

阿里云

专有云企业版

云原生数据仓库AnalyticDB
MySQL版
用户指南（3.0）

产品版本：V3.12.0

文档版本：20201231

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 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.什么是云原生数据仓库AnalyticDB MySQL版	12
2.使用须知	13
3.快速入门	14
3.1. 登录云原生数据仓库 AnalyticDB MySQL 版控制台	14
3.2. 新建数据库集群	14
3.3. 设置白名单	14
3.4. 创建数据库账号	15
3.5. 创建数据库	15
3.6. 连接数据库集群	16
3.7. 申请外网地址	16
3.8. 同步数据	16
4.连接数据库集群	17
4.1. MySQL命令行连接云原生数据仓库AnalyticDB MySQL版	17
4.2. 业务系统连接分析型数据库MySQL版	17
4.2.1. C#	17
4.2.2. PHP	18
4.2.3. Python	18
4.2.4. Druid连接池配置	19
4.2.5. Java	19
4.3. 不同编程语言中如何开启客户端的PreparedStatement	22
4.4. 客户端连接分析型数据库MySQL版	23
4.4.1. SQL WorkBench/J	23
4.4.2. DBVisualizer	23
4.4.3. DBeaver	25
4.4.4. Navicat	26
5.管理数据库集群	27

5.1. 查看监控信息	27
5.2. 变更配置	27
5.3. 删除集群	27
6. 备份与恢复	29
6.1. 备份数据	29
6.2. 恢复数据	29
7. 诊断与优化	30
7.1. 慢SQL	30
8. 权限模型	31
8.1. 权限模型	32
8.2. 数据库账号和权限	33
9. 数据可视化	34
9.1. Tableau	34
9.2. QlikView	34
9.3. FineReport	34
10. 数据迁移和同步	36
10.1. 使用DataWorks同步RDS for MySQL数据	36
10.1.1. 概述	36
10.1.2. 配置RDS for MySQL数据源	36
10.1.3. 配置AnalyticDB for MySQL数据源	36
10.1.4. 配置同步任务中的数据来源和去向	37
10.2. 使用Kettle将本地数据同步至ADB	38
11. SQL手册	40
11.1. 数据类型	40
11.2. DDL	41
11.2.1. CREATE DATABASE	41
11.2.2. CREATE TABLE	41
11.2.3. ALTER TABLE	43

11.2.4. CREATE VIEW	45
11.2.5. DROP DATABASE	45
11.2.6. DROP TABLE	45
11.2.7. DROP VIEW	45
11.3. DML	46
11.3.1. INSERT INTO	46
11.3.2. REPLACE INTO	47
11.3.3. INSERT SELECT FROM	47
11.3.4. REPLACE SELECT FROM	47
11.3.5. INSERT OVERWRITE INTO SELECT	48
11.3.6. UPDATE	48
11.3.7. DELETE	48
11.3.8. TRUNCATE TABLE	49
11.3.9. KILL PROCESS	49
11.3.10. SHOW PROCESSLIST	49
11.4. SELECT	50
11.4.1. 语法	50
11.4.2. WITH	50
11.4.3. GROUP BY	51
11.4.4. HAVING	52
11.4.5. JOIN	53
11.4.6. LIMIT	53
11.4.7. ORDER BY	54
11.4.8. 子查询	54
11.4.9. UNION、INTERSECT和EXCEPT	54
11.5. CREATE USER	55
11.6. GRANT	55
11.7. REVOKE	56

11.8. 查询用户	56
11.9. RENAME USER	57
11.10. DROP USER	57
11.11. SHOW	57
12.系统函数	59
12.1. 窗口函数	59
12.2. 聚合函数	64
12.3. 日期和时间函数	68
12.3.1. ADDDATE/DATE_ADD	68
12.3.2. ADDTIME	69
12.3.3. CONVERT_TZ	69
12.3.4. CURDATE	69
12.3.5. CURTIME	69
12.3.6. DATE	70
12.3.7. DATE_FORMAT	70
12.3.8. SUBDATE/DATE_SUB	71
12.3.9. DATEDIFF	71
12.3.10. DAY/DAYOFMONTH	72
12.3.11. DAYNAME	72
12.3.12. DAYOFWEEK	72
12.3.13. DAYOFYEAR	73
12.3.14. EXTRACT	73
12.3.15. FROM_DAYS	73
12.3.16. FROM_UNIXTIME	74
12.3.17. HOUR	74
12.3.18. LAST_DAY	74
12.3.19. LOCALTIME/LOCALTIMESTAMP/NOW	75
12.3.20. MAKEDATE	75

12.3.21. MAKETIME	75
12.3.22. MINUTE	75
12.3.23. MONTH	76
12.3.24. MONTHNAME	76
12.3.25. PERIOD_ADD	76
12.3.26. PERIOD_DIFF	77
12.3.27. QUARTER	77
12.3.28. SEC_TO_TIME	77
12.3.29. SECOND	78
12.3.30. STR_TO_DATE	78
12.3.31. SUBTIME	78
12.3.32. SYSDATE	79
12.3.33. TIME	79
12.3.34. TIME_FORMAT	79
12.3.35. TIME_TO_SEC	79
12.3.36. TIMEDIFF	80
12.3.37. TIMESTAMP	80
12.3.38. TIMESTAMPADD	80
12.3.39. TIMESTAMPDIFF	81
12.3.40. TO_DAYS	81
12.3.41. TO_SECONDS	81
12.3.42. UNIX_TIMESTAMP	82
12.3.43. UTC_DATE	82
12.3.44. UTC_TIME	82
12.3.45. UTC_TIMESTAMP	83
12.3.46. WEEK	83
12.3.47. WEEKDAY	83
12.3.48. WEEKOFYEAR	84

12.3.49. YEAR	84
12.3.50. YEARWEEK	84
12.4. 字符串函数	85
12.4.1. ASCII	85
12.4.2. BIN	85
12.4.3. BIT_LENGTH	85
12.4.4. CHAR	85
12.4.5. CHAR_LENGTH/CHARACTER_LENGTH	86
12.4.6. CONCAT	86
12.4.7. CONCAT_WS	86
12.4.8. ELT	86
12.4.9. EXPORT_SET	87
12.4.10. FIELD	87
12.4.11. FIND_IN_SET	87
12.4.12. FORMAT	87
12.4.13. HEX	88
12.4.14. INSTR	88
12.4.15. LEFT	88
12.4.16. LENGTH/OCTET_LENGTH	88
12.4.17. LIKE	88
12.4.18. LOCATE	89
12.4.19. LOWER/LCASE	89
12.4.20. LPAD	89
12.4.21. LTRIM	89
12.4.22. MAKE_SET	90
12.4.23. MID	90
12.4.24. OCT	90
12.4.25. POSITION	90

12.4.26. REPEAT	91
12.4.27. REPLACE	91
12.4.28. REVERSE	91
12.4.29. RIGHT	91
12.4.30. RLIKE/REGEXP	92
12.4.31. RPAD	92
12.4.32. RTRIM	92
12.4.33. SPACE	92
12.4.34. STRCMP	92
12.4.35. SUBSTR/SUBSTRING	93
12.4.36. SUBSTRING_INDEX	93
12.4.37. TRIM	93
12.4.38. UPPER/UCASE	93
12.5. 数值函数	94
12.5.1. ABS	94
12.5.2. ROUND	94
12.5.3. SQRT	94
12.5.4. LN	95
12.5.5. LOG	95
12.5.6. LOG2	95
12.5.7. PI	95
12.5.8. LOG10	95
12.5.9. POWER/POW	96
12.5.10. RADIANS	96
12.5.11. DEGREES	96
12.5.12. SIGN	96
12.5.13. CEILING/CEIL	97
12.5.14. FLOOR	97

12.5.15. EXP	97
12.5.16. COS	97
12.5.17. ACOS	98
12.5.18. TAN	98
12.5.19. ATAN	98
12.5.20. ATAN2	98
12.5.21. COT	98
12.5.22. ASIN	99
12.5.23. SIN	99
12.6. 算术运算符	99
12.7. 位函数和操作符	100
12.8. GEO函数	102
12.9. JSON函数	103
12.10. 条件判断函数	104
12.11. 加密和压缩函数	106

1.什么是云原生数据仓库AnalyticDB MySQL版

云原生数据仓库AnalyticDB MySQL版（简称ADB，原ADB）是阿里巴巴针对海量数据分析自主研发的实时高并发在线分析RT-OLAP（Realtime OLAP）云计算服务，支持对千亿级数据进行即时地（毫秒级）多维分析透视和业务探索。

② 说明 联机分析处理OLAP（Online Analytical Processing）系统是相对联机事务处理OLTP（Online Transaction Processing）系统而言的，它擅长对已有的海量数据进行多维度地、复杂地查询和分析，适用于ADB系统。OLTP系统擅长事务处理，数据操作保持着严格的一致性和原子性，支持频繁的数据插入和修改，通常用于MySQL、Microsoft SQL Server等关系型数据库系统。

ADB是一套RT-OLAP系统，具有以下特点：

- 兼容MySQL、现有的商业智能BI（Business Intelligence）工具和ETL（Extract-Transform-Load）工具，可以经济、高效、轻松地分析与集成您的所有数据。
- 采用关系模型存储，可以使用SQL进行自由灵活地计算分析，无需预先建模。
- 采用分布式计算技术，具有强大的实时计算能力。

在处理百亿条甚至更多量级的数据上，ADB的性能可以达到甚至超越MOLAP（Multidimension OLAP）类系统。ADB在数百毫秒内可以完成百亿级的数据计算，使用者可以根据自己的想法在海量数据中进行自由地探索，而不是根据预先设定好的逻辑查看已有的数据报表。

- 兼具实时和自由的海量数据计算能力，拥有快速处理千亿级别的海量数据的能力。

在ADB系统中，数据分析使用的数据是业务系统中产生的全量数据而不再是抽样的，这使得数据分析结果具有最大的代表性。

- 能够支撑较高并发查询量，同时通过动态的多副本数据存储计算技术也保证了较高的系统可用性，所以ADB能够直接作为面向最终用户（End User）的产品（包括互联网产品和企业内部的分析产品）的后端系统。

当前ADB已普遍应用于拥有数十万至上千万最终用户的互联网业务系统中，例如淘宝数据魔方、淘宝指数、快的打车、阿里妈妈达摩盘（DMP）、淘宝美食频道等。

作为海量数据下的实时计算系统，ADB给使用者带来了极速的、自由的大数据在线分析计算体验。

2.使用须知

在使用云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）之前，您需要先了解以下内容。

对象	命名规则	限制
数据库名	以小写字母开头，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。	数据库名不能是analyticdb，analyticdb是内置数据库。
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。	<ul style="list-style-type: none">表名不包含引号、感叹号（!）和空格。表名不能是SQL保留关键字。
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。	<ul style="list-style-type: none">列名不包含引号、感叹号（!）和空格。列名不能是SQL保留关键字。
账号名	以小写字母开头，小写字母或数字结尾，可包含小写字母、数字以及下划线（_），长度为2到16个字符。	-
密码限制	包含大写字母、小写字母、数字以及特殊字符（!）、（@）、（#）、（\$）、（%）、（^）、（&）、（*）、（()）、（_）、（+）、（-）、（=），每个密码至少包含其中三项（大写字母、小写字母、数字以及特殊字符），长度为8到32个字符。	-
表中COMMENT	-	最大长度为1024个字符。
列中COMMENT	-	最大长度为1024个字符。
索引名长度	-	最大长度为64个字符。
列默认值	-	最大长度为127个字符。

3.快速入门

3.1. 登录云原生数据仓库 AnalyticDB MySQL 版控制台

本文介绍如何登录云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）控制台。

前提条件

- 登录ASCM控制台前，确认您已从部署人员处获取ASCM控制台的服务域名地址。
- 推荐使用Chrome浏览器。

操作步骤

1. 在浏览器地址栏中，输入ASCM控制台的访问地址，按回车键。
2. 输入正确的用户名及密码。

请向运营管理员获取登录控制台的用户名和密码。

② 说明 首次登录ASCM控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码长度必须为 8~20 位，且至少包含以下两种类型：

- 英文大写或小写字母（A~Z、a~z）
- 阿拉伯数字（0~9）
- 特殊符号（感叹号（!）、at（@）、井号（#）、美元符号（\$）、百分号（%）等）

3. 单击登录，进入ASCM控制台页面。
4. 在页面顶部导航栏中，单击产品，选择数据库 > 分析型数据库MySQL版。

3.2. 新建数据库集群

本文介绍如何创建云原生数据仓库 AnalyticDB MySQL 版集群。

操作步骤

1. 登录分析型数据库MySQL版控制台。
2. 单击控制台右侧的新建集群，根据页面提示进行参数配置。

参数	说明
地域	集群所在的地理位置，创建后无法更换地域。一般建议申请离业务最近的地域，从而提升访问速度。
可用区	可用区是地域中的一个独立物理区域，不同可用区之间没有实质性区别。
组织	设置集群所属组织。
资源集	设置资源组。
版本	仅支持3.0版本。
系列	仅支持基础版
网络类型	分析型数据库MySQL版支持两种类型的网络。 <ul style="list-style-type: none">◦ 专有网络：（Virtual Private Cloud，简称VPC）专有网络帮助您在阿里云上构建出一个隔离的网络环境。您可以自定义专有网络里面的路由表、IP地址范围和网关。建议您选择专有网络，更加安全。◦ 经典网络：经典网络中的云服务在网络上不进行隔离，只能依靠云服务自身的安全组或白名单策略来阻挡非法访问。
规格	ECU类型。
节点组数量	节点组的数量，每个节点组默认三副本。
存储空间	单个节点组的存储空间。

3. 完成上述参数配置后，单击提交

3.3. 设置白名单

本文介绍如何通过云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）控制台设置集群白名单。


背景信息

创建ADB集群后，您需要设置ADB集群的白名单，以允许外部设备访问该集群。集群默认在白名单只包含默认IP地址127.0.0.1，表示任何设备均无法访问该集群。白名单可以让ADB集群得到高级别的访问安全保护，建议您定期维护白名单。设置白名单不会影响ADB集群的正常运行。

操作步骤

1. 登录分析型数据库MySQL版控制台。
2. 在页面左上角，选择集群所在地域。
3. 在集群列表中，单击待设置白名单的集群ID。
4. 在左侧导航栏单击数据安全。

5. 在白名单设置页面，单击default白名单分组右侧的修改。

 说明

您也可以单击添加白名单分组新建自定义分组。

6. 在修改白名单分组对话框中，删除默认IP 127.0.0.1，填写需要访问该集群的IP地址或IP段，然后单击确定。
- 0.0.0.0/0表示允许任何设备访问该集群，请谨慎使用。
 - 若填写IP段，如10.10.10.0/24，表示10.10.10.X的IP地址都可以访问该集群。
 - 若您需要添加多个IP地址或IP段，请用英文逗号(,)隔开（逗号前后都不能有空格），例如192.168.0.1,172.16.213.9。
 - 设置白名单后，新的白名单将于1 min后生效。

3.4. 创建数据库账号

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）中的数据库账号类型以及如何创建数据库账号。

数据库账号类型

ADB支持两种类型的数据库账号：高权限账号和普通账号。

数据库账号类型说明表

数据库账号类型	说明
高权限账号	<ul style="list-style-type: none">只能通过控制台创建和管理高权限账号。一个集群中只能创建一个高权限账号，高权限账号可以管理所有普通账号和数据库。可以断开任意普通账号的连接。开放了更多权限，可满足个性化和精细化的权限管理需求，例如可按用户分配不同表的查询权限等。ADB中的高权限账号相当于MySQL中的root账号。
普通账号	<ul style="list-style-type: none">只能通过数据链路的SQL语句创建和管理普通账号。一个集群最多可以创建256个普通账号。需要手动给普通账号授予特定数据库的权限。普通账号不能断开其他账号的连接。

创建高权限账号

- 登录分析型数据库MySQL版控制台
- 页面左上角，选择集群所在地域。
- 在集群列表中，单击目标集群的集群ID。
- 左侧导航栏单击账号管理，然后在账号管理页面单击创建账号。
- 在创建账号页面，按照页面提示进行参数设置。

参数	说明
数据库账号	高权限账号的账号名。 命名规则：以小写字母开头，小写字母或数字结尾，可包含小写字母、数字以及下划线（_），长度为2到16个字符。
账号类型	高权限账号，不可更改。
密码	高权限账号的密码。 密码设置规则：包含大写字母、小写字母、数字以及特殊字符(!)、(@)、(#)、(\$)、(%)、(^)、(&)、(*)、(())、()、(+)、(-)、(=)，每个密码至少包含其中三项（大写字母、小写字母、数字以及特殊字符），长度为8到32个字符。
确认密码	确认高权限账号的密码。
备注说明	备注信息，可选项。

6. 单击确定，创建高权限账号。

创建普通账号

创建及授权普通账号，请参见CREATE USER。

下一步

创建数据库

3.5. 创建数据库

您可以使用MySQL客户端（Navicat for MySQL、DBeaver、DBVisualizer、SQL WorkBench/J）、业务系统中的程序代码或者MySQL命令行工具连接ADB集群，然后通过CREATE DATABASE语句创建数据库。

前提条件

已连接数据库集群。

语法

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

示例

以下示例创建adb_demo数据库。

```
create database adb_demo;
```

3.6. 连接数据库集群

本文介绍如何连接云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）。

前提条件

完成[设置白名单](#)和[创建数据库账号](#)后，您可以使用MySQL客户端（Navicat for MySQL、DBeaver、DBVisualizer、SQL WorkBench/J）或者MySQL命令行工具连接ADB集群。您也可以[在应用程序中通过配置集群连接地址、端口、账号等信息连接ADB集群](#)。

应用开发中通过代码连接ADB

- [C#](#)
- [PHP](#)
- [Python](#)
- [Druid连接池配置](#)
- [Java](#)

通过MySQL命令行工具连接ADB

[MySQL命令行连接分析型数据库MySQL版](#)

通过客户端连接ADB

- [SQL WorkBench/J](#)
- [DBVisualizer](#)
- [DBeaver](#)
- [Navicat](#)

3.7. 申请外网地址

本文介绍如何通过ADB控制台为集群申请外网地址。

背景信息

- 外网地址需要手动申请，不需要时也可以释放。
- 外网地址适用场景：
 - 阿里云以外的设备需要通过外网地址访问ADB集群。
 - 需要访问的ADB集群位于不同的地域或者网络类型不同。

申请外网地址

1. [登录分析型数据库MySQL版控制台](#)。
2. 在页面左上角，选择集群所在地域。
3. 在集群列表中，单击目标集群的**集群ID**。
4. 在**集群信息**页面的**网络信息**区域，单击**申请外网**。
5. 在**申请外网**提示框中单击**确认**，成功生成外网地址。

② 说明 外网地址生成后，如果需要使用外网地址访问ADB集群，您还需要将待访问ADB集群的设备IP[加入集群白名单](#)。

释放外网地址

1. [登录分析型数据库MySQL版控制台](#)。
2. 在页面左上角，选择集群所在地域。
3. 在集群列表中，单击目标集群的**集群ID**。
4. 在**集群信息**页面，单击**释放外网**。
5. 在**释放外网**提示框中单击**确认**，释放外网地址。

3.8. 同步数据

云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）提供多种数据加载方案，可满足不同场景下的数据同步需求。

- 通过[阿里云数据集成服务](#)将MaxCompute、OSS、MySQL、Oracle、SQLServer中的数据同步至AnalyticDB for MySQL。
- 通过Kettle将关系型数据库、Hbase等NoSQL数据，以及Excel、Access中的数据同步至AnalyticDB for MySQL。
- 通过INSERT外表方式将OSS数据导入AnalyticDB for MySQL或者将AnalyticDB for MySQL数据导出到OSS。
- 通过LOAD DATA将本地数据写入AnalyticDB for MySQL。
- 如果数据已经存在于AnalyticDB for MySQL数据库的其他表中，可以通过 `INSERT INTO...SELECT FROM` 同步数据。

4. 连接数据库集群

4.1. MySQL命令行连接云原生数据仓库AnalyticDB MySQL版

本文介绍如何通过MySQL命令行工具连接云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）。

语法

```
mysql -hadb_url -P3306 -uadb_user -padb_password
```

参数

- `adb_url`：ADB的连接地址，通过AnalyticDB 控制台 > 集群信息 > 网络信息 页面中的版块获取连接地址。
- `3306`：端口为 3306。
- `adb_user`：ADB中的高权限账号或者拥有相关权限的普通账号。
- `adb_password`：账号对应的密码。

示例

```
mysql -ham-bp****.ads.aliyuncs.com -P3306 -utest -pTest123
```

4.2. 业务系统连接分析型数据库MySQL版

4.2.1. C#

本文介绍如何通过MySQL的.NET connector连接云原生数据仓库AnalyticDB MySQL版（简称ADB，原分析型数据库MySQL版）。

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
namespace adbdemo
{
    public class Tutorial2
    {
        public static void Main()
        {
            //server是ADB集群的连接地址URL，可以在控制台的集群信息页面获取连接URL。
            //UID是ADB集群中的用户账号：高权限账号或者普通账号。
            //database是ADB集群中的数据库名称。
            //port是ADB集群连接端口号。
            //password是ADB集群中用户账号对应的密码。
            string connStr = "server=...;UID=...;database=...;port=...;password=...;SslMode=none;";
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
                string sql = "select c_custkey, c_name from customer limit 1";
                MySqlCommand cmd = new MySqlCommand(sql, conn);
                MySqlDataReader rdr = cmd.ExecuteReader();
                while (rdr.Read())
                {
                    Console.WriteLine(rdr[0] + " --- " + rdr[1]);
                }
                rdr.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            conn.Close();
            Console.WriteLine("Done.");
        }
    }
}
```

4.2.2. PHP

本文介绍如何在PHP程序中连接ADB。

注意事项

- 操作系统为Linux时，需要安装php-mysql 5.1.x模块。
- 操作系统为Windows时，需要安装php_MySQL.dll。
- 如果需要在PDO中开启PrepareStatement，请参见。

使用mysqli连接ADB

```
//ADB集群的连接地址URL，可以在控制台的集群信息页面获取连接URL。
$sads_server_name="am-bp***.ads.aliyuncs.com";
//ADB集群中的用户账号：高权限账号或者普通账号。
$sads_username="account_name";
//ADB集群中用户账号对应的密码。
$sads_password="account_password";
//ADB集群中的数据库名称。
$sads_database="db_name";
//ADB集群的连接端口号。
$sads_port=3306;
//连接ADB。
$sads_conn=mysqli_connect($sads_server_name,$sads_username,$sads_password,$sads_database,$sads_port);
```

```
$strsql="SELECT user_id FROM my_ads_db.my_first_table limit 20;";
$result=mysqli_query($sads_conn,$strsql);
while($row = mysqli_fetch_array($result)) {
    //user_id为列名
    echo $row["user_id"];
}
```

使用PDO连接ADB

```
//ADB集群的连接地址URL，可以在控制台的集群信息页面获取连接URL。
$sads_server_name = "am-bp***.ads.aliyuncs.com";
//ADB集群中的用户账号：高权限账号或者普通账号。
$sads_username = "account_name";
//ADB集群中用户账号对应的密码。
$sads_password = "account_password";
//ADB集群中的数据库名称。
$sads_database = 'db_name';
//ADB集群的连接端口号。
$sads_port = 3306;
$dns = "mysql:host={$sads_server_name};dbname={$sads_database};port={$sads_port}";
try {
    $dbh = new PDO($dns, $sads_username, $sads_password);
    echo 'PDO Success!';
} catch (PDOException $e) {
    echo 'PDO Connection failed: ' . $e->getCode() . "\n" . $e->getMessage() . "\n" . $e->getTraceAsString();
}
```

4.2.3. Python

本文介绍如何在Python中通过MySQLdb的module连接AnalyticDB for MySQL。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import MySQLdb
# 打开数据库连接
# host是AnalyticDB for MySQL集群的URL或IP。
# port是AnalyticDB for MySQL集群的URL对应的端口。
# user是AnalyticDB for MySQL集群的用户账号：高权限账号或者普通账号。
# passwd是AnalyticDB for MySQL集群的用户账号对应的密码。
# db是AnalyticDB for MySQL集群中的数据库名。
db = MySQLdb.connect(host='am-bp***.ads.aliyuncs.com', port=3306, user='account_name', passwd='account_password', db='db_name')
# 使用cursor()方法获取操作游标。
cursor = db.cursor()
# 使用execute方法执行SQL语句。
cursor.execute("SELECT VERSION()")
# 使用 fetchone() 方法获取一条数据。
data = cursor.fetchone()
print "Database version : %s" % data
# 关闭数据库连接。
db.close()
```

4.2.4. Druid连接池配置

本文介绍如何通过JDBC Druid连接池连接ADB。

注意事项

- 使用Druid连接池连接ADB时，建议配置 `keepAlive=true`，可以复用连接和避免短连接。
- 使用Druid 1.1.12以上的版本。

配置Druid连接池

```
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">
  <!-- jdbc_url是ADB集群的连接地址URL，可以在控制台的集群信息页面获取连接URL。 -->
  <property name="url" value="${jdbc_url}" />
  <!-- jdbc_user是ADB集群中的用户账号：高权限账号或者普通账号。 -->
  <property name="username" value="${jdbc_user}" />
  <!-- jdbc_password是ADB集群中用户账号对应的密码。 -->
  <property name="password" value="${jdbc_password}" />
  <!-- 配置初始化连接池大小、最小连接数、最大连接数。 -->
  <property name="initialSize" value="5" />
  <property name="minIdle" value="10" />
  <property name="maxActive" value="20" />
  <!-- 配置获取连接等待超时的时间。 -->
  <property name="maxWait" value="60000" />
  <!-- 配置间隔多久进行一次检测，检测需要关闭的空闲连接，单位毫秒。 -->
  <property name="timeBetweenEvictionRunsMillis" value="2000" />
  <!-- 配置一个连接在连接池中的最小生存时间，单位毫秒。 -->
  <property name="minEvictableIdleTimeMillis" value="600000" />
  <property name="maxEvictableIdleTimeMillis" value="900000" />
  <property name="validationQuery" value="select 1" />
  <property name="testWhileIdle" value="true" />
  <!-- 配置从连接池获取连接时，是否检查连接有效性，true每次都检查；false不检查。 -->
  <property name="testOnBorrow" value="false" />
  <!-- 配置向连接池归还连接时，是否检查连接有效性，true每次都检查；false不检查。 -->
  <property name="testOnReturn" value="false" />
  <property name="keepAlive" value="true" />
  <property name="phyMaxUseCount" value="100000" />
  <!-- 配置监控统计拦截的filters。 -->
  <property name="filters" value="stat" />
</bean>
```

4.2.5. Java

本文介绍Java中如何通过MySQL JDBC连接ADB。

MySQL JDBC驱动版本

ADB支持以下版本的MySQL JDBC驱动。

- 5.0版本系列：5.0.2, 5.0.3, 5.0.4, 5.0.5, 5.0.7, 5.0.8。
- 5.1版本系列：
5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7, 5.1.8, 5.1.11, 5.1.12, 5.1.13, 5.1.14, 5.1.15, 5.1.16, 5.1.17, 5.1.18, 5.1.19, 5.1.20, 5.1.21, 5.1.22, 5.1.23, 5.1.24, 5.1.25, 5.1.26, 5.1.27, 5.1.28, 5.1.29, 5.1.30, 5.1.31, 5.1.32, 5.1.33, 5.1.34。

注意事项

Java中创建MySQL JDBC连接依赖于MySQL-JDBC驱动包，您需要手动将MySQL-JDBC驱动包（mysql-connector-java-x.x.x.jar）加入到 `CLASSPATH` 中，否则无法创建MySQL JDBC连接。

不带重试的JDBC连接示例

您可以在业务系统的Java代码中添加以下代码，通过MySQL JDBC连接ADB数据库。

```
Connection connection = null;
Statement statement = null;
ResultSet rs = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
    //adb_url是ADB集群的连接地址URL，可以在控制台的集群信息页面获取连接URL，3306是端口号。
    //db_name是ADB集群中的数据库名称。
    String url = "jdbc:mysql://adb_url:3306/db_name?useUnicode=true&characterEncoding=UTF-8";
    Properties connectionProps = new Properties();
    //account_name是ADB集群中的用户账号：高权限账号或者普通账号。
    connectionProps.put("user", "account_name");
    //account_password是ADB集群中用户账号对应的密码。
    connectionProps.put("password", "account_password");
    connection = DriverManager.getConnection(url, connectionProps);
    statement = connection.createStatement();
    String query = "select count(*) from information_schema.tables";
    rs = statement.executeQuery(query);
    while (rs.next()) {
        System.out.println(rs.getObject(1));
    }
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SQLException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (statement != null) {
        try {
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

带重试的JDBC连接示例

在JDBC中通过配置参数可以实现连接重试机制。

```
public static final int MAX_QUERY_RETRY_TIMES = 3;
```



```

public static final int MAX_QUERY_RETRY_TIMES = 5;
public static Connection conn = null;
public static Statement statement = null;
public static ResultSet rs = null;
public static void main(String[] args) throws ClassNotFoundException {
    //ADB集群中的数据库名称。
    String yourDB = "db_name";
    //ADB集群中的用户账号：高权限账号或者普通账号。
    String username = "account_name";
    //ADB集群中用户账号对应的密码。
    String password = "account_password";
    Class.forName("com.mysql.jdbc.Driver");
    //adb_url是ADB集群的连接地址URL，可以在控制台的集群信息页面获取连接URL，3306是端口号。
    String url = "jdbc:mysql://adb_url:3306/" + yourDB + "?useUnicode=true&characterEncoding=UTF-8";
    Properties connectionProps = new Properties();
    connectionProps.put("user", username);
    connectionProps.put("password", password);
    String query = "select id from test4dmp.test limit 10";
    int retryTimes = 0;
    // 通过循环自动重试。
    while (retryTimes < MAX_QUERY_RETRY_TIMES) {
        try {
            getConn(url, connectionProps);
            execQuery(query); // 执行query。
            break; // query执行成功后，结束整个循环。
        } catch (SQLException e) {
            System.out.println("Met SQL exception: " + e.getMessage() + ", then go to retry task ...");
            try {
                if (conn == null || conn.isClosed()) {
                    retryTimes++;
                }
            } catch (SQLException e1) {
                if (conn != null) {
                    try {
                        conn.close();
                    } catch (SQLException e2) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
    // Clear connection resource.
    closeResource();
}
/**
 * Get connection.
 *
 * @param url
 * @param connectionProps
 * @throws SQLException
 */
public static void getConn(String url, Properties connectionProps) throws SQLException {
    conn = DriverManager.getConnection(url, connectionProps);
}
/**
 * Query task execution logic.
 *
 * @param sql
 * @throws SQLException
 */
public static void execQuery(String sql) throws SQLException {
    Statement statement = null;
    ResultSet rs = null;
    statement = conn.createStatement();
    for (int i = 0; i < 10; i++) {

```

```
for (int i = 0; i < 10; i++) {
    long startTs = System.currentTimeMillis();
    rs = statement.executeQuery(sql);
    int cnt = 0;
    while (rs.next()) {
        cnt++;
        System.out.println(rs.getObject(1) + " ");
    }
    long endTs = System.currentTimeMillis();
    System.out.println("Elapse Time: " + (endTs - startTs));
    System.out.println("Row count: " + cnt);
    try {
        Thread.sleep(160000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}
/**
 * Close connection resource.
 */
public static void closeResource() {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (statement != null) {
        try {
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

4.3. 不同编程语言中如何开启客户端的PrepareStatement

背景信息

大多数数据库中，依靠服务器端预处理语句可以提高数据库性能。ADB数据库自身具备强大的查询计算能力和计划缓存功能，无需依靠服务器端预处理语句获得大部分性能优势。

ADB数据库目前不支持服务器端预编译协议，大部分开发语言中支持通过配置开启客户端预编译（PrepareStatement），也称之为客户端准备语句仿真或参数插值。

本文将介绍如何在不同编程语言中开启客户端的PrepareStatement。

MySQL Connector/J (JDBC) driver

在MySQL Connector/J (JDBC) driver中开启PrepareStatement时，需要配置 `useServerPrepStmts=false` 参数，详情请参见[Configuration Properties for Connector/J](#)。

② 说明 无需配置 `useCursorFetch=true` 参数，否则将覆盖 `useServerPrepStmts=false` 参数，无法开启PrepareStatement。

MariaDB Connector/J

在MariaDB Connector/J中开启PrepareStatement时，需要配置 `useServerPrepStmts=false` 参数，详情请参见[About MariaDB Connector/J](#)。

Go MySQL driver

在Go MySQL driver中开启PrepareStatement时，需要配置 `interpolateParams=true` 参数，详情请参见[Go-MySQL-Driver](#)。

PDO

在PDO中使用PrepareStatement时，需要配置 `PDO::ATTR_EMULATE_PREPARES=TRUE` 参数，详情请参见[setAttribute](#)。

4.4. 客户端连接分析型数据库MySQL版

4.4.1. SQL WorkBench/J

SQL Workbench/J是一个独立于DBMS，跨平台的SQL查询分析工具。本文介绍如何通过SQL WorkBench/J连接AnalyticDB for MySQL集群。

准备工作

开始使用SQL WorkBench/J前，您需要完成以下准备工作。

- 安装MySQL JDBC驱动。
- 安装SQL WorkBench/J。
- 将安装了SQL WorkBench/J/软件的设备IP添加到AnalyticDB for MySQL集群白名单中。

注意事项

如果需要在SQL WorkBench/J中开启PrepareStatement，请参见[SQL WorkBench/J中开启PrepareStatement](#)。

操作步骤

1. 打开SQL WorkBench/J，单击**File > Manage Drivers...**。

② 说明 首次使用SQL WorkBench/J时，需要添加JDBC驱动和jar包，后续使用SQL WorkBench/J时可直接从步骤三开始。

2. 在**Manage drivers**页面，选择驱动为**MySQL**，添加驱动jar包，单击**OK**。
3. 单击**File > Connect window**，在**Select Connection Profile**页面进行参数配置。

参数	说明
New profile	为连接设置一个名字，便于后续管理。
Driver	选择MySQL
URL	AnalyticDB for MySQL集群的连接地址，格式为： <code>jdbc:mysql://hostname:port/name_of_database</code> 。 <ul style="list-style-type: none">◦ <code>hostname</code>：集群的外网地址或者VPC地址。◦ <code>port</code>：3306。◦ <code>name_of_database</code>：数据库名字，可选项。
Username	AnalyticDB for MySQL集群中创建的账号： <ul style="list-style-type: none">◦ 高权限账号◦ 拥有连接集群权限的普通账号
Password	账号对应的密码。

4. 完成上述参数配置后，单击**Test**测试连通性，测试通过后单击**OK**，连接至AnalyticDB for MySQL数据库。

4.4.2. DBVisualizer

本文介绍如何通过DBVisualizer连接ADB集群。

准备工作

开始使用DBVisualizer前，您需要完成以下准备工作。

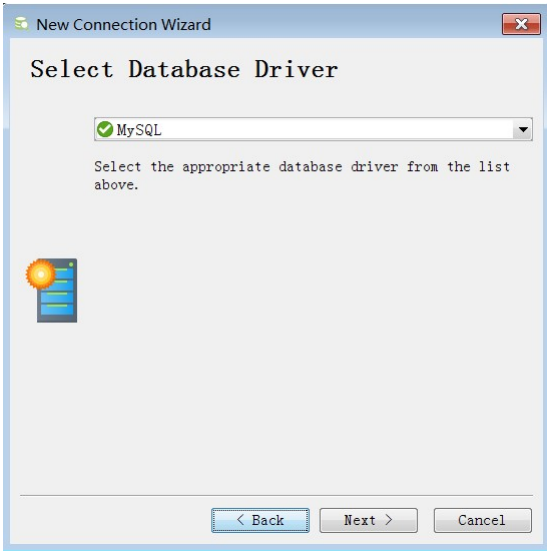
- 安装MySQL JDBC驱动。
- 安装DBVisualizer。
- 将安装了DBVisualizer软件的设备IP添加到ADB集群白名单中。

操作步骤

1. 打开DBVisualizer，在菜单栏单击**Tools > Connection Wizard**，进入**New Connection Wizard**页面，您需要为连接输入一个名字，便于后续管理。

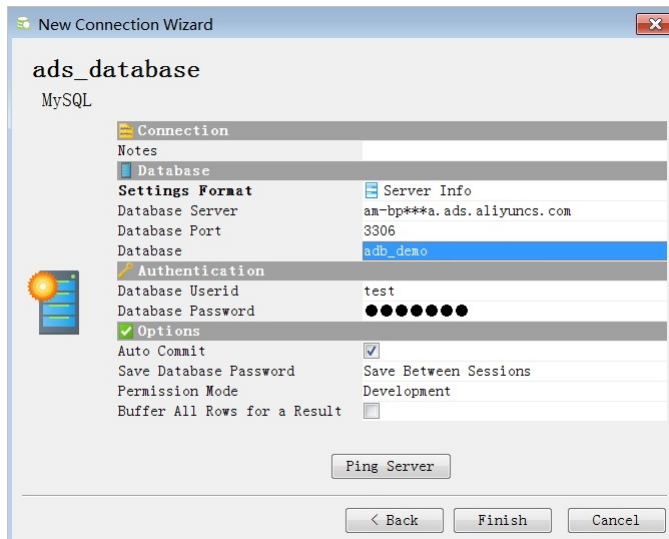


2. 单击Next，选择MySQL作为Database Driver。



3. 单击Next，配置连接参数。

参数	说明
Notes	备注信息
Database Server	ADB集群的连接地址，通过控制台集群信息页面，查看连接信息。
Database Port	3306。
Database	ADB集群中数据库的名字。
Database Userid	ADB集群中创建的账号： <ul style="list-style-type: none">高权限账号拥有连接集群权限的普通账号
Database Password	账号对应的密码。



4. 完成上述参数配置后，单击Ping Server测试连通性，测试通过后，单击Finish。

成功连接ADB后，您可以通过DBVisualizer进行数据管理。

4.4.3. DBeaver

本文介绍如何通过DBeaver连接ADB集群。

背景信息

DBeaver是一款免费、开源（GPL）的专门为开发人员和数据库管理员提供的通用数据库工具。DBeaver支持MySQL、PostgreSQL、Oracle、DB2、MSSQL、Sybase以及其他兼容JDBC的数据库。您可以通过DBeaver的图形界面查看数据库结构、执行SQL查询和脚本、浏览和导出数据、处理BLOB/CLOB数据以及修改数据库结构等。

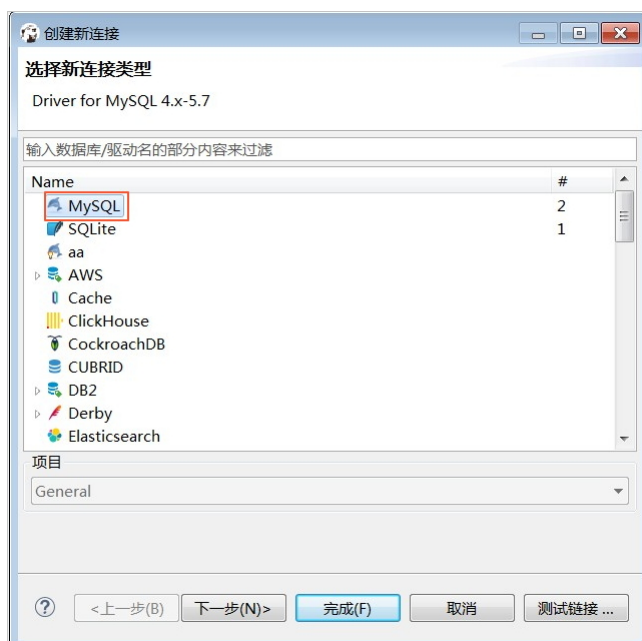
准备工作

开始使用DBeaver前，您需要完成以下准备工作。

- 下载并安装DBeaver软件。
- 安装MySQL JDBC驱动。
- 将安装了DBeaver软件的设备IP添加到ADB集群白名单中。

操作步骤

1. 打开DBeaver，在菜单栏单击数据库 > 新建连接。
2. 在创建新连接页面，连接类型选择MySQL，单击下一步。



3. 在创建新连接页面，进行参数配置。

参数	说明
服务器地址	ADB集群的连接地址，通过控制台集群信息页面，查看连接信息。
端口	3306。
数据库	ADB集群中数据库的名字。
用户名	ADB集群中创建的账号： <ul style="list-style-type: none">高权限账号拥有连接集群权限的普通账号
密码	账号对应的密码。



4. 完成上述参数配置后，单击**测试连接...**测试连通性，测试成功后单击**完成**连接至集群。

4.4.4. Navicat

本文介绍如何通过Navicat连接ADB集群。

背景信息

Navicat是一套快速、可靠且价格相宜的数据库管理工具，专为简化数据库的管理及降低系统管理成本而设。Navicat提供图形化用户界面，您可以简单、方便地创建本机到ADB集群的远程连接，然后使用Navicat进行数据管理。

准备工作

开始使用Navicat for MySQL前，您需要完成以下准备工作。

- **下载**并安装Navicat for MySQL软件。
- 将安装了Navicat for MySQL软件的设备IP添加到ADB集群**白名单**中。

操作步骤

1. 打开Navicat for MySQL，单击**文件 > 新建连接 > MySQL**，在新建连接页面，进行参数配置。

参数	说明
连接名	为连接设置一个名字，便于后续管理。
主机名或IP地址	ADB集群的连接地址，通过控制台集群信息页面，查看连接信息。
端口	3306
用户名	ADB集群中创建的账号： <ul style="list-style-type: none">高权限账号拥有连接集群权限的普通账号
密码	账号对应的密码。

如果您的操作系统是macOS，配置完连接信息后，需要加上数据库名字。

2. 单击**连接测试**测试连通性，测试成功后单击**确定**。
至此已成功建立集群连接，但连接处于关闭状态需要您手动打开连接。
3. 右键单击**连接名 > 打开连接**，然后右键单击**数据库名**，打开某个数据库连接，接下来您就可以利用Navicat for MySQL进行数据管理。

5.管理数据库集群

5.1. 查看监控信息

您可以通过云原生数据仓库 AnalyticDB MySQL 版控制台实时查看集群监控信息。

操作步骤

1. 登录分析型数据库MySQL版控制台。
2. 页面左上角，选择集群所在地域。
3. 在集群列表中，单击目标集群ID。
4. 在左侧导航栏单击监控信息，查看集群监控信息。



监控信息包括CPU使用率、集群连接数、查询QPS、查询响应时间等。

5.2. 变更配置

本文介绍如何变更集群配置。

背景信息

② 说明 暂时仅支持变更节点组数量。

操作步骤

1. 登录分析型数据库MySQL版控制台。
2. 在页面左上角，选择集群所在地域。
3. 单击目标集群右侧的变配。
4. 在分析型数据库MySQL版变配页面，设置节点组数量，然后单击提交。



5.3. 删除集群

据业务需求，您可以手动删除云原生数据仓库 AnalyticDB MySQL 版。

操作步骤

1. [登录分析型数据库MySQL版控制台](#)。
2. 页面左上角，选择集群所在地域。
3. 单击目标集群右侧的**删除**。
4. 在弹出的**删除集群**窗口，单击**确认删除集群**。

6. 备份与恢复

6.1. 备份数据

ADB采用物理备份（快照备份），每周二、周五的02:00-03:00自动备份一次，备份文件保留的时间为7天。

操作步骤

1. [登录分析型数据库MySQL版控制台](#)。
2. 在页面左上角，选择集群所在地域。
3. 在集群列表中，单击目标集群的**集群ID**。
4. 在左侧导航栏单击**备份恢复**。
5. 单击**备份设置**页签，查看数据自动设置。

备份恢复

数据备份

备份设置

数据备份保留天数	7
备份周期	周二、周五
备份时间	02:00-03:00
日志备份保留天数	7
日志备份	关闭

6.2. 恢复数据

在ADB中，集群管理员可以通过备份集恢复数据。

背景信息

如何恢复数据请参见 [云原生数据仓库 AnalyticDB MySQL 版运维指南](#) 中的单实例管理章节。

7. 诊断与优化

7.1. 慢SQL

ADB提供慢SQL分析功能，能够查看慢日志趋势和统计信息，并且提供SQL建议和诊断分析。

操作步骤

1. [登录分析型数据库MySQL版控制台](#)。
2. 在页面左上角，选择集群所在地域。
3. 在集群列表中，单击目标集群的**集群ID**。
4. 单击左侧导航栏中的**慢SQL**。

您可以通过两种方式查看慢日志数据：

- 慢日志趋势：查看慢日志趋势折线图。
- 慢日志明细：查看慢日志详细数据。

8. 权限模型

权限粒度

AnalyticDB for MySQL集群支持以下四个粒度的权限控制。

- GLOBAL：集群级别
- DB：数据库级别
- TABLE：表级别
- COLUMN：列（字段）级别

如果您希望某个用户只查询一张表的某一列数据，可以将具体列的 `SELECT` 权限授予该用户。例如 `GRANT select (customer_id) ON customer TO 'test321'`。

操作和权限关系

操作	需要的权限	权限支持的粒度
SELECT	SELECT	<ul style="list-style-type: none">• DB• TABLE• COLUMN
INSERT	INSERT	<ul style="list-style-type: none">• DB• TABLE• COLUMN
INSERT...SELECT...FROM...	<ul style="list-style-type: none">• INSERT• SELECT	<ul style="list-style-type: none">• DB• TABLE• COLUMN
UPDATE	UPDATE	<ul style="list-style-type: none">• DB• TABLE• COLUMN
DELETE	DELETE	<ul style="list-style-type: none">• DB• TABLE
TRUNCATE TABLE	DROP	<ul style="list-style-type: none">• DB• TABLE
ALTER TABLE	<ul style="list-style-type: none">• ALTER• INSERT• CREATE	<ul style="list-style-type: none">• DB• TABLE
CREATE DATABASE	CREATE	-
CREATE TABLE	CREATE	<ul style="list-style-type: none">• DB• TABLE
SHOW CREATE TABLE	SELECT	<ul style="list-style-type: none">• DB• TABLE
DROP DATABASE	DROP	DB
DROP TABLE	DROP	<ul style="list-style-type: none">• DB• TABLE
CREATE VIEW	<ul style="list-style-type: none">• CREATE_VIEW• SELECT	<ul style="list-style-type: none">• DB• TABLE <div>执行</div> <div>CREATE VIEW REPLACE</div> <div>命令时，除了上述权限，还需要DROP权限。</div>
DROP VIEW	DROP	<ul style="list-style-type: none">• DB• TABLE
SHOW CREATE VIEW	<ul style="list-style-type: none">• SHOW_VIEW• SELECT	<ul style="list-style-type: none">• DB• TABLE
CREATE PROCEDURE	CREATE_ROUTINE	-

操作	需要的权限	权限支持的粒度
DROP_PROCEDURE	ALTER_ROUTINE	-
CREATE_EVENT	EVENT	-
DROP_EVENT	EVENT	-
CREATE USER/DROP USER/RENAME USER	CREATE_USER	-
SET PASSWORD	SUPER	-
GRANT /REVOKE	GRANT	-

8.1. 权限模型

权限粒度

AnalyticDB for MySQL集群支持以下四个粒度的权限控制。

- GLOBAL：集群级别
- DB：数据库级别
- TABLE：表级别
- COLUMN：列（字段）级别

如果您希望某个用户只查询一张表的某一列数据，可以将具体列的 SELECT 权限授予该用户。例如 GRANT select (customer_id) ON customer TO 'test321'。

操作和权限关系

操作	需要的权限	权限支持的粒度
SELECT	SELECT	<ul style="list-style-type: none">DBTABLECOLUMN
INSERT	INSERT	<ul style="list-style-type: none">DBTABLECOLUMN
INSERT...SELECT...FROM...	<ul style="list-style-type: none">INSERTSELECT	<ul style="list-style-type: none">DBTABLECOLUMN
UPDATE	UPDATE	<ul style="list-style-type: none">DBTABLECOLUMN
DELETE	DELETE	<ul style="list-style-type: none">DBTABLE
TRUNCATE TABLE	DROP	<ul style="list-style-type: none">DBTABLE
ALTER TABLE	<ul style="list-style-type: none">ALTERINSERTCREATE	<ul style="list-style-type: none">DBTABLE
CREATE DATABASE	CREATE	-
CREATE TABLE	CREATE	<ul style="list-style-type: none">DBTABLE
SHOW CREATE TABLE	SELECT	<ul style="list-style-type: none">DBTABLE
DROP DATABASE	DROP	DB
DROP TABLE	DROP	<ul style="list-style-type: none">DBTABLE

操作	需要的权限	权限支持的粒度
CREATE VIEW	<ul style="list-style-type: none"> CREATE_VIEW SELECT 	<ul style="list-style-type: none"> DB TABLE 执行 CREATE VIEW REPLACE 命令时，除了上述权限，还需要DROP权限。
DROP VIEW	DROP	<ul style="list-style-type: none"> DB TABLE
SHOW CREATE VIEW	<ul style="list-style-type: none"> SHOW_VIEW SELECT 	<ul style="list-style-type: none"> DB TABLE
CREATE PROCEDURE	CREATE_ROUTINE	-
DROP PROCEDURE	ALTER_ROUTINE	-
CREATE EVENT	EVENT	-
DROP EVENT	EVENT	-
CREATE USER/DROP USER/RENAME USER	CREATE_USER	-
SET PASSWORD	SUPER	-
GRANT /REVOKE	GRANT	-

8.2. 数据库账号和权限

- 创建子账号，请参见[CREATE USER](#)。
- 授权子账号，请参见[GRANT](#)。
- 撤销子账号权限，请参见[REVOKE](#)。
- 更改账号名，请参见[RENAME USER](#)。
- 删除用户，请参见[DROP USER](#)。

9.数据可视化

9.1. Tableau

本文介绍如何通过Tableau连接ADB集群并对数据进行可视化分析。

准备工作

开始使用Tableau前，您需要完成以下准备工作。

- 安装ODBC MySQL Driver，推荐您使用MySQL Connector/ODBC 3.5.1或5.3版本。
- 安装Tableau 9.0及以上版本。

操作步骤

1. 打开Tableau，在左侧菜单栏单击**MySQL**创建MySQL连接，根据页面提示进行参数配置。

参数	说明
服务器名	ADB集群的连接地址。
端口	连接地址对应的端口号。
用户名	ADB集群中创建的账号： <ul style="list-style-type: none">◦ 高权限账号。◦ 拥有连接集群权限的普通账号。
密码	账号对应的密码。

2. 完成上述参数配置后，单击**登录**连接至分析型数据库MySQL版。

使用Tableau

在Tableau上，您可以查看拥有权限的数据库。选择某个数据库后，可以读取数据表以及预览数据进行可视化报表制作。如何使用Tableau，请参见[Tableau](#)

9.2. QlikView

本文介绍如何通过QlikView连接AnalyticDB for MySQL集群并通过QlikView构建BI系统。

准备工作

开始使用QlikView之前，您需要完成以下准备工作。

- 安装ODBC MySQL Driver，建议您使用MySQL Connector/ODBC 3.5.1或5.3版本。
- 安装QlikView 11.20.x版本。

操作步骤

1. 在安装QlikView的主机上，单击**控制面板 > 系统和安全 > 管理工具 > 数据源（ODBC）**（操作系统不同，此步骤可能不同），新建一个DSN，数据源选择**MySQL ODBC 5.xx Driver**。

参数	说明
Data Source Name	AnalyticDB for MySQL集群中数据库的名字。
TCP/IP Server	AnalyticDB for MySQL集群的连接地址。
Port	连接地址对应的端口号。
User	AnalyticDB for MySQL集群中创建的账号： <ul style="list-style-type: none">◦ 高权限账号。◦ 拥有连接集群权限的普通账号。
Password	账号对应的密码。

2. 完成上述参数配置后，单击**Test**测试连通性，测试通过后，单击**OK**添加ODBC数据源。
3. 打开QlikView，单击**文件 > 编辑脚本**，选择步骤1中设置的数据库名称，进行连通性测试。
4. 连通性测试成功后，您可以通过SELECT语句获取AnalyticDB for MySQL集群数据库中的数据。

```
SELECT * FROM DATABASE_NAME.TABLE_NAME;
```

例如，获取user_info表中的数据。

```
SELECT * FROM adb_database.user_info;
```

使用QlikView

在QlikView中获取数据库数据后，您可以利用QlikView进行更多数据操作。如何使用QlikView，请参见[QlikView](#)

9.3. FineReport

本文介绍如何通过FineReport连接ADB集群并进行报表管理。

准备工作

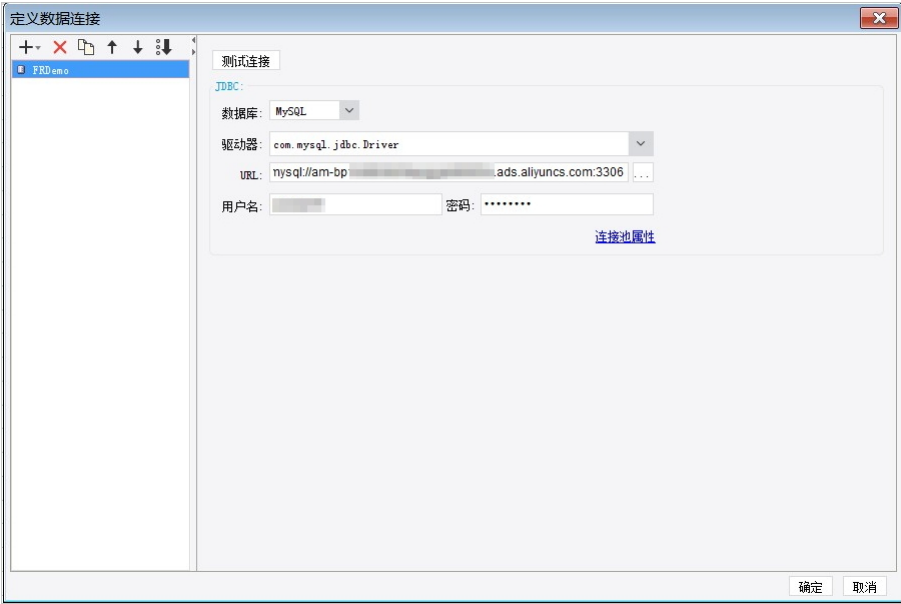
开始使用FineReport前，您需要完成以下准备工作。

- 安装MySQL JDBC驱动。
- 安装FineReport。

操作步骤

1. 打开FineReport，在菜单栏中单击服务器 > 定义数据连接。
2. 在定义数据连接页面进行参数配置。

参数	说明
数据库	选择MySQL。
驱动器	选择MySQL jdbc驱动。
URL	ADB集群的连接地址，格式为：jdbc:mysql://hostname:port 。 <ul style="list-style-type: none">◦ hostname：集群的外网地址或者VPC地址。◦ port：3306。
用户名	ADB集群中创建的账号： <ul style="list-style-type: none">◦ 高权限账号。◦ 拥有连接集群权限的普通账号。
密码	账号对应的密码。



3. 完成上述参数配置后，单击测试连接测试连通性，测试通过后单击确定，连接至ADB集群。

使用FineReport

FineReport连接ADB集群后，您就可以获取ADB中的数据，然后通过FineReport制作各类报表。如何使用FineReport，请参见[FineReport](#)

10.数据迁移和同步

10.1. 使用DataWorks同步RDS for MySQL数据

10.1.1. 概述

数据集成（DataWorks）是阿里云重要的PaaS平台产品，对外提供稳定高效、弹性伸缩的数据同步平台，致力于在复杂网络环境下、丰富的异构数据源之间高速稳定地同步数据。

DataWorks支持的数据源类型

DataWorks数据同步支持丰富的数据源：

- 数据库，例如RDS、DRDS、MySQL、PostgreSQL等。
- NoSQL数据库，例如Memcache、Redis、MongoDB、HBase等。
- 大数据，例如MaxCompute、分析型数据库MySQL版、HDFS等。
- MPP数据库，例如HybridDB for MySQL等。

前提条件

- 根据RDS for MySQL快速入门，准备好测试数据。
- 根据AnalyticDB for MySQL快速入门，完成创建数据库集群、设置白名单等准备工作。
- 在AnalyticDB for MySQL数据库中创建实时表（普通表），用于存储同步过来的RDS for MySQL数据。

② 说明 通过DataWorks同步数据到AnalyticDB for MySQL中时，推荐您使用实时导入的方式，导入效率高而且流程简单，但需要您在AnalyticDB for MySQL数据库中提前创建实时表（普通表）。

- 在DataWorks中创建一个项目。

实施步骤

1. 配置RDS for MySQL数据源
2. 配置AnalyticDB for MySQL数据源
3. 配置同步任务中的数据来源和去向

10.1.2. 配置RDS for MySQL数据源

本文介绍如何在DataWorks中添加RDS for MySQL数据源。

操作步骤

1. 登录DataWorks控制台，在目标项目栏单击**进入数据集成**。
2. 在**数据集成**页面，单击左侧导航栏的**数据源**，然后单击**新增数据源**。
3. 在新增数据源页面，选择**MySQL**。
4. 在新增**MySQL数据源**页面，进行参数配置。

参数	说明
数据源类型	选择 阿里云数据库（RDS） 。
数据源名称	为数据源设置一个名字，便于后续管理。
数据源描述	添加数据源描述，该项为可选项。
地区	RDS for MySQL实例所属地域。
RDS实例ID	RDS for MySQL实例的ID。通过RDS for MySQL实例 基本信息 页面查看实例ID。
RDS实例主帐号ID	阿里云主账号的ID。实例购买者登录控制台，进入 安全设置 中查看实例所属账号ID。
数据库名	RDS for MySQL实例中数据库的名字。
用户名	RDS for MySQL实例中登录数据库使用的用户名。
密码	RDS for MySQL实例中登录数据库使用的用户名对应的密码。

② 说明 如果测试连通性时提示白名单错误，请参见添加RDS文档，在RDS for MySQL实例中添加白名单。

5. 完成上述参数配置后，单击**测试连通性**进行连通性测试，测试通过后单击**完成**添加RDS for MySQL数据源。

10.1.3. 配置AnalyticDB for MySQL数据源

本文介绍如何在DataWorks中添加AnalyticDB for MySQL数据源。

操作步骤

1. 登录DataWorks控制台，单击目标项目栏中的**进入数据集成**。
2. 在**数据集成**页面，单击左侧导航栏的**数据源**，然后单击**新增数据源**。
3. 在新增数据源页面，选择**AnalyticDB for MySQL 3.0**。
4. 在新增**AnalyticDB for MySQL 3.0数据源**页面，进行参数配置。

参数	说明
数据源类型	选择阿里云数据库（AnalyticDB for MySQL）。
数据源名称	为数据源设置一个名字，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。
地区	选择目标AnalyticDB for MySQL实例所属Region。
ADB实例ID	目标AnalyticDB for MySQL的实例ID，通过集群信息页面查看实例ID。
数据库名	AnalyticDB for MySQL中的数据库名。从RDS for MySQL中同步过来的数据将存储在该数据库中。
用户名	AnalyticDB for MySQL中登录数据库使用的用户名。
密码	用户名对应的密码。

5. 完成上述参数配置后，单击**测试连通性**进行连通性测试，测试通过后单击**完成**添加AnalyticDB for MySQL数据源。

10.1.4. 配置同步任务中的数据来源和去向

本文介绍如何在DataWorks中配置数据来源和去向，完成数据同步。

操作步骤

1. 登录DataWorks控制台，单击对应项目操作栏中的**进入数据开发**。
2. 在**数据开发**下右键单击**业务流程**新建一个流程。
3. 单击步骤2新建的业务流程下的**数据集成 > 数据同步**，新建数据同步节点。
4. 双击步骤3中创建的节点，配置数据同步任务的**数据来源（Reader）**、**数据去向（Writer）**、**字段映射**、**通道控制**信息，然后单击**保存**保存任务配置，单击**提交**提交任务配置。



类别	参数	说明
数据来源	数据源	选择MySQL，系统将自动关联配置 配置RDS for MySQL数据源 时设置的数据源名称。
	表	选择RDS for MySQL中的一张表进行数据同步。
	数据过滤	同步数据的筛选条件，暂时不支持limit关键字过滤。SQL语法随着所选择的数据源不同而不同。
	切分键	RDS for MySQL数据表中的主键为切分键。
数据去向	数据源	选择AnalyticDB for MySQL，系统将自动关联 配置AnalyticDB for MySQL数据源 时设置的数据源名称。
	表	选择AnalyticDB for MySQL中的一张表，将RDS for MySQL中的表数据同步至该表中。
	导入前准备语句	输入导入数据前执行的SQL脚本。
	导入后完成语句	输入导入数据后执行的SQL脚本。
	主键冲突	设置主键冲突时的数据处理方式。

② 说明 列与列之间进行字段映射时，字段类型需要与数据兼容。



参数	说明
同行映射	自动将同一行的数据设置映射关系。
自动排版	设置映射关系后，字段排序展示。



参数	说明
任务期望最大并发数	无
同步速率	设置同步速率可以保护读取端数据库，避免抽取速度过大，给读取端造成太大的压力。同步速率建议限流，请结合源库的配置，合理配置抽取速率。
错误记录数超过	当错误记录数超过设置条数后，同步任务自动结束。
任务资源组	无

5. 对同步任务进行调度配置后，单击保存保存任务配置，单击提交提交任务配置。
6. 单击运行开始同步数据。
7. 数据同步成功后，连接分析型数据库MySQL版，通过SELECT查看表数据。
- 成功将数据导入AnalyticDB for MySQL后，您可以使用AnalyticDB for MySQL进行数据分析。

10.2. 使用Kettle将本地数据同步至ADB

本文以Excel为例，介绍如何通过Kettle将本地Excel数据同步至ADB。

背景信息

Kettle是一款非常受欢迎的开源ETL工具软件，主要用于数据整合、转换和迁移。Kettle不仅支持各类关系型数据库和NoSQL数据源（HBase、MongoDB），也支持Excel、Access类型的小型数据源。通过扩展插件，Kettle可以支持更多数据源。

更多Kettle信息，请参见[Kettle](#)

准备工作

开始使用Kettle前，您需要完成以下准备工作。


- 下载并安装Kettle。
- 在ADB中创建目标数据库和表。
如何创建数据库和表，请参见[创建数据库和CREATE TABLE](#)。
- 将安装了Kettle软件的设备IP添加到ADB集群白名单中。

操作步骤

1. 打开Kettle，在菜单栏单击文件 > 新建 > 转换，新建一个转换。
2. 在菜单栏单击文件 > 新建 > 数据库连接在转换中新建一个数据库连接。

参数	说明
连接名称	为连接设置一个名字，便于后续管理。

参数	说明
连接类型	选择MySQL。
连接方式	选择Native（JDBC）。
主机名称	ADB集群的连接地址，通过控制台集群信息页面，查看连接信息。
数据库名称	ADB中的数据库名称。
端口号	3306。
用户名	ADB集群中创建的账号： <ul style="list-style-type: none">高权限账号拥有连接集群权限的普通账号
密码	账号对应的密码。

 说明 配置参数时，不要选中Use Result Streaming Cursor。

- 完成上述参数配置后，单击**测试**弹出**数据库连接测试**提示框，根据提示判断是否连接至ADB数据库，测试通过后单击**确认**。
- 在Kettle左侧**核心对象**的**输入**中，找到**Excel输入**，将其拖动入到工作区。
- 双击工作区的**Excel输入**，在**Excel输入**对话框中，单击**浏览**上传需要导入的Excel表格，然后单击**增加**将其添加到**选中的文件**中。
根据实际业务需要设置**工作表**、**内容**、**字段**等选项卡，单击**预览记录**查看输入的数据是否符合要求。
- 在Kettle左侧**核心对象**的**输出**中，找到**表输出**，将其拖动入到工作区。
- 新建一条**Excel输入**到**表输出**的连接线。
- 双击**表输出**，在**表输出**对话框中进行参数配置。
 - 目标模式**：您需要手动输入ADB数据库的名字。
 - 目标表**：您需要手动输入ADB数据库中的表名。
 - 选中**指定数据库字段**。
 - 选中**使用批量插入**。在**表输出**的数据库字段选项卡中，单击**获取字段和输入字段映射**，可以设置Excel中的列与ADB表中列名间的映射关系。
- 单击白色三角箭头运行转换，观察运行日志和运行状态。
本地Excel数据成功同步至ADB后，您就可以使用ADB对行Excel数据进行分析。

11.SQL手册

11.1. 数据类型

ADB支持的数据类型

关键字	数据类型	取值/取值范围
<code>boolean</code>	布尔类型	值只能是 0 或 1。 取值 0 的逻辑意义为假，取值 1 的逻辑意义为真，存储字节数1比特位。
<code>tinyint</code>	微整数类型	取值范围 -128~127，存储字节数1字节。
<code>smallint</code>	小整数类型	取值范围 -32768~32767，存储字节数2字节。
<code>int</code> 或 <code>integer</code>	整数类型	取值范围 -2147483648~2147483647，存储字节数4字节。
<code>bigint</code>	大整数类型	取值范围 -9223372036854775808~9223372036854775807，存储字节数8字节。
<code>float</code>	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。
<code>double</code>	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。
<code>decimal(m,d)</code>	decimal类型	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。
<code>varchar</code>	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。
<code>date</code>	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。
<code>time</code>	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。
<code>datetime</code>	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。 <div>② 说明 datetime默认UTC时间，且不支持可更改。</div>
<code>timestamp</code>	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。 <div>② 说明 timestamp默认系统时区，可以在SESSION中设置时区。</div>

与MySQL数据类型对比

ADB	MySQL	差异
<code>boolean</code>	<code>bool</code> 、 <code>boolean</code>	一致。
<code>tinyint</code>	<code>tinyint</code>	一致。
<code>smallint</code>	<code>smallint</code>	一致。
<code>int</code> 、 <code>integer</code>	<code>int</code> 、 <code>integer</code>	一致。
<code>bigint</code>	<code>bigint</code>	一致。
<code>float</code>	<code>float[(m,d)]</code>	一致。

ADB	MySQL	差异
double	double[(m,d)]	一致。
decimal	decimal	ADB支持的最大精度为1000，MySQL支持的最大精度为65。
varchar	varchar	ADB中的varchar类型对应MySQL中的CHAR、VARCHAR、TEXT、MEDIUMTEXT 或者LONGTEXT。
date	date	MySQL支持 0000-00-00 ；在ADB中写入 0000-00-00 时，系统自动将其转化为null。
time	time	ADB精确到毫秒，MySQL支持自定义精度。
datetime	datetime	MySQL支持 0000-00-00 00:00:00 ；在ADB中写入 0000-00-00 00:00:00 时，系统自动将其转化为 null 。ADB精确到毫秒，MySQL支持自定义精度。
timestamp	timestamp	ADB精确到毫秒，MySQL支持自定义精度。

11.2. DDL

11.2.1. CREATE DATABASE

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。以小写字母开头，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        |
```

11.2.2. CREATE TABLE

CREATE TABLE用于在AnalyticDB for MySQL中创建表。

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]
[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]
```

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 for MySQL支持的数据类型请参见 数据类型 。
column_attributes	<ul style="list-style-type: none">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 for MySQL为自增列提供唯一值，但自增列的值不是顺序递增。
column_constraints	<ul style="list-style-type: none">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 [, ..])。
table_constraints	INDEX KEY：倒排索引。 AnalyticDB for MySQL默认为表创建全索引，一般情况下无须手动创建索引。
PRIMARY KEY	主键索引。 <ul style="list-style-type: none">只有定义过主键的表支持DELETE和UPDATE操作。主键中必须包含分区键，建议把分区键放到组合主键之前。
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，每个表仅支持创建一个聚集索引。 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。
DISTRIBUTED BY HASH(column_name,...)	在普通表中定义表的分布键，按照 column_name 的HASH值进行分区。 AnalyticDB for MySQL支持将多个字段作为分区键。

参数	说明
DISTRIBUTED BY BROADCAST	用于定义维度表，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	<p>普通表中定义分区。</p> <p>AanalyticDB for MySQL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。</p> <p>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。</p>

注意事项

- 创建表时，AnalyticDB for MySQL集群默认编码格式为utf-8，相当于MySQL中的utf8mp4编码，暂不支持其他编码格式。
- 目前AnalyticDB for MySQL集群支持创建的最大表数目是 节点组数目*256 。

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。

11.2.3. ALTER TABLE

ALTER TABLE用于修改表。

说明 ADB暂不支持更改列的类型。

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 new_table_name  
| TRUNCATE PARTITION {partition_names | ALL}
```

增加列

语法

```
ALTER TABLE db_name.table_name ADD 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_name data_type;
```

示例

在CUSTOMER表中删除类型为 VARCHAR 的 province 列。

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 修改为顾客所属省份。

```
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 列的值更改为可为空（ 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;
```

新增聚集索引

 说明 一张表仅支持创建一个聚集索引。

语法

```
ALTER TABLE db_name.table_name ADD CLUSTERED KEY index_name(column_name1,column_name2);
```

```
ALTER TABLE db_name.table_name ADD CLUSTERED KEY index_name(column_name1);
```

示例

在CUSTOMER表中，为 customer_id 和 id_number 建立索引。

```
ALTER TABLE adb_demo.customer ADD CLUSTERED KEY c_k(customer_id,id_number);
```

删除索引

语法

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

示例

删除CUSTOMER表中的索引。

```
ALTER TABLE adb_demo.customer DROP CLUSTERED KEY c_k;
```

11.2.4. CREATE VIEW

CREATE VIEW用于创建视图。

语法

```
CREATE VIEW view_name AS select_stmt
```

参数

- `view_name`：视图的名字，视图名前可加上数据库名。
- `select_stmt`：视图中的数据来源。

示例

创建视图v，视图数据来源为CUSTOMER表中的数据。

```
CREATE VIEW adb_demo.v AS SELECT * FROM customer;
```

11.2.5. 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;
```

11.2.6. DROP TABLE

DROP TABLE用于删除表。

语法

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

❗ 说明 执行该命令时会同时删除表数据和表结构。

示例

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

11.2.7. DROP VIEW

DROP VIEW用于删除视图。

语法

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

参数

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

示例

删除视图v。

```
DROP VIEW v;
```

11.3. DML

11.3.1. 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','13678973421','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','13678973421','杭州','0','987300','西湖','转塘云栖小镇','23','2018-03-02 10:00:00'),(002368,'张三','13878971234','杭州','0','987300','西湖','转塘云栖小镇','28','2018-08-01 11:00:00'),(002369,'王五','13968075284','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','13678973421','杭州','0','987300','西湖','转塘云栖小镇','23','2018-03-02 10:00:00'),(002368,'张三','13878971234','杭州','0','987300','西湖','转塘云栖小镇','28','2018-08-01 11:00:00'), (002369,'王五','13968075284','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());
```

11.3.2. REPLACE INTO

REPLACE INTO 用于实时覆盖写入数据。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,...)] 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,'杨过','13678973421','杭州','0','987300','西湖','转塘云栖小镇','23','2018-03-02 10:00:00');
```

- 向CUSTOMER表中插入多条数据时，可以省略列名。

```
REPLACE INTO customer values(002367,'李四','13678973421','杭州','0','987300','西湖','转塘云栖小镇','23','2018-03-02 10:00:00'),(002368,'张三','13878971234','杭州','0','987300','西湖','转塘云栖小镇','28','2018-08-01 11:00:00'), (002369,'王五','13968075284','杭州','1','987300','西湖','转塘云栖小镇','35','2018-09-12 08:11:00');
```

11.3.3. 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';
```

11.3.4. 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，已创建好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';
```

11.3.5. INSERT OVERWRITE INTO SELECT

`INSERT OVERWRITE INTO SELECT` 用于批量向表中插入数据。

语法

```
INSERT OVERWRITE INTO table_name [(column_name,...)]
SELECT select_statement FROM from_statement
```

注意事项

- 执行 `INSERT OVERWRITE INTO SELECT` 命令时，需要提前创建好目标表。
- 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，系统自动一键切换将数据写入目标表中。
- 执行 `INSERT OVERWRITE INTO SELECT` 命令时，遇到主键重复时会自动忽略当前写入数据，不做更新。

11.3.6. 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。

示例

将`CUSTOMER`表中 `customer_id='2369'` 客户的姓名更改为黄蓉。

```
update customer set customer_name='黄蓉' where customer_id ='2369';
```

11.3.7. 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;
```

11.3.8. 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

执行 TRUNCATE TABLE 命令将清空表中的数据，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
```

11.3.9. 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;
```

11.3.10. 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;
```

11.4. SELECT

11.4.1. 语法

SELECT语句用于从一个或多个表中查询数据

```
[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。
- 表名和列名不缺分大小写。
- 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。

11.4.2. 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（,），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;
```

11.4.3. GROUP BY

GROUP BY 子句用于对查询结果进行分组，在 GROUP BY 中使用 GROUPING SETS 、 CUBE 、 ROLLUP 可以以不同的形式展示分组结果。

语法

```
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
47685 | 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
```

11.4.4. 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。

语法

```
[ 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

```

11.4.5. JOIN

语法

```

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;

```

11.4.6. 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。

示例

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;
```

11.4.7. ORDER BY

ORDER BY子句用于对查询结果进行排序，ORDER BY中每个表达式由列名或列序号（从1开始）组成。

语法

```
[ ORDER BY expression
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[ LIMIT { count | ALL } ]
```

11.4.8. 子查询

以下示例查询门票销量排名前10位的卖家，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.city not in(select venuecity from venue)
group by firstname, lastname, city
order by maxsold desc, city 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
```

11.4.9. 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 | MINUS }
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 | MINUS：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并且不在第二个集合中出现的所有行。MINUS和EXCEPT作用相同。

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;
```

11.5. CREATE USER

CREATE USER用于创建账号。

语法

```
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';
```

11.6. GRANT

GRANT用于为用户授权。

语法

```
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。
- **column_list**：可选参数，当**priv_type** **SELECT** 为 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帐号。

```
GRANT insert,select,update,delete on *.* to 'test'@'%' identified by 'Testpassword1';
```

创建数据库级别DML帐号。

```
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';
```

11.7. REVOKE

REVOKE用于撤销用户权限。

语法

```
REVOKE
priv_type [(column_list)]
[, priv_type [(column_list)]] ...
ON [object_type] priv_level
FROM user
```

参数

- `priv_type`：权限类型。
- `column_list`：可选参数，当`priv_type` `SELECT` 为 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';
```

11.8. 查询用户

ADB兼容MySQL数据库，ADB中也有一个名为MySQL的内置数据库，该数据库中存储的是ADB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ADB中的用户信息。

注意事项

- ADB中，只有通过高权限账号查询用户信息。
- ADB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 | % | *0fe79c07e168cab9***** |
```

11.9. 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 | % | *0fe79c07e168cab99b**** |
```

11.10. 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;
```

11.11. SHOW

SHOW DATABASES

查看数据库列表。

语法

```
SHOW DATABASES [EXTRA];
```

指定 `EXTRA` 参数时，将输出关于数据库的更多信息，例如创建者ID、数据库连接信息等。

示例

```
SHOW DATABASES EXTRA;

+-----+
| Database |
+-----+
| adb_demo |
| MYSQL    |
| INFORMATION_SCHEMA |
| adb_demo2 |
```

SHOW TABLES

语法

查看用户当前数据库中的表。

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

示例

```
SHOW TABLES IN adb_demo;
+-----+
| Tables_in_adb_demo |
+-----+
| customer           |
| customer2          |
| new_customer       |
| test_table         |
| v                  |
```

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.test_table;
```

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;
```

SHOW GRANTS

查看当前登录用户的权限。

语法

```
SHOW GRANTS;
```

12. 系统函数

本文介绍分析型数据库MySQL版支持的系统函数。

12.1. 窗口函数

云原生数据仓库AnalyticDB 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 slidewindow;
+-----+-----+-----+
| 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需要您注意。

- 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总额和。

```
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 112234。
- 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。
- 返回值类型：与输入参数类型相同。
- 示例：

```
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 | Finland | Phone     | 10 | 10 |
| 2000 | Finland | Computer  | 1500 | 1500 |
| 2001 | USA     | Calculator | 50 | 50 |
| 2001 | USA     | Computer  | 1500 | 1500 |
| 2000 | India   | Calculator | 75 | 75 |
| 2000 | India   | Calculator | 75 | 75 |
| 2001 | India   | Calculator | 79 | 79 |
```

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 |
```

12.2. 聚合函数

本文介绍云原生数据仓库AnalyticDB MySQL中的聚合函数。

- 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/STDDEV**：返回数值的样本标准偏差。
- 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）的总体标准差。
- SUM**：该函数用于计算汇总值。
- VAR_POP**：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。
- VARIANCE**：返回一组数值（整数、小数或浮点）的总体方差。

本文中的聚合函数均以 testtable 表为测试数据。

```
create table testtable(a int) distributed by hash(a);
```

```
insert into testtable values (1),(2),(3);
```

AVG

avg(bigint x)
avg(double x)
avg(float x)

- 命令说明：该函数用于计算平均值。
- 返回值类型：DOUBLE。

- 示例：

```
select avg(a) from testtable;
+-----+
| avg(a) |
+-----+
|  2.0 |
```

BIT_AND

```
bit_and(float x)
bit_and(bigint x)
bit_and(double x)
```

- 命令说明：返回参数所有位按位 AND 后的结果。
- 返回值类型：BIGINT。
- 示例：

```
select bit_and(a) from testtable;
+-----+
| bit_and(a) |
+-----+
|      0 |
```

BIT_OR

```
bit_or(float x)
bit_or(bigint x)
bit_or(double x)
```

- 命令说明：返回参数所有位按位 OR 后的结果。
- 返回值类型：BIGINT。
- 示例：

```
select bit_or(a) from testtable;
+-----+
| bit_or(a) |
+-----+
|      3 |
```

BIT_XOR

```
bit_xor(double x)
bit_xor(bigint x)
bit_xor(float x)
```

- 命令说明：返回参数所有位按位异或后的结果。
- 返回值类型：BIGINT。
- 示例：

```
select bit_xor(a) from testtable;
+-----+
| bit_xor(a) |
+-----+
|      0 |
```

COUNT

```
count([distinct|all] value x)
```

- 命令说明：该函数用于计算记录数。
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。
- 返回值类型：BIGINT。
- 示例：

```
select count(distinct a) from testtable;
+-----+
| count(DISTINCT a) |
+-----+
|          3 |
```

MAX

max(value x)

- 命令说明：该函数用于计算最大值。

value 可以为任意数据类型，但是 BOOLEAN 类型的数据不允许参与运算。

当列中的值为 null 时，该行不参与计算。

- 返回值类型：LONG。
- 示例：

```
select max(a) from testtable;
+-----+
| max(a) |
+-----+
|    3 |
```

MIN

min(value x)

- 命令说明：该函数用于计算最小值。

value 可以为任意数据类型，但是 BOOLEAN 类型的数据不允许参与运算。

当列中的值为 null 时，该行不参与计算。

- 返回值类型：LONG。
- 示例：

```
select min(a) from testtable;
+-----+
| min(a) |
+-----+
|    1 |
```

STD/STDDEV

std(double x)
std(bigint x)
stddev(double x)
stddev(bigint x)

- 命令说明：返回数值的样本标准偏差。
- 返回值类型：DOUBLE。
- 示例：

```
select std(a) from testtable;
+-----+
| std(a) |
+-----+
| 0.816496580927726 |
```

STDDEV_POP

stddev_pop(double x)
stddev_pop(bigint x)

- 命令说明：返回数值的总体标准差。
- 返回值类型：DOUBLE。
- 示例：

```
select stddev_pop(a) from testtable;
+-----+
| stddev_pop(a) |
+-----+
| 0.816496580927726 |
```

STDDEV_SAMP

```
stddev_samp(double x)
stddev_samp(bigint x)
```

- 命令说明：返回一组数值（整数、小数或浮点）的总体标准差。
- 返回值类型：DOUBLE。
- 示例：

```
select stddev_samp(a) from testtable;
+-----+
| stddev_samp(a) |
+-----+
| 1.0 |
```

SUM

```
sum(double x)
sum(float x)
sum(bigint x)
```

- 命令说明：该函数用于计算汇总值。
- 返回值类型：BIGINT。
- 示例：

```
select sum(a) from testtable;
+-----+
| sum(a) |
+-----+
| 6 |
```

VAR_POP

```
var_pop(double x)
var_pop(bigint 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。也可以使用VARIANCE()函数，具有相同的意义，但VARIANCE()不是标准的SQL。若找不到匹配的项，则VAR_POP()返回NULL。
- 返回值类型：DOUBLE。
- 示例：

```
select var_pop(a) from testtable;
+-----+
| var_pop(a) |
+-----+
| 0.6666666666666666 |
```

VAR_SAMP

```
var_samp(double x)
var_samp(bigint x)
```

- 命令说明：返回一组数值（整数、小数或浮点）的样本方差。
- 返回值类型：DOUBLE。
- 示例：

```
select var_samp(a) from testtable;
+-----+
| var_samp(a) |
+-----+
| 1.0 |
```

VARIANCE

```
variance(double x)
variance(bigint 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。VARIANCE()作为标准SQL的延伸，也可以使用标准SQL函数 VAR_POP()来代替。若找不到匹配的项，则 VARIANCE()返回NULL。
- 返回值类型：DOUBLE。
- 示例：

```
select variance(a) from testtable;
+-----+
| variance(a) |
+-----+
| 0.6666666666666666 |
```

12.3. 日期和时间函数

12.3.1. ADDDATE/DATE_ADD

ADDDATE/DATE_ADD用于返回添加指定时间后的日期。

```
ADDDATE(date,INTERVAL expr unit)
ADDDATE(expr,days)
```

- 命令说明：
 - 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。
- 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。
- 示例：

```
select adddate(date '2001-1-22',interval '3' day);
+-----+
| adddate(DATE '2001-1-22', INTERVAL '3' DAY) |
+-----+
| 2001-01-25 |
```

```
select adddate(timestamp '2001-1-22',interval '3' day);
+-----+
| adddate(TIMESTAMP '2001-1-22', INTERVAL '3' DAY) |
+-----+
| 2001-01-25 00:00:00 |
```

```
select adddate(datetime '2001-1-22',interval '3' day);
+-----+
| adddate(DATETIME '2001-1-22', INTERVAL '3' DAY) |
+-----+
| 2001-01-25 00:00:00 |
```

```
select adddate('2001-1-22',interval '3' day);
+-----+
| adddate('2001-1-22', INTERVAL '3' DAY) |
+-----+
| 2001-01-25 |
```

```
select adddate(datetime '2001-1-22',interval '3' second);
+-----+
| adddate(DATETIME '2001-1-22', INTERVAL '3' SECOND) |
+-----+
| 2001-01-22 00:00:03 |
```

12.3.2. ADDTIME

ADDTIME用于返回添加指定时间后的时间，即返回expr1增加expr2后的结果。

```
ADDTIME(expr1,expr2)
```

- 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：

```
select addtime(date '1998-01-01','01:01:01');
+-----+
| addtime(DATE '1998-01-01', '01:01:01') |
+-----+
| 1998-01-01 01:01:01 |
```

12.3.3. CONVERT_TZ

CONVERT_TZ用于转换dt，从from_tz转到to_tz给出的时区，并返回结果。

```
CONVERT_TZ(dt,from_tz,to_tz)
```

- 参数类型：

```
convert_tz(varchar, varchar, varchar)
```

- 返回值类型： DATETIME。
- 示例：

```
select convert_tz('2004-01-01 12:00:00','+00:00','+10:00');
+-----+
| convert_tz('2004-01-01 12:00:00', '+00:00', '+10:00') |
+-----+
| 2004-01-01 22:00:00 |
```

12.3.4. CURDATE

CURDATE用于返回当前日期。

```
CURDATE()
```

- 返回值类型： DATE。
- 示例：

```
select curdate;
+-----+
| curdate() |
+-----+
| 2019-05-25 |
```

12.3.5. CURTIME

CURTIME用于返回当前时间。

```
CURTIME()
```

- 返回值类型：TIME。
- 示例：

```
select curtime();
+-----+
| curtime() |
+-----+
| 14:39:22.109 |
```

12.3.6. DATE

DATE用于返回日期或日期时间表达式中的日期。

```
DATE(expr)
```

- 参数类型：
- 返回值类型：DATE。
- 示例：

```
date(timestamp)
date(datetime)
date(varchar)

select date(timestamp '2003-12-31 01:02:03');
+-----+
| date(TIMESTAMP '2003-12-31 01:02:03') |
+-----+
| 2003-12-31 |
```

12.3.7. DATE_FORMAT

DATE_FORMAT按照format指定的格式，将日期时间格式化成字符串。

```
DATE_FORMAT(date,format)
```

- 命令说明：format格式如下所示。

符号	说明
%a	工作日缩写名称（Sun.. Sat）
%b	缩写的月份名称（Jan.. Dec）
%c	月，数字（0.. 12）
%d	每月的某一天，数字（00.. 31）
%e	每月的某一天，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	一年中的一天（001.. 366）
%k	小时（0.. 23）
%l	小时（1.. 12）
%M	月份名称（January.. December）
%m	月，数字（00.. 12）
%p	AM或PM
%r	时间，12小时（hh:mm:ss其次是AM或PM）
%S	秒（00... 59）

符号	说明
%s	秒（00... 59）
%T	时间，24小时（hh:mm:ss）
%v	本周是当年的第几周，星期一是一周的第一天，WEEK()模式3；与%x使用
%W	工作日名称（Sunday.. Saturday）
%x	本周所属年份，星期一是一周的第一天，四位数；与%v使用
%Y	年份，数字，四位数
%y	年份，数字，两位数
%%	文字%字符
%x	x，对于上面未列出的任何x

• 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：

```
select date_format(timestamp '2019-05-27 13:23:00', '%W %M %Y')as result;
+-----+
| result |
+-----+
| Monday May 2019 |
```

12.3.8. SUBDATE/DATE_SUB

SUBDATE/DATE_SUB用于返回date减去指定INTERVAL间隔后的日期。

DATE_SUB(date,INTERVAL expr unit)

- 命令说明：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：

```
select date_sub(date '2001-1-22',interval '3' day);
+-----+
| date_sub(DATE '2001-1-22', INTERVAL '3' DAY) |
+-----+
| 2001-01-19 |
```

12.3.9. DATEDIFF

DATEDIFF用于返回expr1减去expr2后的天数。

DATEDIFF(expr1,expr2)

- 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：

```
select datediff('2007-12-31 23:59:59','2007-12-30');
+-----+
| datediff('2007-12-31 23:59:59', '2007-12-30') |
+-----+
| 1 |
```

12.3.10. DAY/DAYOFMONTH

DAY/DAYOFMONTH用于返回date中的日，取值范围 [1,31] 。

```
DAY(date)
DAYOFMONTH(date)
```

- 参数类型：

```
dayofmonth(timestamp)
dayofmonth(datetime)
dayofmonth(date)
dayofmonth(time)
dayofmonth(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select dayofmonth(timestamp '2007-02-03 12:23:09');
+-----+
| dayofmonth(TIMESTAMP '2007-02-03 12:23:09') |
+-----+
| 3 |
```

12.3.11. DAYNAME

DAYNAME用于返回日期对应的工作日的名称，例如星期一为Monday。

```
DAYNAME(date)
```

- 参数类型：

```
dayname(timestamp)
dayname(datetime)
dayname(date)
dayname(varchar)
```

- 返回值类型：VARCHAR。
- 示例：

```
select dayname(timestamp '2007-02-03 00:00:00');
+-----+
| dayname(TIMESTAMP '2007-02-03 00:00:00') |
+-----+
| Saturday |
```

12.3.12. DAYOFWEEK

DAYOFWEEK用于返回日期对应的工作日索引值，即星期日为1，星期一为2，星期六为7。

DAYOFWEEK(date)

- 参数类型：

```
dayofweek(timestamp)
dayofweek(datetime)
dayofweek(date)
dayofweek(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select dayofweek(timestamp '2007-02-03 00:00:00');
+-----+
| dayofweek(TIMESTAMP '2007-02-03 00:00:00') |
+-----+
|                7 |
```

12.3.13. DAYOFYEAR

DAYOFYEAR用于返回指定日期是当年的哪一天，返回值范围为 [1,366] 。

DAYOFYEAR(date)

- 参数类型：

```
dayofyear(timestamp)
dayofyear(datetime)
dayofyear(date)
dayofyear(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select dayofyear(timestamp '2007-02-03 00:12:12');
+-----+
| dayofyear(TIMESTAMP '2007-02-03 00:12:12') |
+-----+
|                34 |
```

12.3.14. EXTRACT

EXTRACT用于返回日期或时间的单独部分，由unit指定，例如年、月、日、小时、分钟等。

EXTRACT(unit FROM date)

- 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。
- 参数类型：VARCHAR、TIMESTAMP、DATETIME、TIME。
- 返回值类型：BIGINT。
- 示例：

```
select extract(second from '2019-07-02 00:12:34');
+-----+
| _col0 |
+-----+
|    34 |
```

12.3.15. FROM_DAYS

FROM_DAYS用于根据指定的天数N，返回对应的DATE值。

FROM_DAYS(N)

- 参数类型：

```
from_days(varchar)
from_days(bigint)
```

- 返回值类型：DATE。
- 示例：

```
select from_days(730669);
+-----+
| from_days(730669) |
+-----+
| 2000-07-03      |
```

12.3.16. FROM_UNIXTIME

FROM_UNIXTIME用于返回unix time时间戳。

```
FROM_UNIXTIME(unix_timestamp[,format])
```

- 命令说明：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：

```
select from_unixtime('1447430881','%Y %M %h:%i:%s %x');
+-----+
| from_unixtime('1447430881','%Y %M %h:%i:%s %x') |
+-----+
| 2015 November 12:08:01 2015 |
```

12.3.17. HOUR

HOUR用于返回时间中的小时。

```
HOUR(time)
```

- 参数类型：

```
hour(timestamp)
hour(datetime)
hour(date)
hour(time)
hour(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select hour(timestamp '2019-12-07 10:05:03');
+-----+
| hour(TIMESTAMP '2019-12-07 10:05:03') |
+-----+
| 10 |
```

12.3.18. LAST_DAY

LAST_DAY用于返回日期或者日期时间中对应月份的最后一天。

```
LAST_DAY(date)
```

- 参数类型：

```
last_day(varchar)
last_day(timestamp)
last_day(datetime)
last_day(date)
```

- 返回值类型：DATE。
- 示例：

```
select last_day('2003-02-05');
+-----+
| last_day('2003-02-05') |
+-----+
| 2003-02-28             |
```

12.3.19. LOCALTIME/LOCALTIMESTAMP/NOW

LOCALTIME/LOCALTIMESTAMP/NOW用于返回当前时间戳。

```
localtime
localtime()
localtimestamp
localtimestamp()
now()
```

- 返回值类型：DATETIME。
- 示例：

```
select now();
+-----+
| now() |
+-----+
| 2019-05-25 00:28:37 |
```

12.3.20. MAKEDATE

MAKEDATE用于按照参数year和dayofyear，返回一个日期。

```
MAKEDATE(year,dayofyear)
```

- 参数类型：

```
makedate(bigint, bigint)
makedate(varchar, varchar)
```

- 返回值类型：DATE。
- 示例：

```
select makedate(2011,31), makedate(2011,32);
+-----+-----+
| makedate(2011, 31) | makedate(2011, 32) |
+-----+-----+
| 2011-01-31        | 2011-02-01         |
```

12.3.21. MAKETIME

MAKETIME用于按照参数hour、minute和second，返回一个时间。

```
MAKETIME(hour,minute,second)
```

- 参数类型：

```
maketime(bigint, bigint, bigint)
maketime(varchar, varchar, varchar)
```

- 返回值类型：TIME。
- 示例：

```
select maketime(12,15,30);
+-----+
| maketime(12, 15, 30) |
+-----+
| 12:15:30             |
```

12.3.22. MINUTE

MINUTE用于返回时间中的分钟。

```
MINUTE(time)
```

- 参数类型：

```
minute(timestamp)
minute(datetime)
minute(date)
minute(time)
minute(varchar)
```

- 返回值类型：BIGINT。

- 示例：

```
select minute(timestamp '2008-02-03 10:05:03');
+-----+
| minute(TIMESTAMP '2008-02-03 10:05:03') |
+-----+
|                    5 |
```

12.3.23. MONTH

MONTH用于返回日期中的月份。

MONTH(date)

- 参数类型：

```
month(timestamp)
month(datetime)
month(date)
month(time)
month(varchar)
```

- 返回值类型：BIGINT。

- 示例：

```
select month(timestamp '2008-02-03 00:00:00');
+-----+
| month(TIMESTAMP '2008-02-03 00:00:00') |
+-----+
|                    2 |
```

12.3.24. MONTHNAME

MONTHNAME用于返回日期中月份的全名。

MONTHNAME(date)

- 参数类型：

```
monthname(timestamp)
monthname(datetime)
monthname(date)
monthname(varchar)
```

- 返回值类型：VARCHAR。

- 示例：

```
select monthname(datetime '2008-02-03 00:00:00');
+-----+
| monthname(DATETIME '2008-02-03 00:00:00') |
+-----+
| February |
```

12.3.25. PERIOD_ADD

PERIOD_ADD用于将日期格式的参数字P增加N个月。

PERIOD_ADD(P,N)

- 参数类型：

```
period_add(bigint, bigint)
period_add(varchar, varchar)
period_add(varchar, bigint)
```

- 返回值类型：BIGINT。
- 示例：

```
select period_add(200801,2);
+-----+
| period_add(200801, 2) |
+-----+
|          200803      |
```

12.3.26. PERIOD_DIFF

PERIOD_DIFF用于返回P1和P2之间相差的月数。

```
PERIOD_DIFF(P1,P2)
```

- 参数类型：

```
period_diff(bigint, bigint)
period_diff(varchar, varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select period_diff(200802,200703);
+-----+
| period_diff(200802, 200703) |
+-----+
|              11            |
```

12.3.27. QUARTER

QUARTER用于返回日期在一年中的季度，取值范围为 [1,4] 。

```
QUARTER(date)
```

- 参数类型：

```
quarter(datetime)
quarter(varchar)
quarter(timestamp)
quarter(date)
```

- 返回值类型：BIGINT。
- 示例：

```
select quarter(datetime '2008-04-01 12:12:12');
+-----+
| quarter(DATETIME '2008-04-01 12:12:12') |
+-----+
|              2                        |
```

12.3.28. SEC_TO_TIME

SEC_TO_TIME用于将seconds转换为时间。

```
SEC_TO_TIME(seconds)
```

- 参数类型：

```
sec_to_time(bigint)
sec_to_time(varchar)
```

- 返回值类型：TIME。
- 示例：

```
select sec_to_time(2378);
+-----+
| sec_to_time(2378) |
+-----+
| 00:39:38         |
```

12.3.29. SECOND

SECOND用于返回时间中的秒，范围为 [0,59] 。

SECOND(time)

- 参数类型：

```
second(timestamp)
second(datetime)
second(date)
second(time)
second(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select second(timestamp '2019-03-12 12:13:14');
+-----+
| second(TIMESTAMP '2019-03-12 12:13:14') |
+-----+
|                      14 |
```

12.3.30. STR_TO_DATE

STR_TO_DATE用于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。

STR_TO_DATE(str,format)

- 参数类型：

str_to_date(varchar, varchar)

- 返回值类型：DATETIME。
- 示例：

```
select str_to_date('2017-01-06 10:20:30','%Y-%m-%d %H:%i:%s') as result;
+-----+
| result      |
+-----+
| 2017-01-06 10:20:30 |
```

12.3.31. SUBTIME

SUBTIME用于返回expr1减去expr2后的时间。

SUBTIME(expr1,expr2)

- 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：

```
select subtime(date '2018-10-31','0:1:1');
+-----+
| subtime(DATE '2018-10-31', '0:1:1') |
+-----+
| 2018-10-30 23:58:59 |
```

12.3.32. SYSDATE

SYSDATE用于获取系统时间。

```
SYSDATE()
```

- 返回值类型：DATETIME。
- 示例：

```
select sysdate();
+-----+
| sysdate() |
+-----+
| 2019-05-26 00:47:21 |
```

12.3.33. TIME

TIME用于以字符串形式返回expr中的时间。

```
TIME(expr)
```

- 参数类型：

```
time(varchar)
time(datetime)
time(timestamp)
```

- 返回值类型：VARCHAR。
- 示例：

```
select time('2003-12-31 01:02:03');
+-----+
| time('2003-12-31 01:02:03') |
+-----+
| 01:02:03 |
```

12.3.34. TIME_FORMAT

TIME_FORMAT用于按照format指定的格式，以字符串格式显示时间time。

```
TIME_FORMAT(time,format)
```

- 命令说明：format遵从DATE_FORMAT中的格式。
- 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：

```
select time_format('12:00:00', '%H %k %h %l %I');
+-----+
| time_format('12:00:00', '%H %k %h %l %I') |
+-----+
| 12 12 12 12 12 |
```

12.3.35. TIME_TO_SEC

TIME_TO_SEC用于返回time转换为秒的结果。

```
TIME_TO_SEC(time)
```

- 参数类型：

```
time_to_sec(vchar)  
time_to_sec(datetime)  
time_to_sec(timestamp)  
time_to_sec(date)  
time_to_sec(time)
```

- 返回值类型：BIGINT。
- 示例：

```
select time_to_sec(datetime '2009-12-12 22:23:00');  
+-----+  
| time_to_sec(DATETIME '2009-12-12 22:23:00') |  
+-----+  
|                80580 |
```

12.3.36. TIMEDIFF

TIMEDIFF用于返回expr1减去expr2后的时间。

```
TIMEDIFF(expr1,expr2)
```

- 参数类型：

```
timediff(time, varchar)  
timediff(time, time)  
timediff(vchar, varchar)
```

- 返回值类型：DATETIME。
- 示例：

```
select timediff(time '12:00:00','10:00:00');  
+-----+  
| timediff(TIME '12:00:00','10:00:00') |  
+-----+  
| 02:00:00 |
```

12.3.37. TIMESTAMP

TIMESTAMP用于返回expr表示的日期或日期时间。

```
TIMESTAMP(expr)
```

- 参数类型：

```
timestamp(date)  
timestamp(vchar)
```

- 返回值类型：DATETIME。
- 示例：

```
select timestamp(date '2019-05-27');  
+-----+  
| timestamp(DATE '2019-05-27') |  
+-----+  
| 2019-05-27 00:00:00 |
```

12.3.38. TIMESTAMPADD

TIMESTAMPADD用于将interval添加到日期或日期时间表达式datetime_expr中。

```
TIMESTAMPADD(unit,interval,datetime_expr)
```

- 命令说明：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(vchar, varchar, timestamp)
timestampadd(vchar, bigint, timestamp)
timestampadd(vchar, varchar, date)
timestampadd(vchar, bigint, date)
timestampadd(vchar, varchar, datetime)
timestampadd(vchar, bigint, datetime)
timestampadd(vchar, varchar, varchar)
timestampadd(vchar, bigint, varchar)
```

- 返回值类型：DATETIME。
- 示例：

```
select timestampadd(second,'1',timestamp '2003-01-02 12:12:12')as result;
+-----+
| result |
+-----+
| 2003-01-02 12:12:13 |
```

12.3.39. TIMESTAMPDIFF

TIMESTAMPDIFF用于返回日期或日期时间表达式datetime_expr1减去datetime_expr2后的结果，结果的单位由unit指定。

```
TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)
```

- 命令说明：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(vchar, timestamp, timestamp)
timestampdiff(vchar, date, date)
timestampdiff(vchar, datetime, datetime)
timestampdiff(vchar, varchar, varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select timestampdiff(second,datetime '2003-02-01 10:12:13',datetime '2003-05-01 10:12:13')as result;
+-----+
| result |
+-----+
| 7689600 |
```

12.3.40. TO_DAYS

TO_DAYS用于根据给定日期date，返回自0年开始的天数。

```
TO_DAYS(date)
```

- 参数类型：

```
to_days(date)
to_days(time)
to_days(vchar)
to_days(timestamp)
to_days(datetime)
```

- 返回值类型：BIGINT。
- 示例：

```
select to_days(date '2018-12-12');
+-----+
| to_days(DATE '2018-12-12') |
+-----+
| 737405 |
```

12.3.41. TO_SECONDS

TO_SECONDS用于根据给定的expr，返回自0年开始的秒数。

```
TO_SECONDS(expr)
```

- 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：

```
select to_seconds(date '2019-09-08');
+-----+
| to_seconds(DATE '2019-09-08') |
+-----+
|      63735120000 |
```

12.3.42. UNIX_TIMESTAMP

UNIX_TIMESTAMP用于返回自 '1970-01-01 00:00:00' UTC以来秒数的Unix时间

```
UNIX_TIMESTAMP([date])
```

- 命令说明：UNIX_TIMESTAMP(date) 将参数的值返回为 '1970-01-01 00:00:00' UTC后的秒数的Unix时间戳。
- 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：

```
select unix_timestamp();
+-----+
| unix_timestamp() |
+-----+
|      1558935850 |
```

12.3.43. UTC_DATE

UTC_DATE用于返回UTC日期。

```
UTC_DATE()
```

- 返回值类型：VARCHAR。
- 示例：

```
select utc_date();
+-----+
| utc_date() |
+-----+
| 2019-05-27 |
```

12.3.44. UTC_TIME

UTC_TIME用于返回UTC时间。

```
UTC_TIME()
```

- 返回值类型：VARCHAR
- 示例：

```
select utc_time();
+-----+
| utc_time() |
+-----+
| 05:53:19 |
```

12.3.45. UTC_TIMESTAMP

UTC_TIMESTAMP用于返回UTC时间戳。

```
utc_timestamp()
```

- 返回值类型： VARCHAR。
- 示例：

```
select utc_timestamp();
+-----+
| utc_timestamp() |
+-----+
| 2019-05-27 05:55:15 |
```

12.3.46. WEEK

WEEK用于返回date对应的周数，即date是日期年份中的哪一周。

```
WEEK(date[,mode])
```

- 命令说明：
 - 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。它允许您指定本周是从星期一还是星期日开始，返回的周数应在0到52之间或0到53之间。mode支持的格式如下表所示。

模式	一周的第一天	范围
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：

```
select week('2019-05-27');
+-----+
| week('2019-05-27') |
+-----+
|          21 |
```

12.3.47. WEEKDAY

WEEKDAY用于返回date对应的工作日即 0= Monday 、 1= Tuesday 、 6= Sunday 。

```
WEEKDAY(date)
```

- 参数类型：

```
weekday(timestamp)
weekday(datetime)
weekday(date)
weekday(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select weekday(timestamp '2019-05-27 00:09:00');
+-----+
| weekday(TIMESTAMP '2019-05-27 00:09:00') |
+-----+
|                0 |
```

12.3.48. WEEKOFYEAR

WEEKOFYEAR用于返回date对应的日历周，取值范围为 [1,53] 。

```
WEEKOFYEAR(date)
```

- 参数类型：

```
weekofyear(timestamp)
weekofyear(datetime)
weekofyear(date)
weekofyear(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select weekofyear(timestamp '2019-05-27 09:00:00');
+-----+
| weekofyear(TIMESTAMP '2019-05-27 09:00:00') |
+-----+
|                22 |
```

12.3.49. YEAR

YEAR用于返回date中的年份。

```
YEAR(date)
```

- 参数类型：

```
year(timestamp)
year(datetime)
year(date)
year(time)
year(varchar)
```

- 返回值类型：BIGINT。
- 示例：

```
select year(timestamp '2019-05-27 00:00:00');
+-----+
| year(TIMESTAMP '2019-05-27 00:00:00') |
+-----+
|                2019 |
```

12.3.50. YEARWEEK

YEARWEEK用于返回日期的年份和星期。

```
YEARWEEK(date)
YEARWEEK(date,mode)
```

- 命令说明：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：

```
select yearweek(timestamp '2019-05-27 00:00:00');
+-----+
| yearweek(TIMESTAMP '2019-05-27 00:00:00') |
+-----+
|                201921 |
```

12.4. 字符串函数

12.4.1. ASCII

ASCII用于返回字符串str或者字符串str最左边字符对应的十进制ASCII值。

```
ASCII(varchar str)
```

- 返回值类型：BIGINT。
- 示例：

```
select ascii('2');
+-----+
| ascii('2') |
+-----+
|         50 |
```

12.4.2. BIN

BIN用于返回N的二进制字符串。

```
BIN(bigint N)
```

- 命令说明：如果N为null，则返回结果为null。
- 返回值类型：VARCHAR。
- 示例：

```
select bin(12);
+-----+
| bin(12) |
+-----+
| 1100 |
```

12.4.3. BIT_LENGTH

BIT_LENGTH用于以位为单位返回参数str的长度。

```
BIT_LENGTH(varchar str)
```

- 返回值类型：BIGINT。
- 示例：

```
select bit_length('text');
+-----+
| bit_length('text') |
+-----+
|          32 |
```

12.4.4. CHAR

CHAR用于返回整数N1、N2...对应的十进制ASCII码组成的字符串。

CHAR(bigint N1, bigint N2...)

- 返回值类型：VARBINARY。
- 示例：

```
select char(97,110,97,108,121,116,105,99,100,98);
+-----+
| char(97, 110, 97, 108, 121, 116, 105, 99, 100, 98) |
+-----+
| analyticdb      |
```

12.4.5. CHAR_LENGTH/CHARACTER_LENGTH

CHAR_LENGTH/CHARACTER_LENGTH用于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。

CHAR_LENGTH(varchar str)

- 返回值类型：BIGINT。
- 示例：

```
select char_length('中国');
+-----+
| char_length('中国') |
+-----+
|          2          |
```

12.4.6. CONCAT

CONCAT用于字符串连接操作，其中任何一个参数为null，则返回值为null。

concat(varchar str1, ..., varchar strn)

- 返回值类型：VARCHAR。
- 示例：

```
select concat('aliyun',' ','analyticdb');
+-----+
| concat('aliyun',' ','analyticdb') |
+-----+
| aliyun,analyticdb      |
```

12.4.7. CONCAT_WS

CONCAT_WS用于字符串连接操作，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。

concat_ws(varchar separator, varchar str1, ..., varchar strn)

- 返回值类型：VARCHAR。
- 示例：

```
select concat_ws(',', 'First name', 'Second name', 'Last Name') as result;
+-----+
| result      |
+-----+
| First name,Second name,Last Name |
```

12.4.8. ELT

ELT用于返回第N个字符串。

ELT(bigint N, varchar str1, varchar str2, varchar str3,...)

- 命令说明：若N<1或大于后面字符串参数的数量，则返回结果为null。
- 返回值类型：VARCHAR。
- 示例：

```
select elt(4, 'Aa', 'Bb', 'Cc', 'Dd');
+-----+
| elt(4, 'Aa', 'Bb', 'Cc', 'Dd') |
+-----+
| Dd          |
```

12.4.9. EXPORT_SET

EXPORT_SET用于返回一个字符串，根据整数bits的二进制位01值，从右到左（从低位到高位）放置on、off字符串。

```
EXPORT_SET(bigint bits, varchar on, varchar off[, varchar separator[, bigint number_of_bits]])EXPORT_SET(bigint bits, varchar on, varchar off[, varchar separator[, bigint number_of_bits]])
```

- 命令说明：1的位置放on字符串，0的位置放off字符串，由分隔符字符串（默认为逗号字符）分隔。检查位数由number_of_bits指定，如果未指定，默认值为64。如果检查位数大于64，number_of_bits将被静默剪裁为64。检查位数为-1、64，返回结果相同。
- 返回值类型：VARCHAR。
- 示例：

```
select export_set(5,'1','0',',',2);
+-----+
| export_set(5, '1', '0', ',', 2) |
+-----+
| 1,0          |
```

12.4.10. FIELD

FIELD用于返回str在str1、str2、str3、...列表中的索引位置。如果未找到str，则返回0。

```
field(varchar str, varchar str1, varchar str2, varchar str3,...)
```

- 返回值类型：BIGINT。
- 示例：

```
select field('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
+-----+
| field('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff') |
+-----+
| 2 |
```

12.4.11. FIND_IN_SET

FIND_IN_SET用于返回str在列表strlist中的位置。

```
FIND_IN_SET(varchar str, varchar strlist)
```

- 命令说明：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。
- 返回值类型：BIGINT。
- 示例：

```
select find_in_set('b', 'a,b,c,d');
+-----+
| find_in_set('b', 'a,b,c,d') |
+-----+
| 2 |
```

12.4.12. FORMAT

FORMAT用于将数字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。

```
format(double X, bigint D)
```

- 命令说明：如果D为0，则返回结果没有小数点或小数部分。。
- 返回值类型：BIGINT。
- 示例：

```
select format(12332.123456,4)as result1,format(12332.1,4)as result2,format(12332.2,0)as result3;
+-----+-----+-----+
| result1 | result2 | result3 |
+-----+-----+-----+
| 12,332.1235 | 12,332.1000 | 12,332 |
```

12.4.13. HEX

HEX用于返回整数N所对应的十六进制字符串，或者返回str中每个字符对应的十六进制数所组成的字符串。

```
HEX(bigint N)
HEX(varchar str)
```

- 返回值类型：VARCHAR。
- 示例：

```
select hex(16);
+-----+
| hex(16) |
+-----+
| 10      |
```

12.4.14. INSTR

INSTR用于返回字符串str中子字符串substr首次出现的位置。

```
INSTR(varchar str, varchar substr)
```

- 返回值类型：BIGINT。
- 示例：

```
select instr('foobarbar','bar');
+-----+
| instr('foobarbar','bar') |
+-----+
|          4               |
```

12.4.15. LEFT

LEFT用于返回字符串str中最左边的len个字符。

```
LEFT(varchar str, bigint len)
```

- 命令说明：如果str或者len为null，则返回结果为null。
- 返回值类型：VARCHAR。
- 示例：

```
select left('foobarbar',5);
+-----+
| left('foobarbar',5) |
+-----+
| fooba               |
```

12.4.16. LENGTH/OCTET_LENGTH

LENGTH/OCTET_LENGTH用于返回字符串str的长度。

```
length(varchar str)
```

- 返回值类型：BIGINT。
- 示例：

```
select length('aliyun');
+-----+
| length('aliyun') |
+-----+
|          6       |
```

12.4.17. LIKE

LIKE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。

```
expression [ NOT ] LIKE pattern [ESCAPE 'escape_char']
```

- 命令说明：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。

- 返回值类型：BIGINT。
- 示例：

```
select 'David!' like 'David_' as result1, 'David!' not like 'David_' as result2, 'David!' like '%D%v%' as result3;
+-----+-----+-----+
| result1 | result2 | result3 |
+-----+-----+-----+
| 1 | 0 | 1 |
```

12.4.18. LOCATE

LOCATE用于返回字符串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。

```
LOCATE(varchar substr, varchar str)
LOCATE(varchar substr, varchar str, bigint pos)
```

- 命令说明：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。
- 返回值类型：BIGINT。
- 示例：

```
select locate('bar', 'foobarbar');
+-----+
| locate('bar', 'foobarbar') |
+-----+
| 4 |
```

12.4.19. LOWER/LCASE

LOWER/LCASE用于将字符串str中的字母转换为小写。

```
lower(varchar str)
```

- 返回值类型：VARCHAR。
- 示例：

```
select lower('Aliyun');
+-----+
| lower('Aliyun') |
+-----+
| aliyun |
```

12.4.20. LPAD

LPAD用于将字符串str左边拼接padstr直到长度达到len，并返回拼接后的字符串。

```
lpad(varchar str, bigint len, varchar padstr)
```

- 命令说明：如果str长于len，则返回值将缩短为len个字符。
- 返回值类型：VARCHAR。
- 示例：

```
select lpad('Aliyun',9,'#');
+-----+
| lpad('Aliyun',9,'#') |
+-----+
| ###Aliyun |
```

12.4.21. LTRIM

LTRIM用于删除字符串str所有前导空格。

LTRIM(varchar str)

- 返回值类型：VARCHAR。
- 示例：

```
select ltrim(' abc');
+-----+
| ltrim(' abc') |
+-----+
| abc          |
```

12.4.22. MAKE_SET

MAKE_SET用于返回一个设置值（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。

MAKE_SET(bits, str1, str2,...)

- 命令说明：str1对应于0位，str2对应于1位，依此类推。str1, str2, ...中的null值不会附加到结果中。
- 返回值类型：VARCHAR。
- 示例：

```
select make_set(5,'hello','nice','world');
+-----+
| make_set(5,'hello','nice','world') |
+-----+
| hello,world                        |
```

12.4.23. MID

MID用于从字符串str的pos开始返回len长度的子字符串。

MID(varchar str, bigint pos, bigint len)

- 返回值类型：VARCHAR。
- 示例：

```
select mid('Quadratically',5,6);
+-----+
| mid('Quadratically', 5, 6) |
+-----+
| ratica                      |
```

12.4.24. OCT

OCT用于返回整数N的八进制字符串表示形式。

OCT(bigint N)

- 命令说明：如果N为null，返回结果为null。
- 返回值类型：VARCHAR。
- 示例：

```
select oct(12);
+-----+
| oct(12) |
+-----+
| 14      |
```

12.4.25. POSITION

POSITION用于返回字符串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。

POSITION(varchar substr IN varchar str)

- 返回值类型：BIGINT。
- 示例：

```
select position('bar' in 'foobarbar');
+-----+
| locate('bar', 'foobarbar') |
+-----+
|          4 |
```

12.4.26. REPEAT

REPEAT用于返回由字符串str重复count次数组成的字符串。

REPEAT(varchar str, bigint count)

- 命令说明：如果count<1，则返回空字符串。
如果str或count为null，则返回null。
- 返回值类型：VARCHAR。
- 示例：

```
select repeat('a', 3);
+-----+
| repeat('a', 3) |
+-----+
| aaa          |
```

12.4.27. REPLACE

REPLACE用于将str中的from_str内容替换为to_str。

REPLACE(varchar str, varchar from_str, varchar to_str)

- 返回值类型：VARCHAR。
- 示例：

```
select replace('WWW.aliyun.com', 'W', 'w');
+-----+
| replace('WWW.aliyun.com', 'W', 'w') |
+-----+
| www.aliyun.com                      |
```

12.4.28. REVERSE

REVERSE用于返回str逆序后的字符串。

REVERSE(varchar str)

- 返回值类型：VARCHAR。
- 示例：

```
select reverse('123456');
+-----+
| reverse('123456') |
+-----+
| 654321           |
```

12.4.29. RIGHT

RIGHT用于返回字符串str中最右边的len个字符。

RIGHT(varchar str, bigint len)

- 命令说明：如果str或者len为null，返回结果为null。
- 返回值类型：VARCHAR。
- 示例：

```
select right('abc', 3);
+-----+
| presto_right('abc', 3) |
+-----+
| abc                    |
```

12.4.30. RLIKE/REGEXP

RLIKE/REGEXP用于将字符串expression与pattern进行正则匹配，匹配成功返回1，否则返回0。

```
expression RLIKE pattern
expression REGEXP pattern
```

- 命令说明：如果expression或者pattern为null，返回结果为null。
- 返回值类型：BIGINT。
- 示例：

```
select 'Michael!' regexp '.*';
+-----+
| regexp_like('Michael!','.*') |
+-----+
| 1 |
```

12.4.31. RPAD

RPAD用于将字符串str右边拼接padstr直到长度达到len，并返回拼接后的字符串。

```
rpad(varchar str, bigint len, varchar padstr)
```

- 命令说明：如果str长于len，则返回值将缩短为len个字符。
- 返回值类型：VARCHAR。
- 示例：

```
select rpad('Aliyun',9,'#');
+-----+
| rpad('Aliyun',9,'#') |
+-----+
| Aliyun### |
```

12.4.32. RTRIM

RTRIM用于删除字符串str所有后置空格。

```
RTRIM(varchar str)
```

- 返回值类型：VARCHAR。
- 示例：

```
select rtrim('barbar ');
+-----+
| rtrim('barbar ') |
+-----+
| barbar |
```

12.4.33. SPACE

SPACE用于返回由指定数量空格组成的字符串。

```
SPACE(bigint N)
```

- 返回值类型：VARCHAR。
- 示例：

```
select concat("#",space(6),"#");
+-----+
| concat('#',space(6),'#') |
+-----+
| # # |
```

12.4.34. STRCMP

如果字符串str1、str2相同，返回结果为0。如果str1根据当前排序顺序小于str2，返回结果为-1，否则返回结果为1。

```
STRCMP(varchar str1, varchar str2)
```

- 返回值类型：BIGINT。

- 示例：

```
select strcmp('text','text2');
+-----+
| strcmp('text','text2') |
+-----+
|          -1          |
```

12.4.35. SUBSTR/SUBSTRING

```
SUBSTRING(varchar str, bigint pos)
SUBSTRING(varchar str FROM pos)
SUBSTRING(varchar str, bigint pos, bigint len)
SUBSTRING(varchar 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。
- 返回值类型： VARCHAR。
- 示例：

```
select substr('helloworld', 6);
+-----+
| substr('helloworld', 6) |
+-----+
| world                   |
```

12.4.36. SUBSTRING_INDEX

SUBSTRING_INDEX用于返回字符串str中最后一次分隔符delim出现之前的子字符串。

```
SUBSTRING_INDEX(varchar str, varchar delim, bigint count)
```

- 命令说明：
 - 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。
- 返回值类型： VARCHAR。
- 示例：

```
select substring_index('www.aliyun.com', '.', 2);
+-----+
| substring_index('www.aliyun.com', '.', 2) |
+-----+
| www.aliyun                                |
```

12.4.37. TRIM

TRIM用于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匹配的字符来剪裁字符串。

```
TRIM([remstr FROM] str)
TRIM([BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

- 返回值类型： VARCHAR。
- 示例：

```
select trim(' bar ');
+-----+
| trim(' bar ') |
+-----+
| bar          |
```

12.4.38. UPPER/UCASE

UPPER/UCASE用于将字符串str中的字母转换为大写。

```
upper(varchar str)
```

- 返回值类型：VARCHAR。
- 示例：

```
select upper('Aliyun');
+-----+
| upper('Aliyun') |
+-----+
| ALIYUN          |
```

12.5. 数值函数

12.5.1. ABS

ABS用于返回x的绝对值。

```
abs(tinyint x)
abs(smallint x)
abs(int x)
abs(bigint x)
abs(float x)
abs(double x)
abs(decimal x)
```

- 返回值类型：LONG、DECIMAL或DOUBLE。
- 示例：

```
select abs(4.5);
+-----+
| abs(4.5) |
+-----+
| 4.5      |
```

12.5.2. ROUND

ROUND用于将x四舍五入，d是要保留的小数位数，默认d为0，舍入算法取决于x的数据类型。

```
round(tinyint x)
round(smallint x)
round(int x)
round(bigint x)
round(float x)
round(double x)
round(x, d)
```

- 命令说明：
 - 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，则四舍五入到最近的整数。
 - 如果d<0，则在小数点左侧进行四舍五入。
- 返回值类型：LONG、DECIMAL或DOUBLE。
- 示例：

```
select round(4);
+-----+
| round(4) |
+-----+
| 4        |
```

12.5.3. SQRT

SQRT用于返回x的平方根。

```
sqrt(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select sqrt(4.222);
+-----+
| sqrt(4.222) |
+-----+
| 2.054750593137766 |
```

12.5.4. LN

LN用于返回x的自然对数。

```
ln(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select ln(2.718281828459045);
+-----+
| ln(2.718281828459045) |
+-----+
| 1.0 |
```

12.5.5. LOG

LOG调用一个参数时，返回x的自然对数。调用两个参数时，返回以x为底的y的对数。

- 返回值类型：DOUBLE。
- 示例：

```
select log(16);
+-----+
| log(16) |
+-----+
| 2.772588722239781 |
```

12.5.6. LOG2

LOG2用于返回以2为底的对数。

- 返回值类型：DOUBLE。
- 示例：

```
select log2(8.7654);
+-----+
| log2(8.7654) |
+-----+
| 3.131819928389146 |
```

12.5.7. PI

PI用于返回圆周率。

```
pi()
```

- 返回值类型：DOUBLE。
- 示例：

```
select pi();
+-----+
| pi() |
+-----+
| 3.141592653589793 |
```

12.5.8. LOG10

LOG10用于返回以10为底的对数。

```
log10(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select log10(100.876);
+-----+
| log10(100.876) |
+-----+
| 2.0037878529824615 |
```

12.5.9. POWER/POW

POWER/POW用于返回x的y次幂。

```
power(double x, double y)
pow(double x, double y)
```

- 返回值类型：DOUBLE。
- 示例：

```
select power(1.2,3.4);
+-----+
| power(1.2, 3.4) |
+-----+
| 1.858729691979481 |
```

12.5.10. RADIANS

RADIANS用于将角度转换为弧度。

```
radians(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select radians(60.0);
+-----+
| radians(60.0) |
+-----+
| 1.0471975511965976 |
```

12.5.11. DEGREES

DEGREES用于将弧度转换为度。

```
degrees(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select degrees(1.3);
+-----+
| degrees(1.3) |
+-----+
| 74.48451336700703 |
```

12.5.12. SIGN

SIGN用于返回数字x的符号的值。

```
sign(smallint x)
sign(tinyint x)
sign(int x)
sign(bigint x)
sign(float x)
sign(double x)
sign(decimal x)
```

- 返回值类型：LONG。
- 示例：


```
select sign(12);
+-----+
| sign(12) |
+-----+
|    1    |
```

12.5.13. CEILING/CEIL

CEILING/CEIL用于返回大于x的最小整数值。

```
ceiling(tinyint x)
ceiling(smallint x)
ceiling(int x)
ceiling(bigint x)
ceiling(float x)
ceiling(double x)
```

- 返回值类型：LONG、DECIMAL或DOUBLE。
- 示例：

```
select ceiling(4);
+-----+
| ceiling(4) |
+-----+
|    4    |
```

12.5.14. FLOOR

FLOOR用于返回小于x的最大整数值。

```
floor(tinyint x)
floor(smallint x)
floor(int x)
floor(bigint x)
floor(float x)
floor(double x)
```

- 返回值类型：LONG、DECIMAL或DOUBLE。
- 示例：

```
select floor(4.5);
+-----+
| floor(4.5) |
+-----+
|    4.0    |
```

12.5.15. EXP

EXP用于返回以e为底、x为幂的值。

```
exp(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select exp(4.5);
+-----+
| exp(4.5) |
+-----+
| 90.01713130052181 |
```

12.5.16. COS

COS用于返回x的余数值。

```
cos(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select cos(1.3);
+-----+
| cos(1.3) |
+-----+
| 0.26749882862458735 |
```

12.5.17. ACOS

ACOS用于返回x的反余弦值。

```
acos(double x)
```

- 命令说明：如果 $x > 1$ 或者 $x < -1$ ，返回结果为null。
- 返回值类型：DOUBLE。
- 示例：

```
select acos(0.5);
+-----+
| acos(0.5) |
+-----+
| 1.0471975511965979 |
```

12.5.18. TAN

TAN用于返回x的正切值。

```
tan(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select tan(8);
+-----+
| tan(8) |
+-----+
| -6.799711455220379 |
```

12.5.19. ATAN

ATAN用于返回x的反正切值。

```
atan(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select atan(0.5);
+-----+
| atan(0.5) |
+-----+
| 0.4636476090008061 |
```

12.5.20. ATAN2

ATAN2用于返回参数x除以参数y之后的反正切值。

```
atan2(double x, double y)
atan(double x, double y)
```

- 返回值类型：DOUBLE。
- 示例：

```
select atan2(0.5,0.3);
+-----+
| atan2(0.5, 0.3) |
+-----+
| 1.0303768265243125 |
```

12.5.21. COT

COT用于返回x的余切值。

```
cot(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select cot(1.234);
+-----+
| cot(1.234) |
+-----+
| 0.35013639786701445 |
```

12.5.22. ASIN

ASIN用于返回x的反正弦值。

```
asin(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select asin(0.5);
+-----+
| asin(0.5) |
+-----+
| 0.5235987755982989 |
```

12.5.23. SIN

SIN用于返回x的正弦值。

```
sin(double x)
```

- 返回值类型：DOUBLE。
- 示例：

```
select sin(1.234);
+-----+
| sin(1.234) |
+-----+
| 0.9438182093746337 |
```

12.6. 算术运算符

使用加号（+）进行加法运算

- 命令说明：加法。
- 示例：

```
select 3+5;
+-----+
| _col0 |
+-----+
| 8 |
```

使用减号（-）进行减法运算

- 命令说明：减法。
- 示例：

```
select 3-5;
+-----+
| _col0 |
+-----+
| -2 |
```

使用乘号（*）进行乘法运算

- 命令说明：乘法。
- 示例：

```
select 3*pi();
+-----+
|_col0 |
+-----+
| 9.42477796076938 |
```

使用除号（/）进行除法运算

- 命令说明：除法。
- 示例：

```
select 3/pi();
+-----+
|_col0 |
+-----+
| 0.954929658551372 |
```

使用DIV进行除法运算

- 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 |
```

使用百分号（%）或MOD进行求余运算

- 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 |
```

使用负号（-）进行正负数运算

- 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 |
```

12.7. 位函数和操作符

BIT_COUNT

- 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 |
```

&

- 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 |
```

<< (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 |
```

12.8. GEO函数

AnalyticDB for MySQL支持的GEO函数包括ST_Point或Point、ST_AsText、ST_GeometryFromText或ST_GeomFromText、ST_Distance、ST_Distance_Sphere，本文介绍GEO函数的用法。

ST_Point或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(g)

- 命令说明：返回g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_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 |
```

```
SELECT ST_Distance_Sphere(point(1,1), point(2,2), 6370986);
+-----+
| ST_Distance_Sphere(point(1,1), point(2,2), 6370986) |
+-----+
|          157225.08654191086 |
```

12.9. JSON函数

本文介绍AnalyticDB for MySQL中JSON函数的用法。

JSON_EXTRACT

json_extract(json, jsonpath)

- 命令说明：从 json 中返回 jsonpath 指定的值。
- 返回值类型：JSON。
- 示例：

```
select json_extract('[10, 20, [30, 40]]', '$[1]');
+-----+
|          20 |
```

JSON_ARRAY_CONTAINS

json_array_contains(json, value)

- 命令说明：判断 json 中是否包含 value 指定的值。
- 返回值类型：BOOLEAN。
- 示例：

```
select json_array_contains('[1, 2, 3]', 2);
+-----+
|          1 |
```

JSON_SIZE

json_size(json, jsonpath)

- 命令说明：返回 json 的大小。
- 返回值类型：BIGINT。
- 示例：

```
select json_size('{ "x": { "a": 1, "b": 2 } }', '$.x') as result;
+-----+
|          2 |
```

```
select json_size('{\"x\":{\"a\":1,\"b\":2}}', '$.x.a') as result;
+-----+
|      0      |
```

JSON_ARRAY_LENGTH

```
json_array_length(json)
```

- 命令说明：返回 json 数组的长度。
- 返回值类型：BIGINT。
- 示例：

```
select json_array_length('[1,2,3]')
+-----+
|      3      |
```

JSON_KEYS

```
json_keys(json,jsonpath)
```

- 命令说明：获取 json 在指定路径下的所有键值。
- 返回值类型：JSON ARRAY。
- 示例：

```
select json_keys(cast('{\"a\":1,\"b\":{\"c\":30}}' as json),'$.b')
+-----+
|      [\"c\"]      |
```

12.10. 条件判断函数

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 |
```

12.11. 加密和压缩函数

AnalyticDB for MySQL支持以下功能为加密和压缩的函数。

- **MD5**：计算参数MD5的hash值。
- **SHA1**：计算字符串的SHA-1校验和。
- **SHA2**：计算SHA-2校验和。
- **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。
- **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。

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 |
```

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 |
```

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 |
```

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 aes_decrypt(aes_encrypt(CAST('中国' AS VARBINARY), '0123'), '0123');
+-----+
| aes_decrypt(aes_encrypt(CAST('中国' AS varbinary), '0123'), '0123') |
+-----+
| 中国 |
```