式为 `yyyy-MM-ddTHH:mm:ssZ`（UTC时间）。

名称	类型	是否必选	示例值	描述
RegionId	String	是	cn-hangzhou	目标待处理运维事件所在地域的地域ID。 说明 • 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。

返回数据

名称	类型	示例值	描述
Ids	String	11111	目标运维事件ID。
RequestId	String	7856CBE7-5BD0-4EE1-AC62-749392******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=ModifyMaintenanceAction
&Ids=11111
&SwitchTime=2021-07-09T22:00:00Z
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ModifyMaintenanceActionResponse>
    <Ids>11111</Ids>
    <RequestId>7856CBE7-5BD0-4EE1-AC62-749392******</RequestId>
</ModifyMaintenanceActionResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Ids" : "11111",
  "RequestId" : "7856CBE7-5BD0-4EE1-AC62-749392******"
}
```

错误码

访问错误中心查看更多错误码。

9.17. SQL诊断

9.17.1. DescribeDiagnosisRecords

调用DescribeDiagnosisRecords接口查看目标AnalyticDB MySQL版集群中符合指定检索条件的SQL语句摘要信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDiagnosisRecords	系统规定参数。取值：**DescribeDiagnosisRecords**。
DBClusterId	String	是	am-bp1r053byu48p****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。

名称	类型	是否必选	示例值	描述
StartTime	String	是	1632931200000	查询开始时间，格式为Unix时间戳，单位：毫秒。 说明 仅支持查看14天内的数据。
EndTime	String	是	1633017540000	查询结束时间，格式为Unix时间戳，单位：毫秒。 说明 • 查询结束时间需晚于查询开始时间。 • 开始时间与结束时间的间隔不能超过24小时。
RegionId	String	是	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。
QueryCondition	String	是	{"Type":"status","Value":"finished"}	指定SQL的查询条件，格式为JSON字符串，包含 `Type` 、 `Value` 、 `Min` 或 `Max` 等字段。其中 `Type` 表示查询维度（当前仅支持3个取值： `maxCost` 、 `status` 和 `cost` ）， `Value` 、 `Min` 或 `Max` 表示该维度下的查询范围。具体支持的取值范围如下： • `{"Type":"maxCost","Value":"100"}` ：表示查看执行耗时最长的前100条SQL详情，当前 `Value` 的取值仅支持设置为100。 • `{"Type":"status","Value":"finished"}` ：表示查看已完成的SQL详情。您也可以将 `Value` 设置为 `running` 或 `failed` 来查询正在执行或执行失败的SQL详情。 • `{"Type":"cost","Min":"10","Max":"200"}` ：表示查看执行耗时为10毫秒~200毫秒的SQL详情，您也可以自定义执行耗时的最大值与最小值，单位：毫秒。
Keyword	String	否	select	查询关键字。
MinPeakMemory	Long	否	0	SQL语句的最小峰值内存，单位：Byte。
MaxPeakMemory	Long	否	89000000	SQL语句的最大峰值内存，单位：Byte。
MinScanSize	Long	否	0	SQL语句的最小扫描量，单位：Byte。
MaxScanSize	Long	否	104428198	SQL语句的最大扫描量，单位：Byte。
ResourceGroup	String	否	user_default	SQL语句所属的资源组。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL语句所属的资源组、数据库名、用户名以及访问源地址信息。
UserName	String	否	test_user	执行SQL语句的用户名。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL语句所属的资源组、数据库名、用户名以及访问源地址信息。
Database	String	否	adb_demo	执行SQL语句的数据库。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL语句所属的资源组、数据库名、用户名以及访问源地址信息。

名称	类型	是否必选	示例值	描述
ClientIp	String	否	59.82.xx.xx	访问源地址。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL语句所属的资源组、数据库名、用户名以及访问源地址信息。

Order	String	否	[{"Field":"StartTime", "Type": "desc" }]	根据指定字段对SQL语句进行排序，格式为JSON，是一个有序JSON数组，按输入数组的顺序进行复合排序，包含 `Field` 和 `Type` 两个字段，例如 `[{"Field":"StartTime", "Type": "desc" }]` 。其中：
				• `Field` 表示需要排序的字段名，支持如下字段：
				  ○ `StartTime` ：执行开始时间。
				  ○ `Status` ：执行状态。
				  ○ `UserName` ：用户名称。
				  ○ `Cost` ：执行耗时。
				  ○ `PeakMemory` ：峰值内存。
				  ○ `ScanSize` ：扫描数据量。
				  ○ `Database` ：数据库名称。
				  ○ `ClientIp` ：访问源地址.
				  ○ `ResourceGroup` ：资源组。
				  ○ `QueueTime` ：排队耗时。
				  ○ `OutputRows` ：输出行数。
				  ○ `OutputDataSize` ：输出数据量。
				  ○ `ResourceCostRank` ：SQL内部算子的耗时排名（仅当 `QueryCondition` 取值为 `{"Type":"status","Value":"running"}` 时，支持使用该字段）。
				• `Type` 表示排序类型，支持如下取值（取值均不区分大小写）：
				  ○ `Desc` ：降序。
				  ○ `Asc` ：升序。

PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。默认值为**1**。
PageSize	Integer	否	30	每页记录数，取值为**30**（默认值）、**50**或**100**。
Lang	String	否	zh	设置下载文件的文件标题以及部分错误信息的语言，支持如下语言： • **zh**：简体中文（默认语言）。 • **en**：英文。 • **ja**：日文。 • **zh-tw**：繁体中文。
PatternId	Long	否	5575924945138******	SQL Pattern的ID。

返回数据

名称	类型	示例值	描述
PageNumber	Integer	1	页数。
PageSize	Integer	30	本页记录数。
TotalCount	Integer	1	总记录数。
Querys	Array of Items		SQL语句详情列表。

名称	类型	示例值	描述
SQL	String	SELECT count(*)\nFROM nation	SQL语句详情。 **说明** 出于性能考虑，当前SQL语句最长支持显示5120个字符，超出限制的SQL语句会被截断。您可以调用DownloadDiagnosisRecords接口下载符合指定条件的SQL语句的摘要信息，包括完整的SQL语句。
SQLTruncatedThreshold	Long	5120	SQL语句的截断阈值，固定为5120个字符。超过该长度的SQL语句会被截断。
Status	String	finished	SQL语句的状态，取值为： • **running**：执行中。 • **finished**：已完成。 • **failed**：执行失败。
OutputDataSize	Long	9	返回数据量，单位：Byte。
Cost	Long	10	查询的总耗时。单位：毫秒。 **说明** 该耗时指标是 `QueuedTime` 、 `TotalPlanningTime` 和 `ExecutionTime` 三个耗时指标的累加值。
OutputRows	Long	1	返回行数。
RcHost	String	10.0.xx.xx:3004	执行SQL语句的AnalyticDB MySQL前端节点IP和端口信息。
ScanSize	Long	9	扫描数据量，单位：Byte。
ProcessId	String	20210930004144010000000235031510******	查询ID。
StartTime	Long	1632933704000	SQL语句的执行开始时间，格式为Unix时间戳，单位：毫秒。
SQLTruncated	Boolean	false	查询结果的长度是否超过阈值（即是否被截断），取值如下： • **true**：查询结果的长度超过阈值。 • **false**：查询结果的长度未超过阈值。
Database	String	adb_demo	执行SQL语句的数据库名称。
ScanRows	Long	1	扫描行数。
ResourceCostRank	Integer	1	SQL语句内部算子的耗时排名。 **说明** 仅状态为执行中，即 `Status` 取值为 `running` 的SQL语句支持返回该字段。
ClientIp	String	59.82.xx.xx	访问源地址。
PeakMemory	Long	16648	峰值内存，单位：Byte。
QueueTime	Long	0	排队耗时，单位：毫秒。
ResourceGroup	String	user_default	SQL语句所属的资源池。
UserName	String	test_user	执行SQL语句的用户名。

名称	类型	示例值	描述
ExecutionTime	Long	6	查询执行的耗时，单位：毫秒。
TotalPlanningTime	Long	4	生成执行计划的耗时，单位：毫秒。
EtlWriteRows	Long	0	ETL任务写表的行数。
TotalStages	Integer	2	查询生成的总Stage数量。
RequestId	String	109462AF-B5FA-3D5A-9377-B27E5B******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDiagnosisRecords
&DBClusterId=am-bp1r053byu48p****
&StartTime=1632931200000
&EndTime=1633017540000
&RegionId=cn-hangzhou
&QueryCondition={"Type":"status","Value":"finished"}
&Keyword=select
&MinPeakMemory=0
&MaxPeakMemory=89000000
&MinScanSize=0
&MaxScanSize=104428198
&ResourceGroup=user_default
&UserName=test_user
&Database=adb_demo
&ClientIp=59.82.xx.xx
&Order=[{"Field":"StartTime", "Type": "desc" }]
&PageNumber=1
&PageSize=30
&Lang=zh
&公共请求参数
```

正常返回示例

`XML` **格式**

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDiagnosisRecordsResponse>
    <PageNumber>1</PageNumber>
    <PageSize>30</PageSize>
    <TotalCount>1</TotalCount>
    <Querys>
        <SQL>SELECT count(*)\nFROM nation</SQL>
        <SQLTruncatedThreshold>5120</SQLTruncatedThreshold>
        <Status>finished</Status>
        <OutputDataSize>9</OutputDataSize>
        <Cost>10</Cost>
        <OutputRows>1</OutputRows>
        <RcHost>10.0.xx.xx:3004</RcHost>
        <ScanSize>9</ScanSize>
        <ProcessId>20210930004144010000000235031 51******</ProcessId>
        <StartTime>1632933704000</StartTime>
        <SQLTruncated>false</SQLTruncated>
        <Database>adb_demo</Database>
        <ScanRows>1</ScanRows>
        <ResourceCostRank>1</ResourceCostRank>
        <ClientIp>59.82.xx.xx</ClientIp>
        <PeakMemory>16648</PeakMemory>
        <QueueTime>0</QueueTime>
        <ResourceGroup>user_default</ResourceGroup>
        <UserName>test_user</UserName>
        <ExecutionTime>6</ExecutionTime>
        <TotalPlanningTime>4</TotalPlanningTime>
        <EtlWriteRows>0</EtlWriteRows>
        <TotalStages>2</TotalStages>
    </Querys>
    <RequestId>109462AF-B5FA-3D5A-9377-B27E5B******</RequestId>
</DescribeDiagnosisRecordsResponse>
```

JSON 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "PageNumber" : 1,
  "PageSize" : 30,
  "TotalCount" : 1,
  "Querys" : {
    "SQL" : "SELECT count(*)\\nFROM nation",
    "SQLTruncatedThreshold" : 5120,
    "Status" : "finished",
    "OutputDataSize" : 9,
    "Cost" : 10,
    "OutputRows" : 1,
    "RcHost" : "10.0.xx.xx:3004",
    "ScanSize" : 9,
    "ProcessId" : "20210930004144010000000023503151******",
    "StartTime" : 1632933704000,
    "SQLTruncated" : false,
    "Database" : "adb_demo",
    "ScanRows" : 1,
    "ResourceCostRank" : 1,
    "ClientIp" : "59.82.xx.xx",
    "PeakMemory" : 16648,
    "QueueTime" : 0,
    "ResourceGroup" : "user_default",
    "UserName" : "test_user",
    "ExecutionTime" : 6,
    "TotalPlanningTime" : 4,
    "EtlWriteRows" : 0,
    "TotalStages" : 2
  },
  "RequestId" : "109462AF-B5FA-3D5A-9377-B27E5B******"
}
```

错误码

访问错误中心查看更多错误码。

9.17.2. DescribeDiagnosisDimensions

调用DescribeDiagnosisDimensions接口查看SQL在资源组、数据库名、用户名以及访问源地址等维度下的去重统计信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDiagnosisDimensions	系统规定参数。取值：**DescribeDiagnosisDimensions**。
DBClusterId	String	是	am-bt6u59zcmd945****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
StartTime	String	是	1625220210000	查询开始时间，格式为Unix时间戳，单位：毫秒。 说明 最多仅支持查看14天前的数据，查询超过14天前的数据返回结果为空。

名称	类型	是否必选	示例值	描述
EndTime	String	是	1625220213000	查询结束时间，格式为Unix时间戳，单位：毫秒。 说明 • 查询结束时间需晚于查询开始时间。 • 开始时间与结束时间的间隔不能超过24小时。
RegionId	String	否	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。
QueryCondition	String	是	{"Type":"maxCost","Value":"100"}	SQL查询的条件，格式为JSON字符串，包含 Type 、Value 、Min 或 Max 等字段。其中 Type 表示查询维度（当前仅支持3个取值：maxCost 、status 和 cost ）；Value 、Min 或 Max 表示该维度下的查询范围。取值说明： • {"Type":"maxCost","Value":"100"} ：表示查看执行耗时最长的前100条SQL详情，当前 Value 的取值仅支持设置为100。 • {"Type":"status","Value":"finished"} ：表示查看已完成的SQL详情。您也可以将 Value 设置为 running 或 failed 来查询正在执行或执行失败的SQL详情。 • {"Type":"cost","Min":"10","Max":"200"} ：表示查看执行耗时为10毫秒~200毫秒的SQL详情，您也可以自定义执行耗时的最大值与最小值，单位：毫秒。
Lang	String	否	zh	设置下载文件的文件标题以及部分错误信息的语言，支持如下语言： • **zh**：简体中文（默认语言）。 • **en**：英文。 • **ja**：日文。 • **zh-tw**：繁体中文。

返回数据

名称	类型	示例值	描述
ClientIps	Array of String	["106.11.xx.xx","106.11.xx.xx"]	访问源地址。
ResourceGroups	Array of String	["user_defaul"]	资源组名称。
UserNames	Array of String	["tset_use"]	用户名。
Databases	Array of String	["tpch_1g"]	数据库名称。
RequestId	String	E0B56BCD-1BED-30EC-8CAF-1D1E5F******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDiagnosisDimensions
&DBClusterId=am-bt6u59zcmd945****
&StartTime=1625220210000
&EndTime=1625220213000
&RegionId=cn-hangzhou
&QueryCondition={"Type":"maxCost","Value":"100"}
&Lang=zh
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDiagnosisDimensionsResponse>
    <ClientIps>106.11.xx.xx, 106.11.xx.xx</ClientIps>
    <ResourceGroups>user_defaul</ResourceGroups>
    <UserNames>tset_use</UserNames>
    <Databases>tpch_1g</Databases>
    <RequestId>E0B56BCD-1BED-30EC-8CAF-1D1E5F******</RequestId>
</DescribeDiagnosisDimensionsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "ClientIps" : [ "106.11.xx.xx,106.11.xx.xx" ],
  "ResourceGroups" : [ "user_defaul" ],
  "UserNames" : [ "tset_use" ],
  "Databases" : [ "tpch_1g" ],
  "RequestId" : "E0B56BCD-1BED-30EC-8CAF-1D1E5F******"
}
```

错误码

访问错误中心查看更多错误码。

9.17.3. DescribeDownloadRecords

调用DescribeDownloadRecords接口查看目标AnalyticDB MySQL版集群中最近5次的SQL查询下载任务列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDownloadRecords	系统规定参数。取值：**DescribeDownloadRecords**。
DBClusterId	String	是	am-********** ********** **********	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
RegionId	String	否	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。
Lang	String	否	zh	设置下载文件的文件标题以及部分错误信息的语言，支持如下语言： • **zh**：简体中文（默认语言）。 • **en**：英文。 • **ja**：日文。 • **zh-tw**：繁体中文。

返回数据

名称	类型	示例值	描述
Records	Array of Items		下载列表详情。
Status	String	finished	任务状态，取值为： • **running**：正在下载。 • **finished**：已完成。 • **failed**：下载失败。
DownloadId	Long	69	下载任务ID。
ExceptionMsg	String	The query result is empty.	下载任务失败时返回的异常信息。
Url	String	https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**********/20210805104301-20210805164302.xlsx?Expires=1943514161&OSSAccessKeyId=*********&Signature=******"	文件下载地址。
FileName	String	20210806094635-20210806095135	下载文件名。
RequestId	String	987F51BE-C4CB-332A-B159-63CE87******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDownloadRecords
&DBClusterId=am-****************
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDownloadRecordsResponse>
    <RequestId>987F51BE-C4CB-332A-B159-63CE87******</RequestId>
    <Records>
        <Status>failed</Status>
        <FileName>20210806094635-20210806095135.xlsx</FileName>
        <ExceptionMsg>The query result is empty.</ExceptionMsg>
        <Url/>
        <DownloadId>73</DownloadId>
    </Records>
    <Records>
        <Status>finished</Status>
        <FileName>20210805104301-20210805164302.xlsx</FileName>
        <ExceptionMsg/>
        <Url>https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210805104301-2021080516430
2.xlsx?Expires=1943514161&amp;OSSAccessKeyId=*********&amp;Signature=******</Url>
        <DownloadId>72</DownloadId>
    </Records>
    <Records>
        <Status>finished</Status>
        <FileName>20210805104301-20210805164301.xlsx</FileName>
        <ExceptionMsg/>
        <Url>https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210805104301-2021080516430
1.xlsx?Expires=1943513847&amp;OSSAccessKeyId=*********&amp;Signature=******</Url>
        <DownloadId>71</DownloadId>
    </Records>
    <Records>
        <Status>finished</Status>
        <FileName>20210804234858-20210805114858.xlsx</FileName>
        <ExceptionMsg/>
        <Url>https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210804234858-2021080511485
8.xlsx?Expires=1943495349&amp;OSSAccessKeyId=*********&amp;Signature=******</Url>
        <DownloadId>70</DownloadId>
    </Records>
    <Records>
        <Status>finished</Status>
        <FileName>20210702180330-20210702180333.xlsx</FileName>
        <ExceptionMsg/>
        <Url>https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210702180330-2021070218033
3.xlsx?Expires=1941607637&amp;OSSAccessKeyId=*********&amp;Signature=******</Url>
        <DownloadId>69</DownloadId>
    </Records>
</DescribeDownloadRecordsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "987F51BE-C4CB-332A-B159-63CE87******",
  "Records" : [ {
    "Status" : "failed",
    "FileName" : "20210806094635-20210806095135.xlsx",
    "ExceptionMsg" : "The query result is empty.",
    "Url" : "",
    "DownloadId" : 73
  }, {
    "Status" : "finished",
    "FileName" : "20210805104301-20210805164302.xlsx",
    "ExceptionMsg" : "",
    "Url" : "https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210805104301-20210805164302.xlsx?Expires=1943514161&OSSAccessKeyId=*********&Signature=******",
    "DownloadId" : 72
  }, {
    "Status" : "finished",
    "FileName" : "20210805104301-20210805164301.xlsx",
    "ExceptionMsg" : "",
    "Url" : "https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210805104301-20210805164301.xlsx?Expires=1943513847&OSSAccessKeyId=*********&Signature=******",
    "DownloadId" : 71
  }, {
    "Status" : "finished",
    "FileName" : "20210804234858-20210805114858.xlsx",
    "ExceptionMsg" : "",
    "Url" : "https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210804234858-20210805114858.xlsx?Expires=1943495349&OSSAccessKeyId=*********&Signature=******",
    "DownloadId" : 70
  }, {
    "Status" : "finished",
    "FileName" : "20210702180330-20210702180333.xlsx",
    "ExceptionMsg" : "",
    "Url" : "https://perth-download-task.oss-cn-beijing.aliyuncs.com/adbmysql/query-sql-logs/am-**************/20210702180330-20210702180333.xlsx?Expires=1941607637&OSSAccessKeyId=*********&Signature=******",
    "DownloadId" : 69
  } ]
}
```

错误码

访问错误中心查看更多错误码。

9.17.4. DownloadDiagnosisRecords

调用DownloadDiagnosisRecords接口下载目标AnalyticDB MySQL版集群中符合指定条件的查询SQL的摘要信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DownloadDiagnosisRecords	系统规定参数。取值：**DownloadDiagnosisRecords**。
DBClusterId	String	是	am-bp1r053byu48p****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
StartTime	String	是	1625220210000	查询开始时间，格式为Unix时间戳，单位：毫秒。 说明 最多仅支持查看14天前的数据。

名称	类型	是否必选	示例值	描述
EndTime	String	是	1625220213000	查询结束时间，格式为Unix时间戳，单位：毫秒。 说明 • 查询结束时间需晚于查询开始时间。 • 开始时间与结束时间的间隔不能超过24小时。
RegionId	String	否	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。
QueryCondition	String	是	{"Type":"status","Value":"finished"}	指定SQL查询的条件，格式为JSON字符串，包含 `Type` 、 `Value` 、 `Min` 或 `Max` 等字段。其中 `Type` 表示查询维度（当前仅支持 `maxCost` 、 `status` 和 `cost` 3个取值）； `Value` 、 `Min` 或 `Max` 表示该维度下的查询范围。具体支持的取值范围如下： • `{"Type":"maxCost","Value":"100"}` ：表示查看执行耗时最长的前100条SQL详情，当前 `Value` 的取值仅支持设置为100。 • `{"Type":"status","Value":"finished"}` ：表示查看已完成的SQL详情。您也可以将 `Value` 设置为 `running` 或 `failed` 来查询正在执行或执行失败的SQL详情。 • `{"Type":"cost","Min":"10","Max":"200"}` ：表示查看执行耗时为10毫秒~200毫秒的SQL详情，您也可以自定义执行耗时的最大值与最小值，单位：毫秒。
Keyword	String	否	select	查询关键字。
MinPeakMemory	Long	否	88000000	目标SQL的最小峰值内存，单位：Byte。
MaxPeakMemory	Long	否	88000000	目标SQL的最大峰值内存，单位：Byte。
MinScanSize	Long	否	100000000	目标SQL的最小扫描量，单位：Byte。
MaxScanSize	Long	否	110000000	目标SQL的最大扫描量，单位：Byte。
ResourceGroup	String	否	user_default	目标SQL所属的资源组。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL所属的资源组、数据库名、用户名以及访问源地址信息。
UserName	String	否	test_user	执行目标SQL的用户名。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL所属的资源组、数据库名、用户名以及访问源地址信息。
Database	String	否	adb_demo	执行目标SQL的数据库。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL所属的资源组、数据库名、用户名以及访问源地址信息。

名称	类型	是否必选	示例值	描述
ClientIp	String	否	106.11.XX.XX	访问源地址。 说明 您可以调用DescribeDiagnosisDimensions接口查看符合指定检索条件的SQL所属的资源组、数据库名、用户名以及访问源地址信息。
Lang	String	否	zh	设置下载文件的文件标题以及部分错误信息的语言，支持如下语言： • zh：简体中文（默认语言）。 • en：英文。 • ja：日文。 • zh-tw：繁体中文。

返回数据

名称	类型	示例值	描述
DownloadId	Integer	68	下载ID。
RequestId	String	D4ACF4E0-2952-3A87-9A2C-474058******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DownloadDiagnosisRecords
&DBClusterId=am-bp1r053byu48p****
&StartTime=1625220210000
&EndTime=1625220213000
&RegionId=cn-hangzhou
&QueryCondition={"Type":"status","Value":"finished"}
&Keyword=select
&MinPeakMemory=88000000
&MaxPeakMemory=88000000
&MinScanSize=100000000
&MaxScanSize=110000000
&ResourceGroup=user_default
&UserName=test_user
&Database=adb_demo
&ClientIp=106.11.XX.XX
&Lang=zh
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DownloadDiagnosisRecordsResponse>
    <RequestId>D4ACF4E0-2952-3A87-9A2C-474058******</RequestId>
    <DownloadId>68</DownloadId>
</DownloadDiagnosisRecordsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "D4ACF4E0-2952-3A87-9A2C-474058******",
  "DownloadId" : 68
}
```

错误码

访问错误中心查看更多错误码。

9.18. 实例运行报告

9.18.1. DescribeTableAccessCount

调用DescribeTableAccessCount接口查看指定日期内AnalyticDB MySQL版集群下目标表或所有表的被访问次数。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeTableAccessCount	系统规定参数。取值：**DescribeTableAccessCount**。
DBClusterId	String	是	am-bp1r053byu48p****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
TableName	String	否	CUSTOMER	目标表名。 说明 若该参数留空，则会返回当前日期内目标集群下所有表的使用频次数据。
StartTime	String	是	2021-08-30	需要查询的日期，格式为*yyyy-MM-dd*（UTC时间）。 说明 仅支持查看30天内的数据。

Order	String	否	[{"Field":"TableSchema","Type":"Asc"}]	将查询结果按指定字段进行排序。格式为JSON字符串，例如 `[{"Field":"TableSchema","Type":"Asc"}]`。其中：
				• `Field` 表示排序字段。支持如下取值：
				  ◦ `TableSchema`：表所属的数据库名。
				  ◦ `TableName`：表名。
				  ◦ `AccessCount`：表被访问的次数。
				• `Type` 表示排序方式。支持如下取值：
				  ◦ `Asc`：升序排序。
				  ◦ `Desc`：降序排序。
				说明 若不设置参数，默认按照目标表所属的数据库升序排列。

PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。默认值为**1**。
PageSize	Integer	否	30	每页记录数，取值为任意正整数。默认为**30**。
RegionId	String	是	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	C242707A-01D1-54DA-A5F6-671557******	请求ID。
PageNumber	Integer	1	页数。
PageSize	Integer	30	本页记录数。

名称	类型	示例值	描述
TotalCount	Integer	1	总记录数。
Items	Array of Items		表使用详情。
ReportDate	String	2021-08-30	表使用日期。
TableSchema	String	tpch	表所属的数据库。
AccessCount	String	6	表被访问的次数。
TableName	String	CUSTOMER	表名。
InstanceName	String	am-bp1r053byu48p****	表所属集群的集群ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeTableAccessCount
&DBClusterId=am-bp1r053byu48p****
&StartTime=2021-08-30
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeTableAccessCountResponse>
    <RequestId>C242707A-01D1-54DA-A5F6-671557******</RequestId>
    <PageNumber>1</PageNumber>
    <PageSize>30</PageSize>
    <TotalCount>1</TotalCount>
    <Items>
        <ReportDate>2021-08-30</ReportDate>
        <TableSchema>tpch</TableSchema>
        <AccessCount>6</AccessCount>
        <TableName>CUSTOMER</TableName>
        <InstanceName>am-bp1r053byu48p****</InstanceName>
    </Items>
</DescribeTableAccessCountResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "C242707A-01D1-54DA-A5F6-671557******",
  "PageNumber" : 1,
  "PageSize" : 30,
  "TotalCount" : 1,
  "Items" : {
    "ReportDate" : "2021-08-30",
    "TableSchema" : "tpch",
    "AccessCount" : 6,
    "TableName" : "CUSTOMER",
    "InstanceName" : "am-bp1r053byu48p****"
  }
}
```

错误码

HttpCode	错误码	错误信息	描述
503	ServiceUnavailable	An error occurred while processing your request.	系统暂时不可用，请稍后重试

访问错误中心查看更多错误码。

9.18.2. DescribeSqlPattern

调用DescribeSqlPattern接口查看指定日期内AnalyticDB MySQL版集群下的SQL Pattern详情。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeSqlPattern	系统规定参数。取值：**DescribeSqlPattern**。
StartTime	String	是	2021-08-30	需要查询的日期，格式为*yyyy-MM-dd*（UTC时间）。 说明 仅支持查看30天内的数据。

Order	String	否	[{"Field":"Pattern","Type":"Asc"}]	将查询结果按指定字段进行排序。格式为JSON字符串，例如 `[{"Field":"Pattern","Type":"Asc"}]`。其中：
				• `Field` 表示排序字段。支持如下取值：
				  ◦ `Pattern`：SQL Pattern。
				  ◦ `AccessIP`：客户端IP。
				  ◦ `User`：用户名。
				  ◦ `QueryCount`：查询总数。
				  ◦ `AvgPeakMemory`：平均峰值内存，单位：KB。
				  ◦ `MaxPeakMemory`：最大峰值内存，单位：KB。
				  ◦ `AvgCpuTime`：平均计算耗时，单位：ms。
				  ◦ `MaxCpuTime`：最大计算耗时，单位：ms。
				  ◦ `AvgStageCount`：平均Stage个数。
				  ◦ `MaxStageCount`：最大Stage个数。
				  ◦ `AvgTaskCount`：平均Task个数。
				  ◦ `MaxTaskCount`：最大Task个数。
				  ◦ `AvgScanSize`：平均扫描量，单位：KB。
				  ◦ `MaxScanSize`：最大扫描量，单位：KB。
				• `Type` 表示排序方式。支持如下取值：
				  ◦ `Asc`：升序排序。
				  ◦ `Desc`：降序排序。
				说明
				• 若不设置参数，默认按照 `Pattern` 字段升序排列。
				• 若需要按照 `AccessIP` 字段排序，则 `Type` 参数必须设置为 `accessip`；若需要按照 `User` 字段排序，则 `Type` 参数必须留空或设置为 `user`。

PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。默认值为**1**。
SqlPattern	String	否	SELECT	需要包含的SQL Pattern的关键字。 说明 若该参数不填，默认返回目标集群在 `StartTime` 参数所设日期内的所有SQL Pattern详情。
Type	String	否	user	按指定维度来聚合SQL Pattern。支持如下取值： • `user`：按用户维度聚合。 • `accessip`：按访问IP来源聚合。 说明 若该参数不填，默认按照 `user` 维度聚合。

名称	类型	是否必选	示例值	描述
DBClusterId	String	是	am-bp1r053byu48p****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
PageSize	Integer	否	30	每页记录数，取值为任意正整数。默认为**30**。
RegionId	String	是	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。

返回数据

名称	类型	示例值	描述
PageSize	Integer	30	本页记录数。
PageNumber	Integer	1	页数。
TotalCount	Integer	1	总记录数。
Items	Array of Items		SQL Pattern详情。
AvgStageCount	String	2	平均Stage个数。
MaxCpuTime	String	17	最大计算耗时，单位：ms。
AccessIP	String	100.104.***.***	客户端IP。 说明 仅当 `Type` 参数设置为 `accessip` 时，支持返回该参数。
AvgScanSize	String	0	平均扫描量，单位：KB。
MaxScanSize	String	0	最大扫描量，单位：KB。
MaxPeakMemory	String	480096	最大峰值内存，单位：KB。
AvgCpuTime	String	1.0625	平均计算耗时，单位：ms。
User	String	test_acc	用户名。 说明 仅当 `Type` 参数留空或设置为 `user` 时，支持返回该参数。
AvgPeakMemory	String	240048	平均峰值内存，单位：KB。
MaxStageCount	String	2	最大Stage个数。
MaxTaskCount	String	2	最大Task个数。
InstanceName	String	am-bp1r053byu48p****	集群ID。
QueryCount	String	16	查询总数。

名称	类型	示例值	描述
ReportDate	String	2021-08-30	查询日期。
Pattern	String	SELECT table_name, table_schema AS schema_name, create_time, create_time AS last_ddl_time, table_comment AS description , ceil((data_length + index_length) / ? / ?) AS store_capacity , data_length AS data_bytes, index_length AS index_bytes, table_collation AS collation, auto_increment, table_rows AS num_rows , engine FROM information_schema.tables WHERE table_type != ? AND table_schema = ? AND table_name IN (?) ORDER BY 1	SQL Pattern。
AvgTaskCount	String	2	平均Task个数。
RequestId	String	B6F2D1B4-2C9F-5622-B424-5E7965******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeSqlPattern
&StartTime=2021-08-30
&DBClusterId=am-bp1r053byu48p****
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeSqlPatternResponse>
    <PageSize>30</PageSize>
    <PageNumber>1</PageNumber>
    <TotalCount>1</TotalCount>
    <Items>
        <AvgStageCount>2</AvgStageCount>
        <MaxCpuTime>17</MaxCpuTime>
        <AccessIP>100.104.***.***</AccessIP>
        <AvgScanSize>0</AvgScanSize>
        <MaxScanSize>0</MaxScanSize>
        <MaxPeakMemory>480096</MaxPeakMemory>
        <AvgCpuTime>1.0625</AvgCpuTime>
        <User>test_acc</User>
        <AvgPeakMemory>240048</AvgPeakMemory>
        <MaxStageCount>2</MaxStageCount>
        <MaxTaskCount>2</MaxTaskCount>
        <InstanceName>am-bp1r053byu48p****</InstanceName>
        <QueryCount>16</QueryCount>
        <ReportDate>2021-08-30</ReportDate>
        <Pattern>SELECT table_name, table_schema AS schema_name, create_time, create_time AS last_ddl_time, table_comment AS description , ceil((data_length + index_length) / ? / ?) AS store_capacity , data_length AS data_bytes, index_length AS index_bytes, table_collation AS collation, auto_increment, table_rows AS num_rows , engine FROM information_schema.tables WHERE table_type != ? AND table_schema = ? AND table_name IN (?) ORDER BY 1</Pattern>
        <AvgTaskCount>2</AvgTaskCount>
    </Items>
    <RequestId>B6F2D1B4-2C9F-5622-B424-5E7965******</RequestId>
</DescribeSqlPatternResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "PageSize" : 30,
  "PageNumber" : 1,
  "TotalCount" : 1,
  "Items" : {
    "AvgStageCount" : 2,
    "MaxCpuTime" : 17,
    "AccessIP" : "100.104.***.***",
    "AvgScanSize" : 0,
    "MaxScanSize" : 0,
    "MaxPeakMemory" : 480096,
    "AvgCpuTime" : 1.0625,
    "User" : "test_acc",
    "AvgPeakMemory" : 240048,
    "MaxStageCount" : 2,
    "MaxTaskCount" : 2,
    "InstanceName" : "am-bp1r053byu48p****",
    "QueryCount" : 16,
    "ReportDate" : "2021-08-30",
    "Pattern" : "SELECT table_name, table_schema AS schema_name, create_time, create_time AS last_ddl_time, table_comment AS description , ceil((data_length + index_length) / ? / ?) AS store_capacity , data_length AS data_bytes, index_length AS index_bytes, table_collation AS collation, auto_increment, table_rows AS num_rows , engine FROM information_schema.tables WHERE table_type != ? AND table_schema = ? AND table_name IN (?) ORDER BY 1",
    "AvgTaskCount" : 2
  },
  "RequestId" : "B6F2D1B4-2C9F-5622-B424-5E7965******"
}
```


错误码

访问错误中心查看更多错误码。

9.18.3. DescribeDBClusterHealthReport

调用DescribeDBClusterHealthReport接口查看指定日期内目标AnalyticDB MySQL版集群的健康报告主要性能指标（如CPU使用率、存储磁盘IOPS、存储磁盘IO吞吐等）的平均值和最大值。

关于健康报告的更多详情，请参见每日健康报告。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDBClusterHealthReport	系统规定参数。取值：**DescribeDBClusterHealthReport**。
StartTime	String	是	2021-08-30	需要查询的日期，格式为*yyyy-MM-dd*（UTC时间）。 说明 仅支持查看30天内的数据。
DBClusterId	String	是	am-****************	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
RegionId	String	是	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。

返回数据

名称	类型	示例值	描述
Items	Array of Items		健康报告主要性能指标详情。

名称	类型	示例值	描述
Key	String	Query	性能指标维度。支持查看如下维度： • **Query**：查询。 • **INSERT**：写入。 • **CPU**：CPU使用率。 • **DiskUsage**：磁盘总体水位。 • **IOPS**：存储磁盘IOPS。 • **IOThroughput**：存储磁盘IO吞吐。 • **IOAwait**：存储磁盘IO等待耗时。
Max	String	0	当前性能指标的最大值。

Name	String	QPS	性能指标名称。支持查看如下性能指标：
			• **QPS**：查询QPS。
			• **RT**：该性能指标在不同维度下意义不同，其中：
			  ◦ `Query` 维度：表示查询响应时间，单位：毫秒。
			  ◦ `INSERT` 维度：表示写入响应时间，单位：毫秒。
			• **TPS**：写入TPS。
			• **DATA**：写入流量，单位：Byte/s。
			• **WORKER**：存储节点的CPU使用率，单位：%。
			• **EXECUTOR**：计算节点的CPU使用率，单位：%。
			• **RC**：前端节点的CPU使用率，单位：%。
			• **DiskUsage**：磁盘总使用率，单位：%。
			• **READ**：该性能指标在不同维度下意义不同，其中：
			  ◦ `IOPS` 维度：表示存储磁盘的每秒读取次数。
			  ◦ `IOThroughput` 维度：表示存储磁盘的每秒读取量，单位：KB。
			• **WRITE**：该性能指标在不同维度下意义不同，其中：
			  ◦ `IOPS` 维度：表示存储磁盘的每秒写入次数。
			  ◦ `IOThroughput` 维度：表示存储磁盘的每秒写入量，单位：KB。
			• **IOAwait**：存储磁盘IO等待耗时，单位：毫秒。

Avg	String	0	当前性能指标的平均值。
RequestId	String	4ED0E20D-A1BD-56DC-8122-B20D44******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDBClusterHealthReport
&StartTime=2021-08-30
&DBClusterId=am-****************
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDBClusterHealthReportResponse>
    <RequestId>4ED0E20D-A1BD-56DC-8122-B20D44******</RequestId>
    <Items>
        <Avg>0</Avg>
        <Max>0</Max>
        <Key>Query</Key>
        <Name>QPS</Name>
    </Items>
    <Items>
        <Avg>4</Avg>
        <Max>426</Max>
        <Key>Query</Key>
        <Name>RT</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
```

```
        <Max>0</Max>
        <Key>INSERT</Key>
        <Name>TPS</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>1</Max>
        <Key>INSERT</Key>
        <Name>RT</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>1</Max>
        <Key>INSERT</Key>
        <Name>DATA</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>4</Max>
        <Key>CPU</Key>
        <Name>WORKER</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>0</Max>
        <Key>CPU</Key>
        <Name>EXECUTOR</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>0</Max>
        <Key>CPU</Key>
        <Name>RC</Name>
    </Items>
    <Items>
        <Avg>12.59</Avg>
        <Max>12.6</Max>
        <Key>DiskUsage</Key>
        <Name>DiskUsage</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>0</Max>
        <Key>IOPS</Key>
        <Name>READ</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>3</Max>
        <Key>IOPS</Key>
        <Name>WRITE</Name>
    </Items>
    <Items>
        <Avg>9</Avg>
        <Max>4095</Max>
        <Key>IOThroughput</Key>
        <Name>READ</Name>
    </Items>
    <Items>
        <Avg>252</Avg>
        <Max>19114</Max>
        <Key>IOThroughput</Key>
        <Name>WRITE</Name>
    </Items>
    <Items>
        <Avg>0</Avg>
        <Max>0</Max>
        <Key>IOAwait</Key>
        <Name>IOAwait</Name>
    </Items>
</DescribeDBClusterHealthReportResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "4ED0E20D-A1BD-56DC-8122-B20D44******",
  "Items" : [ {
    "Avg" : "0",
    "Max" : "0",
    "Key" : "Query",
    "Name" : "QPS"
  }, {
    "Avg" : "4",
    "Max" : "426",
    "Key" : "Query",
    "Name" : "RT"
  }, {
    "Avg" : "0",
    "Max" : "0",
    "Key" : "INSERT",
    "Name" : "TPS"
  }, {
    "Avg" : "0",
    "Max" : "1",
    "Key" : "INSERT",
    "Name" : "RT"
  }, {
    "Avg" : "0",
    "Max" : "1",
    "Key" : "INSERT",
    "Name" : "DATA"
  }, {
    "Avg" : "0",
    "Max" : "4",
    "Key" : "CPU",
    "Name" : "WORKER"
  }, {
    "Avg" : "0",
    "Max" : "0",
    "Key" : "CPU",
    "Name" : "EXECUTOR"
  }, {
    "Avg" : "0",
    "Max" : "0",
    "Key" : "CPU",
    "Name" : "RC"
  }, {
    "Avg" : "12.59",
    "Max" : "12.6",
    "Key" : "DiskUsage",
    "Name" : "DiskUsage"
  }, {
    "Avg" : "0",
    "Max" : "0",
    "Key" : "IOPS",
    "Name" : "READ"
  }, {
    "Avg" : "0",
    "Max" : "3",
    "Key" : "IOPS",
    "Name" : "WRITE"
  }, {
    "Avg" : "9",
    "Max" : "4095",
    "Key" : "IOThroughput",
    "Name" : "READ"
  }, {
    "Avg" : "252",
    "Max" : "19114",
    "Key" : "IOThroughput",
    "Name" : "WRITE"
  }, {
    "Avg" : "0",
    "Max" : "0",
    "Key" : "IOAwait",
    "Name" : "IOAwait"
  } ]
}
```

错误码

访问错误中心查看更多错误码。

9.18.4. DescribeDBClusterForecast

调用DescribeDBClusterForecast接口查看指定日期内目标AnalyticDB MySQL版集群的磁盘水位、QPS和TPS的趋势预测信息。

关于趋势图的更多详情，请参见每日健康报告。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDBClusterForecast	系统规定参数。取值：**DescribeDBClusterForecast**。
StartTime	String	是	2021-08-30	需要查询的日期，格式为*yyyy-MM-dd*（UTC时间）。
DBClusterId	String	是	am-********** 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。	集群ID。
RegionId	String	是	cn-hangzhou 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。	地域ID。
MetricType	String	否	adbmysql_diskStatistics_disk_used	以`StartTime`所设日期为起点，查看目标维度在指定时间内的趋势预测信息。其中维度取值如下： • `adbmysql_diskStatistics_disk_used`：以小时为粒度统计过去2周的磁盘水位信息，并预测未来2周的磁盘水位趋势。 • `adbmysql_qps_qps`：以小时为粒度统计过去7天的QPS信息，并预测未来2天的QPS趋势。 • `adbmysql_tps_tps`：以小时为粒度统计过去7天的TPS信息，并预测未来2天的TPS趋势。 说明 若该参数留空，则返回上述所有维度的趋势预测信息。

返回数据

名称	类型	示例值	描述
Performances	Array of Performances		各指标详情。
Key	String	adbmysql_diskStatistics_disk_used	指标项。
Unit	String	MB 说明 仅当`Key`为`adbmysql_diskStatistics_disk_used`时，会返回该参数。	单位。
Series	Array of Series		各时间点的指标项详情。
Name	String	fact	对应指标项在如下情况中的值： • `fact`：实际观测值。 • `yhat`：趋势预测值。 • `yhatLower`：趋势预测下限值。 • `yhatUpper`：趋势预测上限值。
Values	String	["2021-08-16 23:58:49","12889.5584"]	对应指标项在各时间点的具体取值。

名称	类型	示例值	描述
RequestId	String	B6B60C7B-897D-5F95-A9E9-CCA23C******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeDBClusterForecast
&StartTime=2021-08-30
&DBClusterId=am-*****************
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeDBClusterForecastResponse>
    <RequestId>B6B60C7B-897D-5F95-A9E9-CCA23C******</RequestId>
    <Performances>
        <Series>
            <Values>
                <0>2021-08-16 23:58:49</0>
                <1>12889.5584</1>
            </Values>
            <Values>
                <0>2021-08-17 00:00:19</0>
                <1>12889.5823</1>
            </Values>
            <Name>fact</Name>
        </Series>
        <Series>
            <Values>
                <0>2021-08-16 23:58:49</0>
                <1>12923.9609</1>
            </Values>
            <Values>
                <0>2021-08-17 00:00:19</0>
                <1>12923.8966</1>
            </Values>
            <Name>yhat</Name>
        </Series>
        <Series>
            <Values>
                <0>2021-08-16 23:58:49</0>
                <1>12858.7716</1>
            </Values>
            <Values>
                <0>2021-08-17 00:00:19</0>
                <1>12854.6119</1>
            </Values>
            <Name>yhatLower</Name>
        </Series>
        <Series>
            <Values>
                <0>2021-08-16 23:58:49</0>
                <1>12992.0099</1>
            </Values>
            <Values>
                <0>2021-08-17 00:00:19</0>
                <1>12992.8980</1>
            </Values>
            <Name>yhatUpper</Name>
        </Series>
        <Unit>MB</Unit>
        <Key>adbmysql_diskStatistics_disk_used</Key>
    </Performances>
</DescribeDBClusterForecastResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "B6B60C7B-897D-5F95-A9E9-CCA23C******",
  "Performances" : [ {
    "Series" : [ {
      "Values" : [ [ "2021-08-16 23:58:49", "12889.5584" ], [ "2021-08-17 00:00:19", "12889.5823" ] ],
      "Name" : "fact"
    }, {
      "Values" : [ [ "2021-08-16 23:58:49", "12923.9609" ], [ "2021-08-17 00:00:19", "12923.8966" ] ],
      "Name" : "yhat"
    }, {
      "Values" : [ [ "2021-08-16 23:58:49", "12858.7716" ], [ "2021-08-17 00:00:19", "12854.6119" ] ],
      "Name" : "yhatLower"
    }, {
      "Values" : [ [ "2021-08-16 23:58:49", "12992.0099" ], [ "2021-08-17 00:00:19", "12992.8980" ] ],
      "Name" : "yhatUpper"
    } ],
    "Unit" : "MB",
    "Key" : "adbmysql_diskStatistics_disk_used"
  } ]
}
```

错误码

访问错误中心查看更多错误码。

9.19. SQL Pattern

9.19.1. DescribeSQLPatterns

调用DescribeSQLPatterns接口查看指定日期内AnalyticDB MySQL版集群下的SQL Pattern列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeSQLPatterns	系统规定参数。取值：**DescribeSQLPatterns**。
DBClusterId	String	是	am-bp1r053byu48p****	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
StartTime	String	是	2021-09-30T00:10:00Z	查询开始时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。 说明 • 仅支持查看最近14天的数据。例如：当前时间为2021-11-22T12:00:00Z，最早可以查询到2021-11-09T12:00:00Z的数据。 • 查询开始时间和查询结束时间的间隔不能大于24小时。
EndTime	String	是	2021-09-30T00:15:00Z	查询结束时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。 说明 查询结束时间需晚于查询开始时间。
RegionId	String	是	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。
Keyword	String	否	SELECT	查询关键字。

名称	类型	是否必选	示例值	描述

Order	String	是	[{"Field":"AverageQueryTime","Type":"Asc"}]	将查询结果按指定字段进行排序。格式为JSON字符串，例如 `[{"Field":"AverageQueryTime","Type":"Asc"}]`。其中：
				• `Field` 表示排序字段。取值说明：
				  ○ `PatternCreationTime`：查询时间范围内，Pattern的最早提交时间。
				  ○ `AverageQueryTime`：查询时间范围内，Pattern的平均总耗时。
				  ○ `MaxQueryTime`：查询时间范围内，Pattern的最大总耗时。
				  ○ `AverageExecutionTime`：查询时间范围内，Pattern的平均执行耗时。
				  ○ `MaxExecutionTime`：查询时间范围内，Pattern的最大执行耗时。
				  ○ `AveragePeakMemory`：查询时间范围内，Pattern的平均峰值内存。
				  ○ `MaxPeakMemory`：查询时间范围内，Pattern的最大峰值内存。
				  ○ `AverageScanSize`：查询时间范围内，Pattern的平均数据读取量。
				  ○ `MaxScanSize`：查询时间范围内，Pattern的最大数据读取量。
				  ○ `QueryCount`：查询时间范围内，Pattern的执行次数。
				  ○ `FailedCount`：查询时间范围内，Pattern的失败次数。
				• `Type` 表示排序方式。取值说明（取值不区分大小写）：
				  ○ `Asc`：升序排序。
				  ○ `Desc`：降序排序。

PageNumber	Integer	否	1	页码，取值为大于0且不超过Integer数据类型的最大值。 说明 本参数不填写，默认为1。
PageSize	Integer	否	30	每页记录数，取值说明： • **30** • **50** • **100** 说明 本参数不填写，默认为30。
Lang	String	否	zh	设置下载文件的文件标题以及部分错误信息的语言，取值说明： • **zh**：简体中文（默认语言）。 • **en**：英文。 • **ja**：日文。 • **zh-tw**：繁体中文。

返回数据

名称	类型	示例值	描述
PageNumber	Integer	1	页数。
PageSize	Integer	30	本页记录数。
TotalCount	Integer	1	总记录数。
PatternDetails	Array of patternDetails		SQL Pattern的详细信息。

名称	类型	示例值	描述
SQLPattern	String	SELECT * FROM KEPLER_META_NODE_STATIC_INFO WHERE elastic_node = ? OR (elastic_node = ? AND enable = ?)	SQL Pattern的具体语句。
PatternId	String	5575924945138******	SQL Pattern的ID。
User	String	reporter	提交Pattern相关的SQL数据库用户名。
AccessIp	String	192.168.xx.xx	提交Pattern相关的SQL客户端IP地址。
Tables	String	tpch.orders	SQL Pattern扫描的列表。
PatternCreationTime	String	2021-11-12 03:06:00	查询时间范围内，Pattern的最早提交时间。单位：毫秒。
AverageQueryTime	double	4	查询时间范围内，Pattern的平均总耗时。单位：毫秒。
MaxQueryTime	Long	2341	查询时间范围内，Pattern的最大总耗时。单位：毫秒。
AverageExecutionTime	double	234.78	查询时间范围内，Pattern的平均执行时间。单位：毫秒。
MaxExecutionTime	Long	2142	查询时间范围内，Pattern的最大执行时间。单位：毫秒。
AveragePeakMemory	double	234.22	查询时间范围内，Pattern的平均峰值内存。单位：Byte。
MaxPeakMemory	Long	234149	Pattern相关SQL的最大峰值内存。单位：Byte。
AverageScanSize	double	234149.23	查询时间范围内，Pattern的平均数据读取量。单位：Byte。
MaxScanSize	Long	234149	Pattern相关SQL的最大数据读取量。单位：Byte。
QueryCount	Long	345	查询时间范围内，Pattern的执行次数。
FailedCount	Long	234	查询时间范围内，Pattern的失败次数。
Blockable	Boolean	true	能否拦截当前SQL Pattern的运行。取值说明： • **true**：支持拦截。 • **false**：不支持拦截。 **说明** 目前AnalyticDB MySQL版仅支持Select和Insert相关语句的拦截。
RequestId	String	6BE0EDD1-0DE6-3EB6-81BF-BFE4F2******	请求ID。

示例

请求示例

```
http(s)://adb.aliyuncs.com/?Action=DescribeSQLPatterns
&DBClusterId=am-bp1r053byu48p****
&StartTime=2021-09-30T00:10:00Z
&EndTime=2021-09-30T00:15:00Z
&RegionId=cn-hangzhou
&Keyword=SELECT
&Order=[{"Field":"AverageQueryTime","Type":"Asc"}]
&PageNumber=1
&PageSize=30
&Lang=zh
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribeSQLPatternsResponse>
    <PageNumber>1</PageNumber>
    <PageSize>30</PageSize>
    <TotalCount>1</TotalCount>
    <PatternDetails>
        <SQLPattern>SELECT * FROM KEPLER_META_NODE_STATIC_INFO WHERE elastic_node = ? OR (elastic_node = ? AND enable = ?)</SQLPattern>
        <PatternId>5575924945138******</PatternId>
        <User>reporter</User>
        <AccessIp>192.168.xx.xx</AccessIp>
        <Tables>tpch.orders</Tables>
        <PatternCreationTime>2021-11-12 03:06:00</PatternCreationTime>
        <AverageQueryTime>4</AverageQueryTime>
        <MaxQueryTime>2341</MaxQueryTime>
        <AverageExecutionTime>234.78</AverageExecutionTime>
        <MaxExecutionTime>2142</MaxExecutionTime>
        <AveragePeakMemory>234.22</AveragePeakMemory>
        <MaxPeakMemory>234149</MaxPeakMemory>
        <AverageScanSize>234149.23</AverageScanSize>
        <MaxScanSize>234149</MaxScanSize>
        <QueryCount>345</QueryCount>
        <FailedCount>234</FailedCount>
        <Blockable>true</Blockable>
    </PatternDetails>
    <RequestId>6BE0EDD1-0DE6-3EB6-81BF-BFE4F2******</RequestId>
</DescribeSQLPatternsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "PageNumber" : 1,
  "PageSize" : 30,
  "TotalCount" : 1,
  "PatternDetails" : {
    "SQLPattern" : "SELECT * FROM KEPLER_META_NODE_STATIC_INFO WHERE elastic_node = ? OR (elastic_node = ? AND enable = ?)",
    "PatternId" : "5575924945138******",
    "User" : "reporter",
    "AccessIp" : "192.168.xx.xx",
    "Tables" : "tpch.orders",
    "PatternCreationTime" : "2021-11-12 03:06:00",
    "AverageQueryTime" : 4,
    "MaxQueryTime" : 2341,
    "AverageExecutionTime" : 234.78,
    "MaxExecutionTime" : 2142,
    "AveragePeakMemory" : 234.22,
    "MaxPeakMemory" : 234149,
    "AverageScanSize" : 234149.23,
    "MaxScanSize" : 234149,
    "QueryCount" : 345,
    "FailedCount" : 234,
    "Blockable" : true
  },
  "RequestId" : "6BE0EDD1-0DE6-3EB6-81BF-BFE4F2******"
}
```

错误码

访问错误中心查看更多错误码。

9.19.2. DescribePatternPerformance

调用DescribePatternPerformance接口查看指定时间段内SQL Pattern的各指标（如查询时间、平均内存消耗）详情。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribePatternPerformance	系统规定参数。取值：**DescribePatternPerformance**。

名称	类型	是否必选	示例值	描述
DBClusterId	String	是	am-********** ********** **********	集群ID。 说明 您可以调用DescribeDBClusters接口查看目标地域下所有AnalyticDB MySQL集群的详情，包括集群ID。
StartTime	String	是	2021-11-18T00:00:00Z	查询开始时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。 说明 • 仅支持查看最近14天内的数据。例如：当前日期为北京时间11月22日，最早可以查询到北京时间11月9日（UTC时间：2021-11-08T16:00:00Z)的数据，若查询时间早于11月9日（UTC时间：2021-11-08T16:00:00Z），返回值为空。 • 查询开始时间和查询结束时间的间隔不能大于24小时。
EndTime	String	是	2021-11-18T18:05:00Z	查询结束时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。 说明 查询结束时间需晚于查询开始时间。
RegionId	String	是	cn-hangzhou	地域ID。 说明 您可以调用DescribeRegions接口查看AnalyticDB MySQL版支持的地域和可用区信息，包括地域ID。
PatternId	String	是	3847585356974******	SQL Pattern的ID。 说明 您可以调用DescribeSQLPatterns接口查看指定时间内目标AnalyticDB MySQL集群下所有的SQL Pattern列表信息，包括SQL Pattern的ID。

返回数据

名称	类型	示例值	描述
EndTime	String	2021-11-18T18:05Z	查询结束时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
RequestId	String	210f4701163402661021352 9******	请求ID。
StartTime	String	2021-11-18T00:00Z	查询开始时间。格式：*yyyy-MM-ddTHH:mm:ssZ*（UTC时间）。
Performances	Array of PerformanceItem		性能指标列表。
Key	String	AnalyticDB_PatternQueryCount	查询的性能指标项。取值说明： • **AnalyticDB_PatternQueryCount**：Pattern相关查询的总次数。 • **AnalyticDB_PatternQueryTime**：Pattern相关查询的总耗时。 • **AnalyticDB_PatternExecutionTime**：Pattern相关查询的执行耗时。 • **AnalyticDB_PatternPeakMemory**：Pattern相关查询的峰值内存。 • **AnalyticDB_PatternScanSize**：Pattern相关查询的数据读取量。

名称	类型	示例值	描述
Unit	String	ms	性能指标项对应的单位。取值说明： • 当性能指标项为查询时间相关（即 Key 值为 AnalyticDB_PatternQueryTime 或 AnalyticDB_PatternExecutionTime ）时，该返回值为ms。 • 当性能指标项为峰值内存相关（即 Key 值为 AnalyticDB_PatternPeakMemory ）时，该返回值为MB。 • 当性能指标项为数据读取量（即 Key 值为 AnalyticDB_PatternScanSize ）时，该返回值为MB。 • 当性能指标项为查询次数（即 Key 值为 AnalyticDB_PatternQueryCount ）时，该返回值为空。
Series	Array of SeriesItem		性能指标项下各性能值的详情。
Values	Array of String	["2021-11-18 13:38:00", "224"]	性能值。

Name	String	max_query_time	性能值名称。取值说明：
			• 当 Key 值为 AnalyticDB_PatternQueryCount 时，该参数返回 pattern_query_count ，即当前Pattern相关SQL的执行次数。
			• 当 Key 值为 AnalyticDB_PatternQueryTime 时，该参数返回：
			  ◦ average_query_time ，即当前Pattern相关SQL的平均总耗时。
			  ◦ max_query_time ，即当前Pattern相关SQL的最大总耗时。
			• 当 Key 值为 AnalyticDB_PatternExecutionTime 时，该参数返回：
			  ◦ average_execution_time ，即当前Pattern相关SQL的平均执行耗时。
			  ◦ max_execution_time ，即当前Pattern相关SQL的最大执行耗时。
			• 当 Key 值为 AnalyticDB_PatternPeakMemory 时，该参数返回：
			  ◦ average_peak_memory ，即当前Pattern相关SQL的平均峰值内存。
			  ◦ max_peak_memory ，即当前Pattern相关SQL的最大峰值内存。
			• 当 Key 值为 AnalyticDB_PatternScanSize 时，该参数返回：
			  ◦ average_scan_size ，即当前Pattern相关SQL的平均读取数据量。
			  ◦ max_scan_size ，即当前Pattern相关SQL的最大数据读取量。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribePatternPerformance
&DBClusterId=am-****************
&StartTime=2021-11-18T00:00:00Z
&EndTime=2021-11-18T18:05:00Z
&RegionId=cn-hangzhou
&PatternId=3847585356974******
&公共请求参数
```

正常返回示例

XML 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribePatternPerformanceResponse>
    <EndTime>2021-11-18T18:05Z</EndTime>
    <RequestId>210f470116340266102135 29******</RequestId>
    <StartTime>2021-11-18T00:00Z</StartTime>
    <Performances>
        <Key>AnalyticDB_PatternQueryCount</Key>
        <Unit>ms</Unit>
        <Series>
            <Values>[              "2021-11-18 13:38:00",              "224"            ]</Values>
            <Name>max_query_time</Name>
        </Series>
    </Performances>
</DescribePatternPerformanceResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "EndTime" : "2021-11-18T18:05Z",
  "RequestId" : "210f470116340266102135 29******",
  "StartTime" : "2021-11-18T00:00Z",
  "Performances" : {
    "Key" : "AnalyticDB_PatternQueryCount",
    "Unit" : "ms",
    "Series" : {
      "Values" : "[              \"2021-11-18 13:38:00\",              \"224\"            ]",
      "Name" : "max_query_time"
    }
  }
}
```

错误码

访问错误中心查看更多错误码。

9.20. 附表

9.20.1. 实例状态表

本文主要介绍实例的一些状态说明。

状态	说明
Preparing	准备中
Creating	创建中
Restoring	备份恢复中
Running	运行中
Deleting	删除中
ClassChanging	变配中
NetAddressCreating	创建网络链接中
NetAddressDeleting	删除网络链接中

10.SDK参考

10.1. SDK参考

云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）支持Java、Python、Go等开发。

> **说明** 更多SDK信息，请参见阿里云开放平台。

Alibaba Cloud SDK	ADB	说明文档
Alibaba Cloud SDK for Java	Alibaba Cloud ADB SDK for Java	SDK使用指南（Java）
Alibaba Cloud SDK for Python	Alibaba Cloud ADB SDK for Python	SDK使用指南（Python）
Alibaba Cloud SDK for Go	Alibaba Cloud ADB SDK for Go	SDK使用指南（Go）
Alibaba Cloud SDK for PHP	Alibaba Cloud ADB SDK for PHP	SDK使用指南（PHP）
Alibaba Cloud SDK for .NET	Alibaba Cloud ADB SDK for .NET	SDK使用指南（.NET）