ALIBABA CLOUD

阿里云

大数据计算服务

开发

文档版本：20220708

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.数据类型 16
1.1. 数据类型版本说明 16
1.2. 1.0数据类型版本 17
1.3. 2.0数据类型版本 18
1.4. Hive兼容数据类型版本 21
1.5. 组件与数据类型的兼容 24
2.常用命令 26
2.1. 常用命令列表 26
2.2. 项目空间操作 27
2.3. SET操作 29
2.4. 安全操作 34
2.5. 用户和角色操作 34
2.6. 授权操作 37
2.7. 表操作 37
2.8. 分区和列操作 50
2.9. 实例操作 53
2.10. 资源操作 57
2.11. 函数操作 60
2.12. Tunnel操作 63
2.13. 时区配置操作 65
2.14. SHOW 65
2.15. 其它操作 72
3.开发参考：SQL 74
3.1. SQL概述 74
3.2. 与其他SQL语法的差异 74
3.3. SQL使用限制项 76

3.4. DDL语句 77
3.4.1. 表操作 77
3.4.2. 分区和列操作 91
3.4.3. 生命周期操作 100
3.4.4. 视图操作 101
3.4.5. 物化视图操作 103
3.5. DML操作 113
3.5.1. 插入或覆写数据（INSERT INTO | INSERT OVERWRITE） 113
3.5.2. 插入或覆写动态分区数据（DYNAMIC PARTITION） 116
3.5.3. 更新或删除数据（UPDATE | DELETE） 118
3.5.4. MERGE INTO 124
3.5.5. 多路输出（MULTI INSERT） 125
3.5.6. VALUES 127
3.6. DQL操作（SELECT） 129
3.6.1. SELECT语法 129
3.6.2. SELECT语序 141
3.6.3. 子查询（SUBQUERY） 142
3.6.4. 交集、并集和补集 150
3.6.5. JOIN 154
3.6.6. SEMI JOIN（半连接） 158
3.6.7. MAPJOIN HINT 159
3.6.8. DISTRIBUTED MAPJOIN 161
3.6.9. SKEWJOIN HINT 162
3.6.10. 动态过滤器（Dynamic Filter） 164
3.6.11. Lateral View 166
3.6.12. HAVING子句 168
3.6.13. GROUPING SETS 168
3.6.14. SELECT TRANSFORM 174

3.7. SQL增强操作 180
3.7.1. LOAD 180
3.7.2. UNLOAD 184
3.7.3. EXPLAIN 191
3.7.4. COMMON TABLE EXPRESSION（CTE） 195
3.7.5. CLONE TABLE 196
3.7.6. 参数化视图 198
3.8. SQL常见问题 200
3.8.1. DDL操作常见问题 200
3.8.2. DML操作常见问题 203
3.8.3. DQL操作常见问题 205
3.8.4. SQL其他常见问题 210
3.9. 内建函数 212
3.9.1. 内建函数概述 212
3.9.2. 与Hive、MySQL、Oracle内建函数对照表 213
3.9.3. 日期函数 221
3.9.4. 数学函数 255
3.9.5. 窗口函数 288
3.9.6. 聚合函数 317
3.9.7. 字符串函数 338
3.9.8. 复杂类型函数 367
3.9.9. 其他函数 393
3.9.10. 内建函数典型案例 410
3.9.10.1. ROUND函数精度问题案例 411
3.9.10.2. 实现GROUP_CONCAT函数能力案例 411
3.9.11. 内建函数常见错误码 413
3.9.12. 内建函数常见问题 414
3.10. MaxCompute UDF 417

3.10.1. MaxCompute UDF概述 417
3.10.2. UDF 419
3.10.2.1. UDF概述 419
3.10.2.2. Java UDF 420
3.10.2.3. Python UDF 425
3.10.2.3.1. Python 2 UDF 425
3.10.2.3.2. Python 3 UDF 428
3.10.2.4. 通过UDF访问VPC网络资源 431
3.10.2.5. Java UDF使用示例 432
3.10.2.5.1. 兼容Hive Java UDF示例 432
3.10.2.5.2. 复杂数据类型示例 433
3.10.2.5.3. JSON字符串获取示例 436
3.10.2.5.4. JSON字符串值判断示例 440
3.10.2.5.5. 不同类型数据转换JSON STRING类型示例 445
3.10.2.5.6. JSON字符串增加键值示例 446
3.10.2.5.7. 使用正则表达式替换字符串示例 448
3.10.2.5.8. 获取字符串（不含分隔符）Value示例 449
3.10.2.5.9. 获取字符串（含有分隔符）Value示例 451
3.10.2.5.10. 获取指定日期格式的时间示例 453
3.10.2.5.11. 取余函数示例 457
3.10.2.5.12. 获取URL中指定位置的字符示例 459
3.10.2.6. Python UDF使用示例 461
3.10.2.6.1. 复杂数据类型示例 461
3.10.2.6.2. 引用文件资源示例 463
3.10.2.6.3. 引用表资源示例 465
3.10.2.6.4. Python UDF使用第三方包 466
3.10.3. UDTF 470
3.10.3.1. UDTF概述 470

3.10.3.2. Java UDTF 472
3.10.3.3. Java UDTF使用示例 478
3.10.3.3.1. Java UDTF读取MaxCompute资源示例 478
3.10.3.4. Python UDTF 482
3.10.3.4.1. Python 2 UDTF 482
3.10.3.4.2. Python 3 UDTF 486
3.10.3.5. Python UDTF使用示例 488
3.10.3.5.1. Python 2 UDTF读取MaxCompute资源示例 488
3.10.3.5.2. Python 3 UDTF读取MaxCompute资源示例 490
3.10.4. UDAF 492
3.10.4.1. UDAF概述 492
3.10.4.2. Java UDAF 494
3.10.4.3. Python UDAF 503
3.10.4.3.1. Python 2 UDAF 503
3.10.4.3.2. Python 3 UDAF 506
3.10.5. 代码嵌入式UDF 509
3.10.6. SQL语言定义函数 511
3.10.7. 开源地理空间UDF 514
3.10.8. MaxCompute UDF常见问题 517
3.10.8.1. MaxCompute UDF（Java）常见问题 517
3.10.8.2. MaxCompute UDF（Python）常见问题 518
3.11. MaxCompute UDT 522
3.11.1. UDT概述 523
3.11.2. UDT示例 526
3.12. MaxCompute UDJ 528
3.13. 脚本模式SQL 533
3.14. 渐进式计算 536
3.15. 附录 538

3.15.1. 运算符 538
3.15.2. 转义字符 542
3.15.3. LIKE字符匹配 542
3.15.4. 正则表达式规范 542
3.15.5. 保留字与关键字 544
3.15.6. 数据类型映射表 545
3.15.7. 数据类型转换 546
3.15.8. UDAF和UDTF动态参数说明 549
4.MaxCompute SQLML 551
4.1. 概述 551
4.2. 快速入门 552
4.3. 计量计费 559
5.PyODPS 563
5.1. 快速入门 563
5.2. 安装指南及使用限制 564
5.3. 工具平台使用指南 565
5.3.1. 工具平台使用概述 565
5.3.2. 从平台到自行部署 565
5.3.3. 在DataWorks上使用PyODPS 565
5.4. 基本操作 567
5.4.1. 基本操作概述 567
5.4.2. 项目空间 567
5.4.3. 表 567
5.4.4. SQL 573
5.4.5. 任务实例 575
5.4.6. 资源 576
5.4.7. 函数 577
5.4.8. 集成SQLAlchemy 578

5.5. 配置选项 579
5.6. DataFrame 580
5.6.1. DataFrame概述 580
5.6.2. 快速入门 581
5.6.3. 创建DataFrame 586
5.6.4. Sequence 587
5.6.5. Collection 589
5.6.6. 执行 595
5.6.7. 列运算 598
5.6.8. 聚合操作 605
5.6.9. 排序、去重、采样、数据变换 609
5.6.10. 使用自定义函数及Python第三方库 613
5.6.11. MapReduce API 618
5.6.12. 数据合并 624
5.6.13. 窗口函数 625
5.6.14. 绘图 627
5.6.15. 调试指南 629
5.7. 交互体验增强 631
5.7.1. 命令行增强 631
5.7.2. IPython增强 632
5.7.3. Jupyter Notebook增强 633
5.8. API概述 635
5.9. 示例程序 636
5.9.1. PyODPS使用第三方包 636
5.9.2. PyODPS条件查询 638
5.9.3. PyODPS参数传递 643
5.9.4. PyODPS读取分区表数据 645
5.9.5. PyODPS查看一级分区 648

5.9.6. PyODPS的Sequence及执行操作 650
5.9.7. PyODPS的聚合操作 653
5.9.8. PyODPS的列运算 657
5.9.9. PyDOPS的排序 661
5.9.10. PyODPS的去重 663
5.9.11. PyODPS的采样 665
5.9.12. PyODPS的数据缩放 672
5.9.13. PyODPS的空值处理 675
5.10. PyODPS常见问题 678
6.MapReduce 686
6.1. 概述 686
6.1.1. MapReduce 686
6.1.2. 扩展MapReduce（MR2） 687
6.1.3. Hadoop兼容版本MapReduce 687
6.2. 使用限制 691
6.3. 快速入门 692
6.4. 使用说明 695
6.4.1. 基本概念 695
6.4.2. MapReduce作业提交 695
6.4.3. 输入与输出 696
6.4.4. 资源使用 697
6.4.5. 本地运行 697
6.5. 示例程序 698
6.5.1. WordCount示例 698
6.5.2. MapOnly示例 700
6.5.3. 多路输入输出示例 703
6.5.4. 多任务示例 706
6.5.5. 二次排序示例 707

6.5.6. 使用资源示例 709
6.5.7. 使用Counter示例 712
6.5.8. Grep示例 714
6.5.9. Join示例 716
6.5.10. Sleep示例 718
6.5.11. Unique示例 719
6.5.12. Sort示例 722
6.5.13. 分区表输入示例 723
6.5.14. Pipeline示例 724
6.6. Java SDK 726
6.6.1. 原生SDK概述 726
6.6.2. 兼容版本SDK概述 730
6.6.3. Java沙箱 738
6.7. MapReduce常见问题 741
7.数据科学（Mars） 748
7.1. 版本更新记录 748
7.2. 概述 749
7.3. 准备工作 751
7.4. 使用说明 752
8.Spark 754
8.1. 概述 754
8.2. 搭建开发环境 754
8.3. 运行模式 766
8.4. 开发示例 772
8.4.1. 概述 772
8.4.2. Spark-1.x示例 773
8.4.3. Spark-2.x示例 775
8.5. PySpark开发示例 783

8.6. Spark访问VPC实例 792
8.7. Spark访问OSS 797
8.8. 作业诊断 798
8.9. Spark常见问题 799
9.查询加速（MCQA） 804
9.1. 概述 804
9.2. 包年包月规格下使用查询加速（邀测） 805
9.3. 增强：查询结果缓存机制 806
9.4. 免费体验 807
9.5. 使用说明 807
10.外部表 815
10.1. 概述 815
10.2. STS模式授权 815
10.3. 网络开通流程 817
10.4. OSS外部表 824
10.4.1. 创建OSS外部表 824
10.4.2. 读取OSS数据 834
10.4.3. 读取以分区方式存储的OSS数据 836
10.4.4. 将数据写入OSS 837
10.5. OTS外部表 842
10.6. Hologres外部表 845
10.7. RDS外部表 855
10.8. HBase外部表（HBase标准版或增强版） 857
10.9. Lindorm外部表（云原生多模数据库） 860
10.10. 外部表常见问题 863
11.交互式分析（Lightning） 866
11.1. MaxCompute Lightning服务下线公告 866
11.2. MaxCompute Lightning升级为实时数仓Hologres 866

11.3. 概述 866
11.4. 访问域名（Endpoint） 867
11.5. 使用限制 868
11.6. 快速入门 868
11.7. 使用说明 871
11.7.1. 通过JDBC连接MaxCompute服务 871
11.7.1.1. JDBC驱动程序 871
11.7.1.2. 配置JDBC连接 872
11.7.1.3. 通过其他常见工具连接MaxCompute Lightning服务 873
11.7.2. 查询语法与内建函数 881
11.7.3. 查看或取消查询作业 881
12.Graph 883
12.1. 概述 883
12.2. Aggregator机制 885
12.3. 使用限制 889
12.4. 使用说明 890
12.5. 编写Graph 892
12.6. SDK配置说明 893
12.7. 开发和调试 894
12.8. 示例程序 897
12.8.1. 单源最短距离 897
12.8.2. PageRank 903
12.8.3. K-均值聚类 904
12.8.4. BiPartiteMatching 908
12.8.5. 强连通分量 910
12.8.6. 连通分量 915
12.8.7. 拓扑排序 917
12.8.8. 线性回归 919

12.8.9. 三角形计数 922
12.8.10. 输入点表 924
12.8.11. 输入边表 928
13.查看作业运行信息 933
13.1. 使用Logview查看作业运行信息 933
13.2. 使用Logview 2.0查看Job运行信息 936
13.3. 使用MaxCompute编译器检查脚本 942
14.通过DataWorks开发调度节点 944
14.1. 创建ODPS SQL节点 944
14.2. 创建SQL组件节点 947
14.3. 创建ODPS Spark节点 949
14.4. 创建PyODPS 2节点 952
14.5. 创建PyODPS 3节点 954
14.6. 创建ODPS Script节点 955
14.7. 创建ODPS MR节点 956
14.8. 创建MaxCompute表 958
14.9. 创建MaxCompute资源 960
14.10. 注册MaxCompute函数 962
15.优化器信息收集 964
16.MaxCompute开源支持 969

1.数据类型

1.1. 数据类型版本说明

MaxCompute 2.0推出了兼容开源主流产品的2.0数据类型和Hive兼容数据类型两个数据类型版本。加上原有的1.0数据类型版本，目前MaxCompute一共支持3个数据类型版本。

MaxCompute数据类型版本

MaxCompute设置数据类型版本属性的参数共有3个：

- odps.sql.type.system.odps2：MaxCompute 2.0数据类型版本的开关，属性值为True或False。
- odps.sql.decimal.odps2：MaxCompute 2.0的Decimal数据类型的开关，属性值为True或False。
- odps.sql.hive.compatible：MaxCompute Hive兼容模式（即部分数据类型和SQL行为兼容Hive）数据类型版本的开关，属性值为True或False。

在新增项目时MaxCompute可以对3个版本的数据类型进行选择，各个版本默认设置如下：

- 1.0数据类型版本

  ```
  setproject odps.sql.type.system.odps2=false; --关闭MaxCompute 2.0数据类型。
  setproject odps.sql.decimal.odps2=false; --关闭Decimal 2.0数据类型。
  setproject odps.sql.hive.compatible=false; --关闭Hive兼容模式。
  ```

  适用于早期使用的MaxCompute项目，且该项目依赖的产品组件不支持2.0数据类型版本。

- 2.0数据类型版本

  ```
  setproject odps.sql.type.system.odps2=true; --打开MaxCompute 2.0数据类型。
  setproject odps.sql.decimal.odps2=true; --打开Decimal 2.0数据类型。
  setproject odps.sql.hive.compatible=false; --关闭Hive兼容模式。
  ```

  适用于在2020年04月之前无存量数据的MaxCompute项目，且该项目依赖的产品组件支持2.0数据类型版本。

- Hive兼容数据类型

  ```
  setproject odps.sql.type.system.odps2=true; --打开MaxCompute 2.0数据类型。
  setproject odps.sql.decimal.odps2=true; --打开Decimal 2.0数据类型。
  setproject odps.sql.hive.compatible=true; --打开Hive兼容模式。
  ```

  适用于从Hadoop迁移的MaxCompute项目，且该项目依赖的产品组件支持2.0数据类型版本。

> **说明** 阿里云其他产品或组件对数据类型的支持情况，请参见组件与数据类型的兼容。

选择数据类型版本

项目选择的数据类型版本影响范围如下：

- 表的数据类型。
- DML语句、内置函数的行为。
- 与MaxCompute密切相关的开发组件。

因此，在选择项目的数据类型版本前建议您了解各种版本的说明以及差异点，详情请参见与其他数据类型版本的差异。

查看项目的数据类型版本

您可以在MaxCompute客户端、使用控制台（查询编辑器）连接、MaxCompute Studio或DataWorks上执行如下命令，查看已经创建项目的属性。

```
setproject;
```

查看odps.sql.type.system.odps2、odps.sql.decimal.odps2、odps.sql.hive.compatible属性判断当前项目的数据类型版本。

修改项目的数据类型版本

如果发现项目当前选择的数据类型版本不能满足业务需求，可以修改数据类型版本。

项目管理员（Project Owner）或超级管理员角色（Super_Administrator Role）可以在MaxCompute客户端、使用控制台（查询编辑器）连接、MaxCompute Studio或DataWorks上执行如下命令，修改项目的数据类型版本。

```
--查看项目数据类型版本。
setproject;
--开启/关闭MaxCompute2.0数据类型版本。
setproject odps.sql.type.system.odps2=true/false;
--开启/关闭decimal2.0数据类型。
setproject odps.sql.decimal.odps2=true/false;
--开启/关闭hive兼容模式数据类型版本。
setproject odps.sql.hive.compatible=true/false;
```

修改建议如下：

- 项目当前使用2.0数据类型版本，如果发现必须使用的组件不支持2.0数据类型。您可以使用如下方式解决与不支持2.0数据类型组件的对接问题：
  - 将项目修改为1.0数据类型版本。
  - 在与不支持2.0数据类型组件对接任务时，将Session设置为1.0数据类型版本。

    > **说明** 将如下语句与Session中的命令一起提交，即可将该Session设置为1.0数据类型版本。该命令仅支持小写。
    >
    > ```
    > set odps.sql.type.system.odps2=false;
    > ```

- 项目使用2.0数据类型版本，部分SQL必须使用1.0数据类型，但是部分功能依赖2.0数据类型（例如current_timestamp函数）。解决方法如下：
  - 将SQL拆分为单独的Session，将Session设置为1.0数据类型版本。
  - 将SQL改写为可以替代的功能。
- 如果项目最开始使用了2.0数据类型版本，后面变更成1.0数据类型版本。需要注意以下2点：
  - 对于已经使用了2.0数据类型的表，读取数据时，TINYINT、SMALLINT、INT转为BIGINT类型，CHAR、VARCHAR转为STRING类型。
  - 建立1.0数据类型的表，将2.0数据类型的表中数据通过CAST函数导入1.0数据类型的表，即可将2.0数据类型的数据转化为1.0数据类型。

1.2. 1.0数据类型版本

MaxCompute 1.0数据类型版本是MaxCompute三种数据类型版本之一，该数据类型版本仅支持1.0数据类型。本文为您介绍1.0数据类型版本的设置方法、支持的数据类型以及与其它数据类型版本的差异。

定义

项目空间选择数据类型版本为1.0数据类型版本时，项目的数据类型属性定义如下。

```
setproject odps.sql.type.system.odps2=false; --关闭MaxCompute 2.0数据类型。
setproject odps.sql.decimal.odps2=false; --关闭Decimal 2.0数据类型。
setproject odps.sql.hive.compatible=false; --关闭Hive兼容模式。
```

适用场景

适用于早期的MaxCompute项目，且该项目依赖的产品组件不支持2.0数据类型版本。

数据类型

类型	常量示例	描述
BIGINT	100000000000L、-1L	64位有符号整型。 取值范围：$-2^{63}+1$~$2^{63}-1$。
DOUBLE	3.14159261E+7	64位二进制浮点型。
DECIMAL	3.5BD、99999999999.9999999BD	10进制精确数字类型。 整型部分取值范围：$-10^{36}+1$~$10^{36}-1$，小数部分精确到10^{-18}。
STRING	"abc"、'bcd'、"alibaba"、'inc'	字符串类型。 长度限制为8 MB。
DATETIME	DATETIME'2017-11-11 00:00:00'	日期时间类型。 取值范围：0000年1月1日~9999年12月31日。
BOOLEAN	True、False	BOOLEAN类型。 取值范围：True或False。

数据类型说明如下：

- 上述数据类型均可以为NULL。
- 整型常量的语义默认为BIGINT类型。如果常量超过了BIGINT的值域（例如1,000,000,000,000,000,000,000,000,000），会被作为DOUBLE类型处理。例如 `SELECT 1 + a;` 中的整型常量1会被作为BIGINT类型处理。
- 如果参数涉及2.0数据类型的内置函数，无法在1.0数据类型版本正常使用。
- 分区表的分区列的数据类型只支持STRING类型。
- 支持连接STRING常量。例如，abc和xyz会解析为abcxyz。

- 向DECIMAL字段插入常量时，常量的写法需要与常量定义中的格式保持一致。例如示例代码中的 `3.5BD` 。

```
INSERT INTO test_tb(a) VALUES (3.5BD);
```

- DATETIME查询显示的时间值不包含毫秒。Tunnel通过 `-dfp` 指定时间格式，可以指定显示到毫秒，例如 `tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'` 。Tunnel详情请参见Tunnel命令参考。

与其它数据类型版本差异说明

- 1.0数据类型版本的LIMIT 、ORDER BY、DISTRIBUTE BY、SORT BY和CLUSTER BY操作与2.0数据类型版本不同。

  例如，LIMIT操作语句 `SELECT * FROM t1 UNION ALL SELECT * FROM t2 LIMIT 10;` ：

  - 1.0数据类型版本： `SELECT * FROM t1 UNION ALL SELECT * FROM ( SELECT * FROM t2 LIMIT 10) t2;` 。
  - 2.0数据类型版本： `SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2 ) t LIMIT 10;` 。

- 1.0数据类型版本的IN表达式与2.0数据类型版本不同。

  例如 `a IN (1, 2, 3)` ：

  - 1.0数据类型版本：IN括号里的所有值类型必须一致。
  - 2.0数据类型版本：IN括号里的所有值支持通过隐式类型转换为一致类型。

复杂数据类型

类型	定义方法	构造方法
ARRAY	• `ARRAY<BIGINT>` • `ARRAY<STRUCT<a:BIGINT, b:STRING>>`	• `ARRAY(1, 2, 3)` • `ARRAY(NAMED_STRUCT('a', 1, 'b', '2'), NAMED_STRUCT('a', 3, 'b', '4'))`
MAP	• `MAP<STRING, STRING>` • `MAP<BIGINT, ARRAY<STRING>>`	• `MAP("k1", "v1", "k2", "v2")` • `MAP(1L, ARRAY('a', 'b'), 2L, ARRAY('x', 'y'))`
STRUCT	• `STRUCT<'x', BIGINT, 'y', BIGINT>` • `STRUCT<'field1', BIGINT, 'field2', ARRAY<BIGINT>, 'field3', MAP<BIGINT>>`	• `NAMED_STRUCT('x', 1, 'y', 2)` • `NAMED_STRUCT('field1', 100L, 'field2', ARRAY(1, 2), 'field3', MAP(1, 100, 2, 200))`

> 说明 MaxCompute的复杂数据类型可以被嵌套使用，相关的内建函数说明请参见ARRAY、MAP或STRUCT。

1.3. 2.0数据类型版本

MaxCompute 2.0数据类型版本是MaxCompute三种数据类型版本之一，该数据类型版本下仅支持2.0数据类型。本文为您介绍2.0数据类型版本的定义、支持的数据类型以及与其它数据类型版本的差异。

定义

项目空间选择数据类型版本为2.0数据类型版本时，项目空间的数据类型属性参数定义如下。

```
setproject odps.sql.type.system.odps2=true; --打开MaxCompute 2.0数据类型。
setproject odps.sql.decimal.odps2=true; --打开Decimal 2.0数据类型。
setproject odps.sql.hive.compatible=false; --关闭Hive兼容模式。
```

适用场景

适用于在2020年04月之前无存量数据的MaxCompute项目，且该项目依赖的产品组件支持2.0数据类型版本。

基础数据类型

类型	常量示例	描述
TINYINT	1Y、-127Y	8位有符号整型。 取值范围：-128~127。
SMALLINT	32767S、-100S	16位有符号整型。 取值范围：-32768~32767。

类型	常量示例	描述
INT	1000、-15645787	32位有符号整型。 取值范围：-2^{31}~2^{31}-1。
BIGINT	100000000000L、-1L	64位有符号整型。 取值范围：$-2^{63}+1$~2^{63}-1。
BINARY	unhex('FA34E10293CB42848573A4E39937F479')	二进制数据类型，目前长度限制为8 MB。
FLOAT	cast(3.14159261E+7 as float)	32位二进制浮点型。
DOUBLE	3.14159261E+7	64位二进制浮点型。
DECIMAL(precision,scale)	3.5BD、99999999999.9999999BD	10进制精确数字类型。 • precision：表示最多可以表示多少位的数字。取值范围：`1 <= precision <= 38`。 • scale：表示小数部分的位数。取值范围：`0 <= scale <= 18`。 如果不指定以上两个参数，则默认为`decimal(38,18)`。 **说明** • 同一个表中不能同时存在新老Decimal数据类型。 • 执行`setproject odps.sql.hive.compatible=true;`命令开启Hive兼容模式时，`Decimal(precision, scale)`类型在数据导入（Tunnel Upload）和SQL运算过程中，如果数据的小数位数超过scale的大小，则会进行四舍五入的处理。如果整数部分超出限制，则会报错。
VARCHAR(n)	无	变长字符类型，n为长度。 取值范围：1~65535。
CHAR(n)	无	固定长度字符类型，n为长度。最大取值255。长度不足则会填充空格，但空格不参与比较。
STRING	"abc"、'bcd'、"alibaba"、'inc'	字符串类型，目前长度限制为8 MB。
DATE	DATE'2017-11-11'	日期类型，格式为`yyyy-mm-dd`。 取值范围：0000-01-01~9999-12-31。
DATETIME	DATETIME'2017-11-11 00:00:00'	日期时间类型。 取值范围：0000-01-01 00:00:00.000~9999-12-31 23:59:59.999，精确到毫秒。
TIMESTAMP	TIMESTAMP'2017-11-11 00:00:00.123456789'	时间戳类型。 取值范围：0000-01-01 00:00:00.000000000~9999-12-31 23:59:59.999999999，精确到纳秒。 **说明** TIMESTAMP类型本身与时区无关，在任何时区，TIMESTAMP类型存储的都是从Epoch（UTC 1970-01-01 00:00:00）开始的一个偏移量。您可以通过内建函数对TIMESTAMP类型数据执行与时区相关的计算，如`cast(<a timestamp> as string)`将TIMESTAMP类型数据按照当前时区值转换成STRING类型。
BOOLEAN	True、False	BOOLEAN类型。 取值范围：True、False。

数据类型说明如下：

- 上述的各种数据类型均可以为NULL。
- SQL中的INT关键字是32位整型。

```
--将a转换为32位整型。
cast(a as INT)
```

- 整型常量的语义会默认为INT类型。例如 `SELECT 1 + a;` 中的整型常量1会被作为INT类型处理。如果常量过长，超过了INT的值域而又没有超过BIGINT的值域，则会作为BIGINT类型处理；如果超过了BIGINT的值域，则会被作为DOUBLE类型处理。
- 隐式转换
  - 部分隐式类型转换会被禁用。例如，STRING->BIGINT、STRING->DATETIME、DOUBLE->BIGINT、DECIMAL->DOUBLE、DECIMAL->BIGINT有精度损失或者报错的风险。禁用类型可以通过CAST函数强制进行数据类型转换。
  - VARCHAR类型常量可以通过隐式转换为STRING常量。
- 表、函数以及UDF
  - 参数涉及2.0数据类型的内置函数，在2.0数据类型版本下可以正常使用。
  - UDF包含的数据类型都会按照2.0数据类型进行解析重载。
  - 分区列支持STRING、VARCHAR、CHAR、TINYINT、SMALLINT、INT、BIGINT数据类型。
- STRING常量支持连接，例如，abc和xyz会解析为abcxyz。
- 给DECIMAL字段插入常量时，常量的写法需要与常量定义中的格式保持一致。例如下面示例代码中的 `3.5BD` 。

```
insert into test_tb(a) values (3.5BD)
```

- DATETIME查询显示的时间值不包含毫秒。Tunnel命令通过 `-dfp` 来指定时间格式，可以指定到毫秒显示，例如 `tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'` 。关于Tunnel命令的更多信息，请参见Tunnel命令。

复杂数据类型

类型	定义方法	构造方法
ARRAY	• `array<int>` • `array<struct<a:int, b:string>>`	• `array(1, 2, 3)` • `array(array(1, 2), array(3, 4))`
MAP	• `map<string, string>` • `map<smallint, array<string>>`	• `map("k1", "v1", "k2", "v2")` • `map(1S, array('a', 'b'), 2S, array('x', 'y'))`
STRUCT	• `struct<x:int, y:int>` • `struct<field1:bigint, field2:array<int>, field3:map<int, int>>`	• `named_struct('x', 1, 'y', 2)` • `named_struct('field1', 100L, 'field2', array(1, 2), 'field3', map(1, 100, 2, 200))`

> 说明
> - MaxCompute的复杂数据类型可以被嵌套使用，相关的内建函数说明请参见ARRAY、MAP或STRUCT。
> - MaxCompute的复杂数据类型最大Size建议不要超过1 MB，否则会造成计算过程OOM的错误。

与其他数据类型版本的差异

- DML语法行为差异
  - 集合操作中的LIMIT语句行为不同。

    例如，对于 `SELECT * FROM t1 UNION ALL SELECT * FROM t2 limit 10;` ：
    - 1.0数据类型版本下时为 `SELECT * FROM t1 UNION ALL SELECT * FROM ( SELECT * FROM t2 limit 10) t2;` 。
    - 2.0数据类型版本下时为 `SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2 ) t limit 10;` 。

    ORDER BY、DISTRIBUTE BY、SORT BY和CLUSTER BY操作也具有上述行为特点。
  - IN表达式的类型解析不同。

    例如 `a in (1, 2, 3)` ：
    - 1.0数据类型版本时，要求IN后面的括号里的所有值的类型必须一致。
    - 2.0数据类型版本时，IN后面的括号里的所有值支持隐式类型转换一致即可。

- INSERT语句转换规则不同。
  - Hive兼容数据类型：源数据类型如果可以显式转换为表中的数据类型，系统会自动插入转换函数并允许运行。
  - 1.0和2.0数据类型版本：源数据类型需要隐式转换为表中的数据类型，否则报错。

```
--在Hive模式下成功，在其他模式下报错。
create table t (a bigint);
insert into table select 1.5;
```

- 函数行为差异
  - `+`、`-`、`*`、`/`、POW函数
    - Hive兼容数据类型版本：在数据溢出时绕回（即数据超出范围后返回初始值）。
    - 1.0和2.0数据类型版本：在数据溢出时报告异常，其它模式下返回NULL。
  - `>`、`>=`、`=`、`<`、`<=`
    - Hive兼容数据类型版本：在比较DOUBLE类型数据时，直接比较Double值。
    - 1.0和2.0数据类型版本：在比较DOUBLE类型数据时，如果小数点后15位以前数据相同，认为数据相同。15位以后的数据不作比较。
  - 位操作运算符`&`、`|`、`^`
    - Hive兼容数据类型版本：返回参数类型。
    - 1.0和2.0数据类型版本：返回类型为BIGINT。
  - LENGTH、LENGTHB、FIND_IN_SET、INSTR、SIZE、HASH、SIGN函数
    - Hive兼容数据类型版本：返回类型为INT。
    - 1.0和2.0数据类型版本：返回类型为BIGINT。
  - FLOOR、CEIL
    - Hive兼容数据类型版本：如果参数为DECIMAL类型，在Hive兼容数据类型版本下返回类型仍为DECIMAL。
    - 1.0和2.0数据类型版本：如果参数为DECIMAL类型，在1.0和2.0数据类型版本下返回类型为BIGINT。
  - ROUND
    - Hive兼容数据类型版本：如果参数为TINYINT、SMALLINT、INT、BIGINT时，返回值类型与参数类型相同。
    - 1.0和2.0数据类型版本：如果参数为TINYINT、SMALLINT、INT、BIGINT时，返回值为DATETIME类型。
  - FROM_UNIXTIME
    - Hive兼容数据类型版本：返回值为STRING类型。
    - 1.0和2.0数据类型版本：返回值为DATETIME类型。
  - CONCAT_WS
    - Hive兼容数据类型版本：如果一个被连接的输入字符串为NULL，此字符串被忽略。
    - 1.0和2.0数据类型版本：如果一个被连接的输入字符串为NULL，则返回NULL。
  - FIND_IN_SET
    - Hive兼容数据类型版本：空字符串认为匹配字符串尾部。

```
--Hive兼容模式下
find_in_set("","")  结果是1。
find_in_set("", "a,") 结果是2。
```

    - 1.0和2.0数据类型版本：空字符串认为不匹配返回0。
  - REGEXP_(EXTRACT/REPLACE)
    - Hive兼容数据类型版本：REGEXP的模式定义符合Java regex规范。
    - 1.0和2.0数据类型版本：REGEXP的模式定义符合MaxCompute规范。
  - SUBSTR

```
string substr(string <str>, bigint <start_position>[, bigint <length>])
```

    start_position：必填，为BIGINT类型，默认起始位置为1。
    - Hive兼容数据类型版本：当start_position为0时，与起始位置为1时相同。
    - 1.0和2.0数据类型版本：当start_position为0时，返回NULL。

1.4. Hive兼容数据类型版本

Hive兼容数据类型版本是MaxCompute三种数据类型版本之一，该数据类型版本下仅支持Hive兼容数据类型。本文为您介绍Hive兼容数据类型版本的定义、支持的数据类型以及与其他数据类型版本的差异。

定义

项目空间选择数据类型版本为Hive兼容数据类型版本时，项目空间的数据类型属性参数定义如下。

```
setproject odps.sql.type.system.odps2=true; --打开MaxCompute 2.0数据类型。
setproject odps.sql.decimal.odps2=true; --打开Decimal 2.0数据类型。
setproject odps.sql.hive.compatible=true; --打开Hive兼容模式。
```

适用场景

适用于从Hadoop迁移的MaxCompute项目，且该项目依赖的产品组件支持2.0数据类型版本。

基础数据类型

Hive兼容数据类型版本支持的基础数据类型与2.0数据类型定义基本一致，只有DECIMAL数据类型在两个版本下有些差异。

类型	常量示例	描述
TINYINT	1Y、-127Y	8位有符号整型。 取值范围：-128~127。
SMALLINT	32767S、-100S	16位有符号整型。 取值范围：-32768~32767。
INT	1000、-15645787	32位有符号整型。 取值范围：-2^{31}~$2^{31}-1$。
BIGINT	100000000000L、-1L	64位有符号整型。 取值范围：$-2^{63}+1$~$2^{63}-1$。
BINARY	unhex('FA34E10293CB42848573A4E39937F479')	二进制数据类型，目前长度限制为8MB。
FLOAT	cast(3.14159261E+7 as float)	32位二进制浮点型。
DOUBLE	3.14159261E+7	64位二进制浮点型。
DECIMAL(precision,scale)	3.5BD、 99999999999.9999999BD	10进制精确数字类型。 • precision：表示最多可以表示多少位的数字。取值范围：`1 <= precision <= 38`。 • scale：表示小数部分的位数。取值范围：`0 <= scale <= 38`。 如果不指定以上两个参数，则默认为`decimal(10,0)`。
VARCHAR(n)	无	变长字符类型，n为长度。 取值范围：1~65535。
CHAR(n)	无	固定长度字符类型，n为长度。最大取值255。长度不足则会填充空格，但空格不参与比较。
STRING	"abc"、'bcd'、"alibaba"、'inc'	字符串类型，长度限制为8 MB。
DATE	DATE'2017-11-11'	日期类型，格式为`yyyy-mm-dd`。 取值范围：0000-01-01~9999-12-31。
DATETIME	DATETIME'2017-11-11 00:00:00'	日期时间类型。 取值范围：0000-01-01 00:00:00.000~9999-12-31 23:59:59.999，精确到毫秒。
TIMESTAMP	TIMESTAMP'2017-11-11 00:00:00.123456789'	与时区无关的时间戳类型。 取值范围：0000-01-01 00:00:00.000000000~9999-12-31 23:59:59.999999999，精确到纳秒。 **说明** 对于部分时区相关的函数，例如`cast(<a timestamp> as string)`，要求TIMESTAMP按照与当前时区相符的方式来展现。

类型	常量示例	描述
BOOLEAN	True、False	BOOLEAN类型。 取值范围：True、False。

数据类型说明如下：

- 上述的各种数据类型均可以为NULL。
- SQL中的INT关键字是32位整型。

```
--将a转换为32位整型。
cast(a as INT)
```

- 整型常量的语义会默认为INT类型。例如 `SELECT 1 + a;` 中的整型常量1会被作为INT类型处理。如果常量过长，超过了INT的值域而又没有超过BIGINT的值域，则会作为BIGINT类型处理；如果超过了BIGINT的值域，则会被作为DOUBLE类型处理。
- 隐式转换
  - 部分隐式类型转换会被禁用。例如，STRING->BIGINT、STRING->DATETIME、DOUBLE->BIGINT、DECIMAL->DOUBLE、DECIMAL->BIGINT有精度损失或者报错的风险。禁用类型可以通过CAST函数强制进行数据类型转换。
  - VARCHAR类型常量可以通过隐式转换为STRING常量。
- 表、函数以及UDF
  - 参数涉及2.0数据类型的内置函数，可以正常使用。
  - UDF包含的数据类型都会按照Hive兼容数据类型进行解析重载。
  - 分区列支持STRING、VARCHAR、CHAR、TINYINT、SMALLINT、INT、BIGINT数据类型。
  - Hive模式下，部分函数不支持分区裁剪，详情请参见与Hive、MySQL、Oracle内建函数对照表。
- STRING常量支持连接。例如，abc和xyz会解析为abcxyz。
- 给DECIMAL字段插入常量时，常量的写法需要与常量定义中的格式保持一致。例如下面示例代码中的 `3.5BD` 。

```
insert into test_tb(a) values (3.5BD)
```

- DATETIME查询显示的时间值不包含毫秒。Tunnel命令通过 `-dfp` 来指定时间格式，可以指定到毫秒显示，例如 `tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'` 。关于Tunnel命令的更多信息，请参见Tunnel命令。

复杂数据类型

类型	定义方法	构造方法
ARRAY	• `array<int>` • `array<struct<a:int, b:string>>`	• `array(1, 2, 3)` • `array(array(1, 2), array(3, 4))`
MAP	• `map<string, string>` • `map<smallint, array<string>>`	• `map("k1", "v1", "k2", "v2")` • `map(1S, array('a', 'b'), 2S, array('x', 'y'))`
STRUCT	• `struct<x:int, y:int>` • `struct<field1:bigint, field2:array<int>, field3:map<int, int>>`	• `named_struct('x', 1, 'y', 2)` • `named_struct('field1', 100L, 'field2', array(1, 2), 'field3', map(1, 100, 2, 200))`

> **说明** MaxCompute的复杂数据类型可以被嵌套使用，相关的内建函数说明请参见ARRAY、MAP或STRUCT。

与其他数据类型版本的差异

- INSERT语句转换规则不同。
  - Hive兼容数据类型：源数据类型如果可以显式转换为表中的数据类型，系统会自动插入转换函数并允许运行。
  - 1.0和2.0数据类型版本：源数据类型需要隐式转换为表中的数据类型，否则报错。

```
--在Hive模式下成功，在其他模式下报错。
create table t (a bigint);
insert into table select 1.5;
```

- 函数行为差异
  - `+`、`-`、`*`、`/`、POW函数
    - Hive兼容数据类型版本：在数据溢出时绕回（即数据超出范围后返回初始值）。
    - 1.0和2.0数据类型版本：在数据溢出时报告异常，其它模式下返回NULL。

- `>` 、`>=` 、`=` 、`<` 、`<=`
  - Hive兼容数据类型版本：在比较DOUBLE类型数据时，直接比较Double值。
  - 1.0和2.0数据类型版本：在比较DOUBLE类型数据时，如果小数点后15位以前数据相同，认为数据相同。15位以后的数据不作比较。
- 位操作运算符 `&` 、`|` 、`^`
  - Hive兼容数据类型版本：返回参数类型。
  - 1.0和2.0数据类型版本：返回类型为BIGINT。
- LENGTH、LENGTHB、FIND_IN_SET、INSTR、SIZE、HASH、SIGN函数
  - Hive兼容数据类型版本：返回类型为INT。
  - 1.0和2.0数据类型版本：返回类型为BIGINT。
- FLOOR、CEIL
  - Hive兼容数据类型版本：如果参数为DECIMAL类型，在Hive兼容数据类型版本下返回类型仍为DECIMAL。
  - 1.0和2.0数据类型版本：如果参数为DECIMAL类型，在1.0和2.0数据类型版本下返回类型为BIGINT。
- ROUND
  - Hive兼容数据类型版本：如果参数为TINYINT、SMALLINT、INT、BIGINT时，返回值类型与参数类型相同。
  - 1.0和2.0数据类型版本：如果参数为TINYINT、SMALLINT、INT、BIGINT时，返回值为DATETIME类型。
- FROM_UNIXTIME
  - Hive兼容数据类型版本：返回值为STRING类型。
  - 1.0和2.0数据类型版本：返回值为DATETIME类型。
- CONCAT_WS
  - Hive兼容数据类型版本：如果一个被连接的输入字符串为NULL，此字符串被忽略。
  - 1.0和2.0数据类型版本：如果一个被连接的输入字符串为NULL，则返回NULL。
- FIND_IN_SET
  - Hive兼容数据类型版本：空字符串认为匹配字符串尾部。

    ```
    --Hive兼容模式下
    find_in_set("","")  结果是1。
    find_in_set("", "a,") 结果是2。
    ```

  - 1.0和2.0数据类型版本：空字符串认为不匹配返回0。
- REGEXP_(EXTRACT/REPLACE)
  - Hive兼容数据类型版本：REGEXP的模式定义符合Java regex规范。
  - 1.0和2.0数据类型版本：REGEXP的模式定义符合MaxCompute规范。
- SUBSTR

  ```
  string substr(string <str>, bigint <start_position>[, bigint <length>])
  ```

  start_position：必填，为BIGINT类型，默认起始位置为1。
  - Hive兼容数据类型版本：当start_position为0时，与起始位置为1时相同。
  - 1.0和2.0数据类型版本：当start_position为0时，返回NULL。

1.5. 组件与数据类型的兼容

MaxCompute可以与多款阿里云产品组件集成使用，不同组件支持不同的数据类型。部分组件仅支持MaxCompute 1.0数据类型。

组件	1.0数据类型	2.0数据类型	复杂类型	Hive兼容模式	Decimal2.0
Storage	支持	支持	支持	支持	支持
SDK	支持	支持	支持	支持	支持
SQL	支持	支持	支持	支持	支持
MapReduce	支持	不支持	不支持	不支持	不支持
OpenMapReduce	支持	不支持	不支持	不支持	不支持
PAI	支持除Decimal之外的其它数据类型	2.0新增的数据类型中仅支持INT	部分功能支持map数据类型	不支持	不支持

组件	1.0数据类型	2.0数据类型	复杂类型	Hive兼容模式	Decimal2.0
PyODPS	支持	• DataFrame不支持2.0数据类型 • 其它功能都支持2.0数据类型	支持	• DataFrame不支持2.0数据类型 • 其它功能不支持2.0数据类型中的DATE	不支持
JDBC	支持	不支持	不支持	不支持	不支持
Graph	支持	支持	支持	支持	不支持
Tunnel	支持	支持	支持	支持	支持
DataWorks（数据集成）	支持	支持	支持	支持	支持
DataX	支持	不支持	不支持	不支持	不支持
DataWorks（数据质量）	支持	支持	支持	支持	支持
DataHub	支持	不支持	不支持	不支持	不支持
Lightning	支持	不支持	不支持	不支持	不支持
DataWorks（数据地图/元数据）	支持	支持	支持	支持	支持
DataWorks（IDE）	支持	支持	支持	支持	支持
DataWorks（WebExcel）	支持	不支持	不支持	不支持	不支持
DataWorks（安全中心）	支持	支持	支持	支持	支持
DataWorks（保护伞）	支持	支持	支持	支持	支持

2.常用命令
2.1. 常用命令列表

本文为您介绍MaxCompute SQL常用开发命令。

MaxCompute提供了对项目空间、表、资源及实例等对象的一系列操作。您可以通过各工具平台或SDK来操作这些对象。

> 说明 对于SDK的更多介绍，请参见MaxCompute SDK介绍。

常用命令

MaxCompute常用命令如下。

类别	常用命令
项目空间操作	• 进入项目空间（use） • 查看项目空间属性（setproject） • 设置项目空间属性（setproject） • 查看项目空间账号体系（list accountproviders） • 增加RAM账号体系（add） • 删除RAM账号体系（remove）
set操作	• set • show flags
安全操作	• 开启项目空间数据保护（ProjectProtection） • 添加可信项目空间（add trustedproject） • 移除可信项目空间（remove trustedproject） • 查看可信项目空间（list trustedprojects） • 查看项目空间的安全配置（show SecurityConfiguration）
用户和角色操作	• 添加用户（add user） • 删除用户（remove user） • 查看用户列表（list users） • 创建角色（create role） • 查看角色列表（list roles） • 为用户授予角色（grant） • 删除角色中的用户（revoke） • 删除角色（drop role）
授权操作	• 为角色或用户授权（grant） • 撤销角色或用户授权（revoke）
表操作	• 创建表（create table） • 修改表的所有人（alter table） • 删除表（drop table） • 查看表或视图信息（desc） • 查看分区信息（desc partition） • 列出空间下的表（show tables） • 列出所有分区（show partitions）
分区和列操作	• 添加分区（alter table add） • 删除分区（alter table drop） • 添加列或注释（alter table add columns） • 修改列名及注释（alter table change column）

类别	常用命令
实例操作	• 查看实例信息（show） • 查看实例状态（status） • 查看正在执行的实例信息（top instance） • 停止实例（kill） • 获取实例对应的作业信息（desc） • 获取实例对应的作业运行日志信息（wait）
资源操作	• 添加资源（add） • 查看资源列表（list） • 为资源创建别名（alias） • 下载资源（get） • 删除资源（drop）
函数操作	• 注册函数（create function） • 注销函数（drop function） • 查看函数（desc function） • 查看函数列表（list functions）
Tunnel操作	• 上传数据（Tunnel upload） • 下载数据（Tunnel download）
时区配置操作	时区配置操作
其它操作	计量预估（cost sql）

使用说明

MaxCompute命令中的关键字、项目空间名、表名和列名均对大小写不敏感。

2.2. 项目空间操作

创建MaxCompute项目空间后，您需要进入项目空间才可以执行后续开发、分析、运维等一系列操作。本文为您介绍如何进入或查看项目空间，以及设置或查看项目空间Project级的属性。

项目空间操作的常用命令如下。

类型	功能	角色	操作入口
进入项目空间	进入具备访问权限的项目空间。	具备项目空间访问权限的用户	本文中的命令您需要在MaxCompute客户端执行。
查看项目空间属性	查看项目空间配置的属性信息。	项目空间Owner	
设置项目空间属性	设置项目空间的属性信息。	项目空间Owner	
查看项目空间账号体系	显示当前项目空间支持的账号体系信息。	项目空间Owner	
增加RAM账号体系	为当前项目空间增加RAM账号体系。	项目空间Owner	
删除RAM账号体系	删除当前项目空间的RAM账号体系。	项目空间Owner	

注意事项

- 项目空间非工作空间。项目空间名称获取方式：登录MaxCompute控制台，**项目管理**页签下的**MaxCompute项目名称**即项目空间名称。
- MaxCompute没有提供创建和删除项目空间的命令。创建项目空间操作详情请参见创建MaxCompute项目。

进入项目空间

进入指定的、具有访问权限的项目空间。进入项目空间后，可以直接在该项目空间下操作对象。

- 命令格式

```
--进入项目空间。
use <project_name>;
```

- 参数说明

  project_name：待访问项目空间的名称。如果项目空间不存在或您未被添加至该项目空间时，返回报错。

- 使用示例
  - 示例1：进入指定项目空间并访问项目空间下的对象。

    ```
    --当前项目空间为my_project，待访问项目空间为my_project_test。my_project_test是用户有权限访问的一个项目空间。
    odps@ my_project>use my_project_test;
    --进入项目空间后，即可执行其他操作。例如查询my_project_test项目空间下的表test_src。
    odps@ my_project_test>select * from test_src;
    ```

    MaxCompute自动在项目空间my_project_test中搜索表。如果表存在，返回表中的数据；如果表不存在，返回报错。
  - 示例2：跨项目访问另一项目空间下的对象，需要具备目标项目空间对象的相应操作权限，并指定项目空间名称。

    ```
    --在my_project_test项目空间下访问另一项目空间my_project2的表test_src。
    odps@ my_project_test>select * from my_project2.test_src;
    ```

查看项目空间属性

显示当前项目空间Project级的属性配置信息。MaxCompute也支持查看Session级的属性，详情请参见show flags。

- 命令格式

  ```
  setproject;
  ```

- 属性说明

  项目空间的常用属性如下。

属性名称（KEY）	描述	取值范围（VALUE）
odps.sql.allow.fullscan	设置项目空间是否允许全表扫描。全表扫描会占用大量资源，为提升处理效率，不建议开启该功能。	◦ true：允许全表扫描。 ◦ false：禁止全表扫描。
odps.table.lifecycle	设置项目空间下的表是否需要配置生命周期。	◦ optional：创建表时，Lifecycle子句为可选设置，如果不设置表的生命周期，则该表永久有效。 ◦ mandatory：Lifecycle子句为必选设置，用户必须设置表的生命周期。 ◦ inherit：创建表时，如果不设置表的生命周期，则该表的生命周期为odps.table.lifecycle.value的值。
odps.table.lifecycle.value	设置表的生命周期。单位为：天。	1~37231，默认值为37231。
odps.security.ip.whitelist	设置经典网络场景下可以访问项目空间的IP白名单。详情请参见管理IP白名单。	IP列表，以英文逗号（,）分隔。
odps.security.vpc.whitelist	指定VPC网络场景下可以访问项目空间的IP白名单。详情请参见管理IP白名单。	RegionID_VPCID[IP Address]。
READ_TABLE_MAX_ROW	设置SELECT语句返回的数据条数。	1~10000，默认值为10000。
odps.sql.type.system.odps2	2.0新数据类型开关。2.0数据类型详情请参见2.0数据类型版本。	◦ true：打开 ◦ false：关闭
odps.sql.hive.compatible	Hive兼容模式开关。打开Hive兼容模式后，MaxCompute才支持Hive指定的各种语法，例如 `inputRecordReader` 、 `outputRecordReader` 和 `Serde` 。Hive兼容数据类型详情请参见Hive兼容数据类型版本。	◦ true：打开 ◦ false：关闭
odps.sql.decimal.odps2	DECIMAL 2.0数据类型 `DECIMAL(precision,scale)` 开关。详情请参见2.0数据类型版本。	◦ true：打开 ◦ false：关闭
odps.timemachine.retention.days	设置备份数据的保留天数。详情请参见备份与恢复。	0~30，默认值为1。
odps.instance.priority.enable	使用包年包月计算资源项目空间的作业优先级功能开关。详情请参见作业优先级。	0~9，默认值为9。数值越小，优先级越高。
odps.output.field.formatter	设置SQL查询结果的动态脱敏规则。详情请参见MaxCompute数据动态脱敏。	自定义的动态脱敏规则。
odps.sql.metering.value.max	设置单SQL消费限制值。详情请参见消费监控告警消费控制。	无。

属性名称（KEY）	描述	取值范围（VALUE）
odps.sql.acid.table.enable	设置是否开启ACID机制。更多ACID信息，请参见ACID语义。	○ true：打开 ○ false：关闭
odps.sql.timezone	设置MaxCompute项目的时区。更多时区信息，请参见时区配置操作。	无。
odps.sql.unstructured.oss.commit.mode	设置是否开启通过OSS的分片上传功能向OSS外部表写入数据。详情请参见将数据写入OSS。	○ true：打开 ○ false：关闭
odps.sql.groupby.orderby.position.alias	设置是否开启将 `group by` 和 `order by` 中的整型常量做为SELECT的列序号处理。 **说明** 对于存量项目，如果在项目级别开启该参数，可能会对您已有的任务解析和执行操作造成影响，请确认并务必保证存量任务在此参数设置下仍可以按照原有逻辑正确执行后，再修改该参数。否则，请在Session级别设置。	○ true：打开 ○ false：关闭
odps.forbid.fetch.result.by.bearertoken	设置是否禁止Logview的**Result**页签显示作业运行结果。该参数主要用于保护数据安全。	○ true：禁止显示作业运行结果 ○ false：允许显示作业运行结果

设置项目空间属性

设置项目空间的Project级属性。设置项目空间属性命令会存在0~5分钟的时延。命令执行成功后，请您等待5分钟后验证执行结果。MaxCompute也支持设置Session级的属性，详情请参见set。

- 命令格式

```
setproject <KEY>=<VALUE>;
```

- 参数说明
  - KEY：属性名称。
  - VALUE：属性值。属性详情请参见查看项目空间属性。
- 使用示例

  设置允许在项目空间执行全表扫描操作。

```
setproject odps.sql.allow.fullscan=true;
```

查看项目空间账号体系

显示当前项目空间支持的账号体系信息。账号体系包括ALIYUN、RAM两种。命令格式如下：

```
list accountproviders;
```

说明 MaxCompute项目空间默认仅能识别阿里云账号体系，无法识别RAM账号体系。

增加RAM账号体系

为当前项目空间增加RAM账号体系。命令格式如下：

```
add accountprovider ram;
```

删除RAM账号体系

删除当前项目空间的RAM账号体系。命令格式如下：

```
remove accountprovider ram;
```

2.3. SET操作

MaxCompute支持在Session级设置MaxCompute系统变量，本文为您介绍如何设置及查看MaxCompute系统变量，影响MaxCompute的行为。

set操作相关命令如下。

类型	功能	角色	操作入口
set	对当前Session设置MaxCompute系统变量。	具备项目空间操作权限的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
show flags	显示set命令设置的属性。		

set

对当前Session设置MaxCompute系统变量。MaxCompute也支持设置Project级的属性，详情请参见设置项目空间属性。

- 命令格式

```
set <KEY>=<VALUE>
```

- 参数说明
  - KEY：属性名称。
  - VALUE：属性值。

    Session级的常用属性如下。

属性名称	属性描述	取值范围
console.sql.result.instancetunnel	InstanceTunnel开关。详情请参见使用说明。	■ True：打开 ■ False：关闭
odps.stage.mapper.mem	设置每个Map Worker的内存大小。	256 MB~12288 MB。默认值为1024 MB。
odps.stage.reducer.mem	设置每个Reduce Worker的内存大小。	256 MB~12288 MB。默认值为1024 MB。
odps.stage.joiner.mem	设置每个Join Worker的内存大小。	256 MB~12288 MB。默认值为1024 MB。
odps.stage.mem	设置MaxCompute指定任务下所有Worker的内存大小。优先级低于odps.stage.mapper.mem、odps.stage.reducer.mem和odps.stage.joiner.mem属性。	256 MB~12288 MB。无默认值。
odps.stage.mapper.split.size	修改每个Map Worker的输入数据量，即输入文件的分片大小，从而间接控制每个Map阶段下Worker的数量。 **说明** SQL语句中使用Limit，会限制Limit作用的Worker单并发运行。因此在设置该配置项的时候，SQL语句中应避免使用Limit。	单位MB，默认值为256 MB。
odps.stage.reducer.num	修改每个Reduce阶段的Worker数量。 **说明** SQL语句中使用Limit，会限制Limit作用的Worker单并发运行。因此在设置该配置项的时候，SQL语句中应避免使用Limit。	无默认值。
odps.stage.joiner.num	修改每个Join阶段的Worker数量。 **说明** SQL语句中使用Limit，会限制Limit作用的Worker单并发运行。因此在设置该配置项的时候，SQL语句中应避免使用Limit。	无默认值。
odps.stage.num	修改MaxCompute指定任务下所有Worker的并发数，优先级低于odps.stage.mapper.split.size、odps.stage.reducer.mem和odps.stage.joiner.num属性。 **说明** SQL语句中使用Limit，会限制Limit作用的Worker单并发运行。因此在设置该配置项的时候，SQL语句中应避免使用Limit。	无默认值。

属性名称	属性描述	取值范围
odps.sql.reshuffle.dynamicpt	动态分区开关，以避免拆分动态分区时产生过多小文件。	■ True：打开 ■ False：关闭 说明 如果生成的动态分区个数很少，建议将值设为False，以避免数据倾斜。
odps.sql.type.system.odps2	2.0新数据类型开关。2.0数据类型详情请参见2.0数据类型版本。	■ True：打开 ■ False：关闭
odps.sql.hive.compatible	Hive兼容模式开关。打开Hive兼容模式后，MaxCompute才支持Hive指定的各种语法，例如 `inputRecordReader` 、 `outputRecordReader` 和 `Serde` 。Hive兼容数据类型详情请参见Hive兼容数据类型版本。	■ True：打开 ■ False：关闭
odps.sql.executionengine.coldata.deep.buffer.size.max	设置MaxCompute在写表过程中，为复杂数据类型的列预先申请的缓存大小，以便提高写入性能。 如果输出的表中的复杂数据类型过多或输出表中含有的某个单独的复杂类型变量大小过大，可以设置该参数。 ■ 如果输出的表有3个列的Schema是复杂数据类型，例如列类型为（STRING、MAP、STRUCT、ARRAY或BINARY），则默认情况下MaxCompute将会为写表操作预留64 MB×3大小的内存。每一列预先申请的缓存将会用来存放这一列 `batch row count` 行的数据。 ■ 如果预估表的复杂类型变量占用的空间比较小，建议调小此值。例如，如果每个复杂类型变量大小不会超过1024 Byte，同时 `batch row count` 值使用的是默认值（1024），则可以将属性值设置为1024×1024，示例如下。 `set odps.sql.executionengine.coldata.deep.buffer.size.max=1048576;` ■ 如果您预先知道每个复杂类型的值都在7 MB~8 MB间，同时指定了 `batch row count` 为32，则该值可以被调整为8 MB×32。 ■ 如果输出的表有复杂类型，或 `MAPJOIN` 小表有复杂类型，调整该值会影响到作业执行过程中使用的内存。根据前面的计算方法，值设的过大有可能导致OOM（Out Of Memory）内存溢出。	单位Byte，默认值为67108864 Byte。
odps.task.quota.preference.tag	指定作业的Quota组（即MaxCompute管家中的配额组）。使用包年包月资源的项目空间可以通过该属性指定作业使用某个具体的二级Quota组。如果提交作业时设置的Quota Tag和某个Quota组属性中的Quota Tag相等，作业就会被优先调度到这个Quota组中。否则，会被调度到所属项目空间指定的Quota组中。 执行如下语句进行设置。 `set odps.task.quota.preference.tag = tag_name;` 说明 该属性当前仅支持SQL作业。	tag_name为MaxCompute管家中配额组的**配额组标签**。配额组只能取作业所属项目空间Owner所在区域的配额组。tag_name只允许使用字母、数字和下划线（_）。 说明 仅当您在MaxCompute管家中为配额组设置了标签后，才可以使用odps.task.quota.preference.tag参数。
odps.sql.udf.getjsonobj.new	设置GET_JSON_OBJECT函数的返回行为是否开启保留原始字符串的方式。 自2021年1月21日及之后新创建的MaxCompute项目中，GET_JSON_OBJECT函数的返回行为默认为保留原始字符串。2021年1月21日之前创建的MaxCompute项目中，GET_JSON_OBJECT函数的返回行为默认为JSON保留字符转义，以免行为变更对存量作业造成影响。 更多GET_JSON_OBJECT函数信息，请参见GET_JSON_OBJECT。	■ True：打开 ■ False：关闭

属性名称	属性描述	取值范围
odps.sql.udf.jvm.memory	设置UDF JVM Heap使用的最大内存。 某些UDF在内存计算、排序的数据量比较大时，会报内存溢出错误，这时候可以调大该参数，但该方法只能暂时缓解，您需要从业务上去优化UDF代码。	256 MB~12288 MB。默认值为1024 MB。
odps.sql.udf.timeout	设置UDF超时时间。	0s~3600s。默认值为600s。
LabelSecurity	设置是否开启LabelSecurity安全机制。更多LabelSecurity安全机制信息，请参见Label权限控制。	■ True：打开 ■ False：关闭
odps.sql.session.resources	指定UDT引用的资源，可以指定多个，用英文逗号（,）分隔。更多引用资源信息，请参见UDT概述。	上传的资源
odps.sql.udt.display.tostring	设置是否开启将所有以UDT为最终输出的列Wrap上 `java.util.Objects.toString(...)` 的机制。	■ True：打开 ■ False：关闭
odps.sql.session.java.imports	指定UDT引用的Java Package，可以指定多个，用英文逗号（,）分隔。更多引用Java Package信息，请参见UDT概述。	上传的Java Package
CheckPermissionUsingACL	设置是否开启ACL授权机制。更多ACL授权信息，请参见权限列表。	■ True：打开 ■ False：关闭
CheckPermissionUsingPolicy	设置是否开启Policy授权机制。更多Policy授权信息，请参见Policy权限控制。	■ True：打开 ■ False：关闭
ObjectCreatorHasAccessPermission	设置对象创建者是否默认拥有访问权限。	■ True：允许对象创建者默认拥有访问权限 ■ False：禁止对象创建者默认拥有访问权限
ObjectCreatorHasGrantPermission	设置对象创建者是否默认拥有授权权限。	■ True：允许对象创建者默认拥有授权权限 ■ False：禁止对象创建者默认拥有授权权限
ProjectProtection	设置是否开启数据保护机制。更多数据保护机制信息，请参见数据保护机制。	■ True：打开 ■ False：关闭
odps.output.field.formatter	设置SQL查询结果的动态脱敏规则。更多动态脱敏信息，请参见MaxCompute数据动态脱敏。	自定义的动态脱敏规则
odps.isolation.session.enable	设置是否在单Session级别开启动态脱敏。更多动态脱敏信息，请参见MaxCompute数据动态脱敏。	■ True：打开 ■ False：关闭
odps.sql.skewjoin	设置是否开启SKEWJOIN功能，解决长尾问题。	■ True：打开 ■ False：关闭
odps.sql.skewinfo	设置SKEWJOIN的目标Key及对应的值。更多信息，请参见JOIN长尾优化。	无
odps.sql.reducer.instances	设置哈希桶的数量。更多哈希桶信息，请参见表操作。	0~4000
odps.sql.udf.ppr.deterministic	设置是否在自定义函数中开启分区裁剪机制。更多分区裁剪信息请参见WHERE子句（where_condition）。	■ True：打开 ■ False：关闭
odps.sql.udf.ppr.to.subquery	设置是否开启忽略分区裁剪回填报错问题机制。更多分区裁剪信息请参见WHERE子句（where_condition）。	■ True：打开 ■ False：关闭
odps.optimizer.enable.range.partial.repartitioning	设置Range聚簇表是否开启Shuffle Remove机制。	■ True：打开 ■ False：关闭
odps.optimizer.skew.join.topk.num	设置优化器运行Aggregate时获取的热值数量。详情请参见SKEWJOIN HINT。	无

属性名称	属性描述	取值范围
odps.optimizer.stat.collect.auto	设置是否开启Freeride功能。开启后自动收集表的Column Stats。详情请参见优化器信息收集。	■ True：打开 ■ False：关闭
odps.optimizer.stat.collect.plan	设置收集计划，收集指定列的指定Column Stats指标。详情请参见优化器信息收集。	无
odps.sql.executionengine.batch.rowcount	设置SQL引擎一次同时处理的数据行数。默认值是1024。一般不需要手动调整，遇到单行大字段导致OOM或UDF性能较差导致的超时场景需要手动调小。建议不要设置太小否则会影响性能。	1~1024
odps.sql.external.net.vpc	设置是否开启外部表VPC支持。详情请参见HBase外部表（HBase标准版或增强版）、RDS外部表或Hologres外部表。	■ True：打开 ■ False：关闭
odps.sql.groupby.position.alias	设置是否开启将 `groupby` 中的整型常量做为SELECT的列序号处理。	■ True：打开 ■ False：关闭
odps.sql.groupby.skewindata	设置是否开启 `groupby` 防倾斜机制。	■ True：打开 ■ False：关闭
odps.sql.orderby.position.alias	设置是否开启将 `orderby` 中的整型常量做为SELECT的列序号处理。	■ True：打开 ■ False：关闭
odps.sql.groupby.orderby.position.alias	设置是否开启将 `group by` 和 `order by` 中的整型常量做为SELECT的列序号处理。**推荐**使用该参数。 **说明** 对于存量项目，如果在项目级别开启该参数，可能会对您已有的任务解析和执行操作造成影响，请确认并务必保证存量任务在此参数设置下仍可以按照原有逻辑正确执行后，再修改该参数。否则，请在Session级别设置。	■ True：打开 ■ False：关闭
odps.sql.jobconf.odps2	设置是否开启jobconf2功能来生成作业执行计划。	■ True：打开 ■ False：关闭
odps.sql.mapjoin.memory.max	设置MAPJOIN场景小表读入内存后的大小，单位MB。	0 MB~8192 MB
odps.sql.metering.value.max	设置单SQL消费限制值。详情请参见消费监控告警消费控制。	无
odps.sql.python.version	设置运行SQL句的Python版本。	■ cp27 ■ cp37
odps.sql.select.output.format	设置是否在MaxCompute客户端的返回结果中显示表头信息。详情请参见使用客户端（odpscmd）连接。	■ `{""needHeader"":false,""fieldDelim"":""""}` ：不显示表头信息。 ■ `{""needHeader"":true,""fieldDelim"":""""}` ：显示表头信息
odps.sql.timezone	设置MaxCompute项目的时区。更多时区信息，请参见时区配置操作。	无
odps.sql.unstructured.data.oss.use.https	设置访问外部表时，是否需要开启底层通过HTTPS方式获取数据的机制。更多信息，请参见创建OSS外部表。	■ True：打开 ■ False：关闭
odps.sql.unstructured.oss.commit.mode	设置是否开启通过OSS的分片上传功能向OSS外部表写入数据。详情请参见将数据写入OSS。	■ True：打开 ■ False：关闭

- 使用示例

```
--调整每个Mapper读取数据的大小为256 MB。
set odps.stage.mapper.split.size=256;
```

show flags

显示set命令设置的属性。命令格式如下：

```
show flags;
```

2.4. 安全操作

MaxCompute提供项目数据保护机制，为数据安全提供保障。本文为您介绍常用安全操作命令。

安全操作常用命令如下。详细安全相关操作请参见安全参数列表。

类型	功能	角色	操作入口
开启项目数据保护	开启项目的数据保护机制。	项目Owner或授予Super_Administrator角色的用户	本命令需要您在MaxCompute客户端执行。
添加可信项目	在当前项目中添加可信的项目。		您可以在如下工具平台执行这些命令： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
移除可信项目	在当前项目中删除已添加的可信的项目。		
查看可信项目	在当前项目中查看已添加的可信的项目。	项目Owner或授予Super_Administrator角色或Admin角色的用户	
查看项目的安全配置	在当前项目中查看项目的安全配置属性。		

开启项目数据保护

开启项目的数据保护机制。您只能在项目中访问数据。数据只能在项目内流动，不允许流出到项目之外，以提高数据的安全性。命令格式如下：

```
ProjectProtection={True|False};
```

添加可信项目

在当前项目中添加可信的项目，通过项目互信的方式确保数据顺利流转。

- 命令格式

```
add trustedproject <project_name>;
```

- 参数说明

  project_name：必填。待添加可信项目的名称。

移除可信项目

在当前项目中删除已添加的可信的项目。

- 命令格式

```
remove trustedproject <project_name>;
```

- 参数说明

  project_name：必填。待删除可信项目的名称。

查看可信项目

在当前项目中查看已添加的可信的项目。命令格式如下：

```
list trustedprojects;
```

查看项目的安全配置

在当前项目中查看项目的安全配置属性。命令格式如下：

```
show SecurityConfiguration;
```

2.5. 用户和角色操作

当您需要与其它用户协同开发MaxCompute项目时，您可以添加用户至MaxConpute项目空间。如果需要添加多个用户且授予相同权限，您可以创建角色批量授权。本文为您介绍如何添加、删除、查看用户或角色。

用户和角色常用操作命令如下。

类型	功能	角色	操作入口

类型	功能	角色	操作入口
添加用户	向MaxCompute项目空间添加阿里云用户或RAM用户。	项目空间Owner或授予Super_Administrator角色或Admin角色的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • MaxCompute Studio
删除用户	从MaxCompute项目空间中删除阿里云用户或RAM用户。		
查看用户列表	查看MaxCompute项目空间中的用户信息。		
创建角色	在MaxCompute项目空间中创建角色。		• MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
查看角色列表	查看MaxCompute项目空间中的所有角色信息。		
为用户授予角色	为用户授予某个角色。		
删除角色中的用户	删除指定角色内的用户。		
删除角色	删除MaxCompute项目空间中已存在的角色。		

添加用户

向MaxCompute项目空间添加阿里云用户或RAM用户。

- 使用限制

  添加RAM用户时，MaxCompute只允许阿里云主账号将自身的RAM用户加入到项目空间中，不允许加入其它阿里云账号的RAM用户。

- 命令格式

```
add user <user_name>;
```

- 参数说明

  user_name：必填。待添加阿里云账号或RAM用户账号信息。阿里云账号格式为 `ALIYUN$****@aliyun.com;` ，RAM用户账号格式为 `RAM$****` 。

- 使用示例
  - 示例1：添加阿里云账号test_user@aliyun.com至MaxCompute项目空间。

    ```
    add user ALIYUN$test_user@aliyun.com;
    ```

  - 示例2：添加RAM用户ram_test_user至MaxCompute项目空间。

    ```
    add user RAM$ram_test_user;
    ```

删除用户

当一个用户离开MaxCompute项目团队时，需要将该用户从项目空间中移除。用户一旦从项目空间中被移除，该用户将不再拥有访问该项目空间任何资源的权限。

- 使用限制
  - 删除一个用户前，如果该用户已被赋予某些角色，则需要先撤销该用户的所有角色之后再执行移除操作。
  - MaxCompute不支持在项目空间中彻底移除一个用户及其所有权限数据。当一个用户被移除后，与该用户有关的授权仍然会被保留。一旦该用户后续被再次添加到该项目空间时，该用户的历史授权访问权限将被重新激活。

- 命令格式

```
remove user <user_name>;
```

- 参数说明

  user_name：必填。待删除阿里云用户或RAM用户账号信息。阿里云账号格式为 `ALIYUN$****@aliyun.com;` ，RAM用户账号格式为 `RAM$****` 。

- 使用示例
  - 示例1：移除阿里云用户test_user@aliyun.com。

    ```
    remove user ALIYUN$test_user@aliyun.com;
    ```

  - 示例2：移除RAM用户ram_test_user。

    ```
    remove user RAM$ram_test_user;
    ```

查看用户列表

查看MaxCompute项目空间中的用户信息。命令格式：

```
list users;
```

创建角色

在MaxCompute项目空间中创建角色。

- 命令格式

```
create role <role_name>;
```

- 参数说明

  role_name：必填。待添加的角色名称。

- 使用示例

```
--创建一个player角色。
create role player;
```

查看角色列表

查看MaxCompute项目空间中的所有角色信息。命令格式：

```
list roles;
```

为用户授予角色

为用户授予某个角色，用户即可具备角色被赋予的权限。

- 使用限制

  在对用户授予角色前，您需要先为角色授予项目空间对象的相关操作权限。详情请参见为角色或用户授权。

- 命令格式

```
grant <role_name> to <user_name>;
```

- 参数说明
  - role_name：必填。待授予的角色名称。
  - user_name：必填。待授予角色的阿里云用户或RAM用户账号信息。阿里云账号格式为 `ALIYUN$****@aliyun.com;` ，RAM用户账号格式为 `RAM$****` 。

- 使用示例

```
--对阿里云用户test_user@aliyun.com赋予角色player。
grant player to ALIYUN$test_user@aliyun.com;
```

删除角色中的用户

删除指定角色内的用户。

- 命令格式

```
revoke <role_name> from <user_name>;
```

- 参数说明
  - role_name：必填。待删除用户所属的角色名称。
  - user_name：必填。待删除阿里云用户或RAM用户的账号信息。阿里云账号格式为 `ALIYUN$****@aliyun.com;` ，RAM用户账号格式为 `RAM$****` 。

- 使用示例

```
--删除角色player中的阿里云用户alice@aliyun.com。
revoke player from ALIYUN$alice@aliyun.com;
```

删除角色

删除MaxCompute项目空间中已存在的角色。

- 使用限制

  删除角色前，需要确保角色内已无用户。

- 命令格式

```
drop role <role_name>;
```

- 参数说明

  role_name：必填。待删除的角色名称。

- 使用示例

```
--删除一个player角色。
drop role player;
```

2.6. 授权操作

当您在项目空间中添加用户或角色后，需要对用户授予客体的某操作权限，或对角色进行授权并将角色授予用户，授权后用户才具备相应客体的某操作权限。本文为您介绍如何对用户或角色授权、撤销授权。

各客体及对应操作类型详情请参见权限列表。常用授权操作命令如下。

类型	功能	角色	操作入口
为角色或用户授权	为角色或用户授予某类客体的某种操作权限。	项目空间Owner或授予Super_Administrator角色或Admin角色的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
撤销角色或用户授权	为角色或用户撤销某类客体的某种操作权限。		

为角色或用户授权

为角色或用户授予某类客体的某种操作权限。

- 命令格式

```
grant <Action> [, <Action>] on <Object> <object_name> to {user|role} {<user_name|role_name>};
```

- 参数说明
  - Action：必填。待授予操作权限的名称。详情请参见权限列表。
  - Object：必填。待授予用户或角色的操作权限所属客体类型。详情请参见权限列表。
  - object_name：必填。客体的名称。
  - user|role：二者选其一，user表示授权给用户，role表示授权给角色。
  - user_name|role_name：待授予用户或角色的名称。
- 使用示例
  - 示例1：为test_user@aliyun.com用户授予项目空间名为prj1的CreateTable（创建表）权限。

    ```
    grant CreateTable on Project prj1 to user ALIYUN$test_user@aliyun.com;
    ```

  - 示例2：为player角色授予项目空间名为prj1的CreateInstance（创建实例）权限。

```
grant CreateInstance on Project prj1 to role player;
```

撤销角色或用户授权

为角色或用户撤销某类客体的某种操作权限。

- 命令格式

```
revoke <Action> [, <Action>] on <Object> <object_name> from {user|role} {<user_name|role_name>};
```

- 参数说明
  - Action：必填。待撤销操作权限的名称。详情请参见权限列表。
  - Object：必填。待撤销用户或角色的操作权限所属客体类型。详情请参见权限列表。
  - object_name：必填。客体的名称。
  - user|role：二者选其一，user表示撤销用户授权，role表示撤销角色授权。
  - user_name|role_name：待撤销授权的用户或角色的名称。
- 使用示例
  - 示例1：为test_user@aliyun.com用户撤销项目空间名为prj1的CreateTable（创建表）权限。

    ```
    revoke CreateTable on Project prj1 from user ALIYUN$test_user@aliyun.com;
    ```

  - 示例2：为player角色撤销项目空间名为prj1的CreateInstance（创建实例）权限。

    ```
    revoke CreateInstance on Project prj1 from role player;
    ```

2.7. 表操作

表是MaxCompute的数据存储单元。数据仓库的开发、分析及运维都需要对表数据进行处理。本文为您介绍如何创建、删除和查看表等常用表操作。

表操作常用命令如下。表操作详情请参见表操作。

类型	功能	角色	操作入口
创建表	创建非分区表或分区表。	具备项目空间创建表权限（CreateTable）的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
修改表的所有人	修改表的所有人，即表Owner。	项目空间Owner	
删除表	删除非分区表或分区表。	具备删除表权限（Drop）的用户。	
查看表或视图信息	查看MaxCompute内部表、视图或外部表的信息。	具备读取表元信息权限（Describe）的用户。	
查看分区信息	查看某个分区表具体的分区信息。	具备读取表元信息权限（Describe）的用户。	
列出项目空间下的表和视图	列出项目空间下所有的表、视图或符合某规则（支持正则表达式）的表、视图。	具备项目空间查看对象列表权限（List）的用户。	
列出所有分区	列出一张表中的所有分区。	具备项目空间查看对象列表权限（List）的用户。	

关于分区和列的操作详情请参见分区和列操作。

关于表生命周期的操作详情请参见生命周期操作。

创建表

创建非分区表、分区表、外部表或聚簇表。

- 限制条件
  - 分区表的分区层级不能超过6级。例如某张表以日期为分区列，分区层级为`年/月/周/日/时/分`。
  - 一张表允许的分区个数支持按照具体的项目配置，默认为6万个。

  更多表的限制条件，请参见SQL使用限制项。
- 命令格式

```
--创建新表。
 create [external] table [if not exists] <table_name>
 [(<col_name> <data_type> [not null] [default <default_value>] [comment <col_comment>], ...)]
 [comment <table_comment>]
 [partitioned by (<col_name> <data_type> [comment <col_comment>], ...)]
 --用于创建聚簇表时设置表的Shuffle和Sort属性。
 [clustered by | range clustered by (<col_name> [, <col_name>, ...]) [sorted by (<col_name> [asc | desc] [, <col_name> [asc | desc] ...])] into <number_of_buckets> buckets]
 --仅限外部表。
 [stored by StorageHandler]
 --仅限外部表。
 [with serdeproperties (options)]
 --仅限外部表。
 [location <osslocation>]
 --指定表为Transactional表，后续可以对该表执行更新或删除表数据操作，但是Transactional表有部分使用限制，请根据需求创建。
 [tblproperties("transactional"="true")]
 [lifecycle <days>];
--基于已存在的表创建新表并复制数据，但不复制分区属性。
create table [if not exists] <table_name> [lifecycle <days>] as <select_statement>;
--基于已存在的表创建具备相同结构的新表但不复制数据。
create table [if not exists] <table_name> like <existing_table_name> [lifecycle <days>];
```

- 参数说明
  - external：可选。表示创建的表为外部表。
  - if not exists：可选。如果不指定if not exists选项而存在同名表，会报错。如果指定if not exists，只要存在同名表，即使原表结构与要创建的目标表结构不一致，均返回成功。已存在的同名表的元数据信息不会被改动。
  - table_name：必填。表名。表名大小写不敏感，不能有特殊字符，只能包含a~z、A~Z、数字和下划线（_）。建议以字母开头，名称的长度不超过128字节，否则报错。
  - col_name：可选。表的列名。列名大小写不敏感，不能有特殊字符，只能包含a~z、A~Z、数字和下划线（_）。建议以字母开头，名称的长度不超过128字节，否则报错。
  - col_comment：可选。列的注释内容。注释内容为长度不超过1024字节的有效字符串，否则报错。

- data_type：可选。列的数据类型，包含BIGINT、DOUBLE、BOOLEAN、DATETIME、DECIMAL和STRING等多种数据类型，详情请参见数据类型版本说明。
- not null：可选。禁止该列的值为NULL。更多修改非空属性信息，请参见修改表的列非空属性。
- default_value：可选。指定列的默认值，当 `insert` 操作不指定该列时，该列写入默认值。
- table_comment：可选。表注释内容。注释内容为长度不超过1024字节的有效字符串，否则报错。
- partitioned by (<col_name> <data_type> [comment <col_comment>], ...)：可选。指定分区表的分区字段。
  - col_name：表的分区列名。列名大小写不敏感，不能有特殊字符，只能包含a~z、A~Z、数字和下划线（_）。建议以字母开头，名称的长度不超过128字节，否则报错。
  - data_type：分区列的数据类型。MaxCompute 1.0版本仅支持STRING类型。MaxCompute 2.0版本扩充了分区类型，包含TINYINT、SMALLINT、INT、BIGINT、VARCHAR和STRING类型。详情请参见数据类型版本说明。当使用分区字段对表进行分区时，新增分区、更新分区内数据和读取分区数据均不需要做全表扫描，提高处理效率。
  - col_comment：分区列的注释内容。注释内容为长度不超过1024字节的有效字符串，否则报错。

  > **说明** 分区值不能包含双字节字符（如中文），必须以字母开头，包含字母、数字和允许的字符，长度不超过128字节。允许的字符包括空格、冒号（:）、下划线（_）、美元符号（$）、井号（#）、英文句点（.）、感叹号（!）和at（@），其他字符的行为未定义，例如转义字符 `\t`、`\n` 和 `/`。

- clustered by | range clustered by (<col_name> [, <col_name>, ...]) [sorted by (<col_name> [asc | desc] [, <col_name> [asc | desc] ...])] into <number_of_buckets> buckets：可选。用于创建聚簇表时设置表的Shuffle和Sort属性。

  聚簇表分为Hash聚簇表和Range聚簇表两种：

  - Hash聚簇表
    - clustered by：指定Hash Key。MaxCompute将对指定列进行Hash运算，按照Hash值分散到各个Bucket中。为避免数据倾斜和热点，并取得较好的并行执行效果，`clustered by` 列适宜选择取值范围大，重复键值少的列。此外，为了达到 `join` 优化的目的，也应该考虑选取常用的Join或Aggregation Key，即类似于传统数据库中的主键。
    - sorted by：指定Bucket内字段的排序方式。建议sorted by和clustered by保持一致，以取得较好的性能。此外，当指定sorted by子句后，MaxCompute将自动生成索引，并且在查询时利用索引来加快执行。
    - number_of_buckets：指定哈希桶的数量。该值必须填写，且由数据量大小决定。此外，MaxCompute默认支持最多1111个Reducer，所以此处最多只支持1111个哈希桶。您可以使用 `set odps.sql.reducer.instances=xxx;` 来提升这个限制，但最大不得超过4000，否则会影响性能。

      选择哈希桶数目时，请您遵循以下两个原则：

      - 哈希桶大小适中：建议每个Bucket的大小为500 MB左右。例如，分区大小估计为500 GB，初略估算Bucket数目应该设为1000，这样平均每个Bucket大小约为500 MB。对于特别大的表，500 MB的限制可以突破，每个Bucket在2 GB~3 GB左右比较合适。同时，可以结合 `set odps.sql.reducer.instances=xxx;` 来突破1111个桶的限制。
      - 对于 `join` 优化场景，去除Shuffle和Sort步骤能显著提升性能。因此要求两个表的哈希桶数目成倍数关系，例如256和512。建议哈希桶的数量统一使用2^n，例如512、1024、2048或4096，这样系统可以自动进行哈希桶的分裂和合并，也可以去除Shuffle和Sort的步骤，提升执行效率。
  - Range聚簇表
    - range clustered by：指定范围聚簇列。MaxCompute将对指定列进行分桶运算，按照分桶编号分散到各个Bucket中。
    - sorted by：指定Bucket内字段的排序方式，使用方法与Hash聚簇表相同。
    - number_of_buckets：指定哈希桶的数量。Range聚簇表的Bucket桶数没有Hash聚簇表的2^n最佳实践，在数据量分布合理的情况下，任何桶数都可以。Range聚簇表的Bucket数不是必须的，可以省略，此时系统会根据数据量自动决定最佳的Bucket数目。
    - 当Join或Aggregation的对象是Range聚簇表，且Join Key或Group Key是Range Clustering Key或其前缀时，可以通过控制Flag消除数据的重分布，即Shuffle Remove，提升执行效率。您可以通过 `set odps.optimizer.enable.range.partial.repartitioning=true/flase;` 进行设置，默认关闭。

    > **说明**
    > - 聚簇表的优点：
    >   - 优化Bucket Pruning。
    >   - 优化Aggregation。
    >   - 优化存储。
    > - 聚簇表的限制：
    >   - 不支持 `insert into`，只能通过 `insert overwrite` 来添加数据。
    >   - 不支持Tunnel直接Upload到Range聚簇表，因为Tunnel上传数据是无序的。

- stored by StorageHandler：可选。按照外部表数据格式指定StorageHandler。
- with serdeproperties (options)：可选。外部表的授权、压缩、字符解析等相关参数。
- osslocation：可选。外部表数据OSS存储位置，详情请参见创建OSS外部表。

- tblproperties("transactional"="true")：可选。设置表为Transactional表。后续可以对Transactional表执行 `update` 、 `delete` 操作实现行级更新或删除数据。更多信息，请参见更新或删除数据（UPDATE|DELETE）。

  Transactional表的使用限制如下：

  - MaxCompute只允许在创建表时设置Transactional属性。已创建的表不允许通过 `alter table` 方式修改Transactional属性，执行如下语句会报错：

    ```
    alter table not_txn_tbl set tblproperties("transactional"="true");
    --报错。
    FAILED: Catalog Service Failed, ErrorCode: 151, Error Message: Set transactional is not supported
    ```

  - 在创建表时，不支持将聚簇表、外部表设置为Transactional表。
  - 不支持MaxCompute内部表、外部表、聚簇表与Transactional表互转。
  - 不支持其他系统的作业（例如MaxCompute Spark、PAI、Graph）访问Transactional表。
  - 不支持 `clone table` 、 `merge partition` 操作。
  - 不支持通过备份与恢复功能备份数据，因此在对Transactional表的重要数据执行 `update` 、 `delete` 或 `insert overwrite` 操作前需要手动通过 `select` + `insert` 操作将数据备份至其他表中。

- lifecycle：可选。表的生命周期，仅支持正整数。单位：天。
  - 非分区表：自最后一次修改表数据开始计算，经过days天后数据无改动，则您无需干预此表，MaxCompute会自动回收（类似drop table操作）。
  - 分区表：系统根据各分区的LastDataModifiedTime判断是否需要回收分区。不同于非分区表，分区表的最后一个分区被回收后，该表不会被删除。生命周期只能设定到表级别，不支持在分区级别设置生命周期。

- 通过 `create table [if not exists] <table_name> [lifecycle <days>] as <select_statement>;` 语句可以再创建一个表，并在建表的同时将数据复制到新表中。但通过该语句创建的表不会复制分区属性，只会把源表的分区列作为目标表的一般列处理，也不会复制源表的生命周期属性。您还可以通过lifecycle参数回收表。
- 通过 `create table [if not exists] <table_name> like <existing_table_name> [lifecycle <days>];` 语句可以再创建一个表，使目标表和源表具有相同的表结构。但通过该语句创建的表不复制数据，也不会复制源表的生命周期属性。您还可以通过lifecycle参数回收表。

- 使用示例
  - 示例1：创建非分区表test1。

    ```
    create table test1 (key STRING);
    ```

  - 示例2：创建一张分区表sale_detail。

    ```
    create table if not exists sale_detail(
     shop_name     STRING,
     customer_id   STRING,
     total_price   DOUBLE)
    partitioned by (sale_date STRING, region STRING);
    ```

  - 示例3：创建一个新表sale_detail_ctas1，将sale_detail的数据复制到sale_detail_ctas1中，并设置生命周期。

    ```
    create table sale_detail_ctas1 lifecycle 10 as select * from sale_detail;
    ```

    您可以通过 `desc extended sale_detail_ctas1;` 命令查看到表的结构及生命周期等详细信息。

    此处 `sale_detail` 是一张分区表，而通过 `create table ... as select_statement ...` 语句创建的表 `sale_detail_ctas1` 不会复制分区属性，只会把源表的分区列作为目标表的一般列处理。即 `sale_detail_ctas1` 是一个含有5列的非分区表。

  - 示例4：创建一个新表sale_detail_ctas2，在 `select` 子句中使用常量作为列的值。

    ```
    --指定列的名字。
    create table sale_detail_ctas2
    as
    select shop_name, customer_id, total_price, '2013' as sale_date, 'China' as region
    from sale_detail;
    --不指定列的名字。
    create table sale_detail_ctas3
    as
    select shop_name, customer_id, total_price, '2013', 'China'
    from sale_detail;
    ```

    > **说明** 如果在 `select` 子句中使用常量作为列的值，建议您指定列的名字。创建的表sale_detail_ctas3的第四、五列类似于 `_c4` 、 `_c5` 。

- 示例5：创建一个新表sale_detail_like，与sale_detail具有相同的表结构，并设置生命周期。

  ```
  create table sale_detail_like like sale_detail lifecycle 10;
  ```

  您可以通过 `desc extended sale_detail_like;` 命令查看到表的结构及生命周期等详细信息。

  sale_detail_like的表结构与sale_detail完全相同。除生命周期属性外，列名、列注释以及表注释等均相同。但sale_detail中的数据不会被复制到sale_detail_like表中。

- 示例6：创建使用新数据类型的表test_newtype。

  ```
  set odps.sql.type.system.odps2=true;
  CREATE TABLE test_newtype (
      c1 TINYINT
      ,c2 SMALLINT
      ,c3 INT
      ,c4 BIGINT
      ,c5 FLOAT
      ,c6 DOUBLE
      ,c7 DECIMAL
      ,c8 BINARY
      ,c9 TIMESTAMP
      ,c10 ARRAY<MAP<BIGINT,BIGINT>>
      ,c11 MAP<STRING,ARRAY<BIGINT>>
      ,c12 STRUCT<s1:STRING,s2:BIGINT>
      ,c13 VARCHAR(20))
  LIFECYCLE 1
  ;
  ```

- 示例7：创建Hash聚簇非分区表t1。

  ```
  create table t1 (a STRING, b STRING, c BIGINT) clustered by (c) sorted by (c) into 1024 buckets;
  ```

- 示例8：创建Hash聚簇分区表t2。

  ```
  create table t2 (a STRING, b STRING, c BIGINT) partitioned by (dt STRING) clustered by (c) sorted by (c) into 1024 buckets;
  ```

- 示例9：创建Range聚簇非分区表t3。

  ```
  create table t3 (a STRING, b STRING, c BIGINT) range clustered by (c) sorted by (c) into 1024 buckets;
  ```

- 示例10：创建Range聚簇分区表t4。

  ```
  create table t4 (a STRING, b STRING, c BIGINT) partitioned by (dt STRING) range clustered by (c) sorted by (c);
  ```

- 示例11：创建Transactional非分区表t5。

  ```
  create table t5(id bigint) tblproperties("transactional"="true");
  ```

- 示例12：创建Transactional分区表t6。

  ```
  create table if not exists t6(id bigint) partitioned by(ds string) tblproperties ("transactional"="true");
  ```

修改表的所有人

修改表的所有人，即表Owner。

- 命令格式

  ```
  alter table <table_name> changeowner to <new_owner>;
  ```

- 参数说明
  - table_name：必填。待修改Owner的表名。
  - new_owner：必填。修改后的Owner账号。
- 使用示例

  ```
  --将表test1的所有人修改为ALIYUN$xxx@aliyun.com。
  alter table test1 changeowner to 'ALIYUN$xxx@aliyun.com';
  ```

删除表

删除非分区表或分区表。

- 注意事项
  - 请谨慎操作，确认表可以删除后，再执行删除操作。如果误删了表，当项目开启了备份恢复功能，且删除操作未超过项目设置的备份数据保留天数时，则可以恢复表。更多备份恢复信息，请参见备份与恢复。
  - 删除表之后，MaxCompute项目的存储量会降低。

- 命令格式

```
drop table [if exists] <table_name>;
```

- 参数说明
  - if exists：可选。如果不指定if exists且表不存在，则返回异常。如果指定if exists，无论表是否存在，均返回成功。
  - table_name：必填。待删除的表名。
- 使用示例

```
--删除表sale_detail。无论sale_detail表是否存在，均返回成功。
drop table if exists sale_detail;
```

查看表或视图信息

查看MaxCompute内部表、视图、外部表、聚簇表或Transactional表的信息。如果您需要查看表的详细数据信息，请参见SELECT语法。

- 命令格式

```
--查看表或视图信息。
desc <table_name|view_name> [partition (<pt_spec>)];
--查看外部表、聚簇表或Transactional表信息。也可以查看内部表的扩展信息。
desc extended <table_name>;
```

- 参数说明
  - table_name：必填。待查看表的名称。
  - view_name：必填。待查看视图的名称。
  - pt_spec：可选。待查看分区表的指定分区。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。
  - extended：如果表为外部表、聚簇表或Transactional表，需要包含此参数。显示表的扩展信息。也可以查看内部表的扩展信息，例如列的非空属性。
- 使用示例
  - 示例1：查看创建的test1表的信息。

```
desc test1;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $project_name                      |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 17:47:48                                      |
| LastDDLTime:              2020-11-16 17:47:48                                      |
| LastModifiedTime:         2020-11-16 17:47:48                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                                     |
+------------------------------------------------------------------------------------+
| key             | string     |       |                                             |
+------------------------------------------------------------------------------------+
```

- 示例2：查看创建的sale_detail表的信息。

```
desc sale_detail;
```

返回结果如下。

```
+------------------------------------------------------------------------+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $project_name          |
| TableComment:                                                          |
+------------------------------------------------------------------------+
| CreateTime:               2017-06-28 15:05:17                          |
| LastDDLTime:              2017-06-28 15:05:17                          |
| LastModifiedTime:         2017-06-28 15:05:17                          |
+------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                      |
+------------------------------------------------------------------------+
| Native Columns:                                                        |
+------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                         |
+------------------------------------------------------------------------+
| shop_name       | string     |       |                                 |
| customer_id     | string     |       |                                 |
| total_price     | double     |       |                                 |
+------------------------------------------------------------------------+
| Partition Columns:                                                     |
+------------------------------------------------------------------------+
| sale_date       | string     |                                         |
| region          | string     |                                         |
+------------------------------------------------------------------------+
```

- 示例3：查看创建的sale_detail_ctas1表的详细信息。

```
desc extended sale_detail_ctas1;
```

返回结果如下。

```
+-------------------------------------------------------------------------------------------+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $project_name                             |
| TableComment:                                                                             |
+-------------------------------------------------------------------------------------------+
| CreateTime:               2021-07-07 15:29:53                                             |
| LastDDLTime:              2021-07-07 15:29:53                                             |
| LastModifiedTime:         2021-07-07 15:29:53                                             |
| Lifecycle:                10                                                              |
+-------------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                         |
+-------------------------------------------------------------------------------------------+
| Native Columns:                                                                           |
+-------------------------------------------------------------------------------------------+
| Field    | Type   | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+-------------------------------------------------------------------------------------------+
| shop_name | string |       |               | true     | NULL         |              |
| customer_id | string |       |               | true     | NULL         |              |
| total_price | double |       |               | true     | NULL         |              |
| sale_date | string |       |               | true     | NULL         |              |
| region    | string |       |               | true     | NULL         |              |
+-------------------------------------------------------------------------------------------+
| Extended Info:                                                                            |
+-------------------------------------------------------------------------------------------+
| TableID:                  98cb8a38733c49eabed4735173818147                                |
| IsArchived:               false                                                           |
| PhysicalSize:             0                                                               |
| FileNum:                  0                                                               |
| StoredAs:                 AliOrc                                                          |
| CompressionStrategy:      normal                                                          |
+-------------------------------------------------------------------------------------------+
```

sale_date和region两个字段仅会作为普通列存在，而不是表的分区。

- 示例4：查看创建的sale_detail_ctas2表的信息。

```
desc sale_detail_ctas2;
```

返回结果如下。

```
+------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name           |
| TableComment:                                                          |
+------------------------------------------------------------------------+
| CreateTime:               2017-06-28 15:42:17                          |
| LastDDLTime:              2017-06-28 15:42:17                          |
| LastModifiedTime:         2017-06-28 15:42:17                          |
+------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                      |
+------------------------------------------------------------------------+
| Native Columns:                                                        |
+------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                         |
+------------------------------------------------------------------------+
| shop_name       | string     |       |                                 |
| customer_id     | string     |       |                                 |
| total_price     | double     |       |                                 |
| sale_date       | string     |       |                                 |
| region          | string     |       |                                 |
+------------------------------------------------------------------------+
```

- 示例5：查看创建的sale_detail_like表的详细信息。

```
desc extended sale_detail_like;
```

返回结果如下。

```
+-------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                        |
| TableComment:                                                                       |
+-------------------------------------------------------------------------------------+
| CreateTime:               2021-07-07 15:40:38                                       |
| LastDDLTime:              2021-07-07 15:40:38                                       |
| LastModifiedTime:         2021-07-07 15:40:38                                       |
| Lifecycle:                10                                                        |
+-------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                   |
+-------------------------------------------------------------------------------------+
| Native Columns:                                                                     |
+-------------------------------------------------------------------------------------+
| Field    | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+-------------------------------------------------------------------------------------+
| shop_name | string |       |               | true     | NULL         |              |
| customer_id | string |       |               | true     | NULL         |              |
| total_price | double |       |               | true     | NULL         |              |
+-------------------------------------------------------------------------------------+
| Partition Columns:                                                                  |
+-------------------------------------------------------------------------------------+
| sale_date       | string     |                                                      |
| region          | string     |                                                      |
+-------------------------------------------------------------------------------------+
| Extended Info:                                                                      |
+-------------------------------------------------------------------------------------+
| TableID:                  61782ff7713f426e9d6f91d5deeac99a                          |
| IsArchived:               false                                                     |
| PhysicalSize:             0                                                         |
| FileNum:                  0                                                         |
| StoredAs:                 AliOrc                                                    |
| CompressionStrategy:      normal                                                    |
+-------------------------------------------------------------------------------------+
```

除生命周期属性外，sale_detail_like的其它属性（字段类型、分区类型等）均与sale_detail完全一致。

> **说明** 通过 `desc table_name` 查看到的Size包含了在回收站的数据Size。如果您需要清空回收站，可以先执行 `purge table table_name` ，然后再执行 `desc table_name` 查看除回收站以外的数据大小。您也可以执行 `show recyclebin` 查看本项目中回收站内的数据明细。

- 示例6：查看创建的test_newtype表的信息。

```
desc test_newtype;
```

返回结果如下。

```
| Native Columns:                                                                        |
+----------------------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                                         |
+----------------------------------------------------------------------------------------+
| c1              | tinyint    |       |                                                 |
| c2              | smallint   |       |                                                 |
| c3              | int        |       |                                                 |
| c4              | bigint     |       |                                                 |
| c5              | float      |       |                                                 |
| c6              | double     |       |                                                 |
| c7              | decimal    |       |                                                 |
| c8              | binary     |       |                                                 |
| c9              | timestamp  |       |                                                 |
| c10             | array<map<bigint,bigint>> |       |                                  |
| c11             | map<string,array<bigint>> |       |                                  |
| c12             | struct<s1:string,s2:bigint> |       |                                |
| c13             | varchar(20) |       |                                                |
+----------------------------------------------------------------------------------------+
OK
```

- 示例7：查看创建的Hash聚簇非分区表t1的信息。聚簇属性将显示在Extended Info中。

```
desc extended t1;
```

返回结果如下。

```
+----------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                           |
| TableComment:                                                                          |
+----------------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 18:00:56                                          |
| LastDDLTime:              2020-11-16 18:00:56                                          |
| LastModifiedTime:         2020-11-16 18:00:56                                          |
+----------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                      |
+----------------------------------------------------------------------------------------+
| Native Columns:                                                                        |
+----------------------------------------------------------------------------------------+
| Field    | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+----------------------------------------------------------------------------------------+
| a        | string |       |               | true     | NULL         |              |
| b        | string |       |               | true     | NULL         |              |
| c        | bigint |       |               | true     | NULL         |              |
+----------------------------------------------------------------------------------------+
| Extended Info:                                                                         |
+----------------------------------------------------------------------------------------+
| TableID:                  e6b06f705dc34a36a5b72e5af486cab7                             |
| IsArchived:               false                                                        |
| PhysicalSize:             0                                                            |
| FileNum:                  0                                                            |
| StoredAs:                 AliOrc                                                       |
| CompressionStrategy:      normal                                                       |
| ClusterType:              hash                                                         |
| BucketNum:                1024                                                         |
| ClusterColumns:           [c]                                                          |
| SortColumns:              [c ASC]                                                      |
+----------------------------------------------------------------------------------------+
OK
```

- 示例8：查看创建的Hash聚簇分区表t2的信息。聚簇属性将显示在Extended Info中。

```
desc extended t2;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                       |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime: 2017-12-25 11:18:26                                                    |
| LastDDLTime: 2017-12-25 11:18:26                                                   |
| LastModifiedTime: 2017-12-25 11:18:26                                              |
| Lifecycle: 2                                                                       |
+------------------------------------------------------------------------------------+
| InternalTable: YES | Size: 0                                                       |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field | Type   | Label | Comment                                                   |
+------------------------------------------------------------------------------------+
| a     | string |       |                                                           |
| b     | string |       |                                                           |
| c     | bigint |       |                                                           |
+------------------------------------------------------------------------------------+
| Partition Columns:                                                                 |
+------------------------------------------------------------------------------------+
| dt    | string |                                                                   |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
| TableID: 91a3395d3ef64b4d9ee1d2852755                                              |
| IsArchived: false                                                                  |
| PhysicalSize: 0                                                                    |
| FileNum: 0                                                                         |
| ClusterType: hash                                                                  |
| BucketNum: 1024                                                                    |
| ClusterColumns: [c]                                                                |
| SortColumns: [c ASC]                                                               |
+------------------------------------------------------------------------------------+
OK
```

- 示例9：查看Range聚簇非分区表t3的信息。聚簇属性将显示在Extended Info中。

```
desc extended t3;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                       |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 18:01:05                                      |
| LastDDLTime:              2020-11-16 18:01:05                                      |
| LastModifiedTime:         2020-11-16 18:01:05                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field    | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+------------------------------------------------------------------------------------+
| a        | string |       |               | true     | NULL         |              |
| b        | string |       |               | true     | NULL         |              |
| c        | bigint |       |               | true     | NULL         |              |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
| TableID:                  38d170aca2684f4baadbbe1931a6ae1f                         |
| IsArchived:               false                                                    |
| PhysicalSize:             0                                                        |
| FileNum:                  0                                                        |
| StoredAs:                 AliOrc                                                   |
| CompressionStrategy:      normal                                                   |
| ClusterType:              range                                                    |
| BucketNum:                1024                                                     |
| ClusterColumns:           [c]                                                      |
| SortColumns:              [c ASC]                                                  |
+------------------------------------------------------------------------------------+
OK
```

- 示例10：查看Range聚簇分区表t4的信息。聚簇属性将显示在Extended Info中。

```
desc extended t4;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                       |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 19:17:48                                      |
| LastDDLTime:              2020-11-16 19:17:48                                      |
| LastModifiedTime:         2020-11-16 19:17:48                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field    | Type   | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+------------------------------------------------------------------------------------+
| a        | string |       |               | true     | NULL         |              |
| b        | string |       |               | true     | NULL         |              |
| c        | bigint |       |               | true     | NULL         |              |
+------------------------------------------------------------------------------------+
| Partition Columns:                                                                 |
+------------------------------------------------------------------------------------+
| dt              | string     |                                                     |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
| TableID:                  6ebc3432e283449188c861427bcd6ee4                         |
| IsArchived:               false                                                    |
| PhysicalSize:             0                                                        |
| FileNum:                  0                                                        |
| StoredAs:                 AliOrc                                                   |
| CompressionStrategy:      normal                                                   |
| ClusterType:              range                                                    |
| BucketNum:                0                                                        |
| ClusterColumns:           [c]                                                      |
| SortColumns:              [c ASC]                                                  |
+------------------------------------------------------------------------------------+
OK
```

- 示例11：查看非分区表t5是否为Transactional表。

> **说明** 推荐您使用MaxCompute客户端查看表是否为Transactional表，需要将MaxCompute客户端升级到0.35.4版本。其他工具可能会存在版本升级未就绪的问题，导致查询结果不显示Transactional信息。

```
desc extended t5;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@aliyun.com | Project: $project_name                            |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2021-02-18 10:56:27                                      |
| LastDDLTime:              2021-02-18 10:56:27                                      |
| LastModifiedTime:         2021-02-18 10:56:27                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field    | Type   | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+------------------------------------------------------------------------------------+
| id       | bigint |       |               | true     | NULL         |              |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
...
| Transactional:            true                                                     |
+------------------------------------------------------------------------------------+
```

- 示例12：查看分区表t6是否为Transactional表。

  > **说明** 推荐您使用MaxCompute客户端查看表是否为Transactional表，需要将MaxCompute客户端升级到0.35.4版本。其他工具可能会存在版本升级未就绪的问题，导致查询结果不显示Transactional信息。

  ```
  desc extended t6;
  ```

  返回结果如下。

  ```
  +------------------------------------------------------------------------------------+
  | Owner: ALIYUN$xxxxx@test.aliyunid.com | Project: $project_name                      |
  | TableComment:                                                                      |
  +------------------------------------------------------------------------------------+
  | CreateTime:               2021-02-18 15:34:54                                      |
  | LastDDLTime:              2021-02-18 15:34:54                                      |
  | LastModifiedTime:         2021-02-18 15:34:54                                      |
  +------------------------------------------------------------------------------------+
  | InternalTable: YES      | Size: 0                                                  |
  +------------------------------------------------------------------------------------+
  | Native Columns:                                                                    |
  +------------------------------------------------------------------------------------+
  | Field           | Type       | Label | Comment                                     |
  +------------------------------------------------------------------------------------+
  | id              | bigint     |       |                                             |
  +------------------------------------------------------------------------------------+
  | Partition Columns:                                                                 |
  +------------------------------------------------------------------------------------+
  | ds              | string     |                                                     |
  +------------------------------------------------------------------------------------+
  | Extended Info:                                                                     |
  +------------------------------------------------------------------------------------+
  ...
  | Transactional:            true                                                     |
  +------------------------------------------------------------------------------------+
  ```

查看分区信息

查看某个分区表具体的分区的信息。

- 命令格式

  ```
  desc <table_name> partition (<pt_spec>);
  ```

- 参数说明
  - table_name：必填。待查看分区信息的分区表名称。
  - pt_spec：必填。待查看的分区信息。格式为 `partition_col1=col1_value1, partition_col2=col2_value1...` 。对于有多级分区的表，必须指明全部的分区值。
- 使用示例

  ```
  --查询分区表sale_detail的分区信息。
  desc sale_detail partition (sale_date='201310',region='beijing');
  ```

  返回结果如下。

  ```
  +------------------------------------------------------------------------------------+
  | PartitionSize: 2109112                                                             |
  +------------------------------------------------------------------------------------+
  | CreateTime:               2015-10-10 08:48:48                                      |
  | LastDDLTime:              2015-10-10 08:48:48                                      |
  | LastModifiedTime:         2015-10-11 01:33:35                                      |
  +------------------------------------------------------------------------------------+
  OK
  ```

列出项目下的表和视图

列出项目下所有的表和视图，或符合某规则的表和视图。

- 命令格式

  ```
  --列出项目下所有的表和视图。
  show tables;
  --列出项目下表名或视图名与chart匹配的表。
  show tables like '<chart>';
  ```

- 使用示例

```
--列出项目下表名与sale*匹配的表。*表示任意字段。
show tables like 'sale*';
```

返回结果如下。

```
ALIYUN$account_name:sale_detail
......
--ALIYUN是系统提示符，表示您是阿里云主账号用户。如果您是阿里云RAM用户，系统提示符为RAM。
```

列出所有分区

列出一张表中的所有分区，表不存在或为非分区表时，返回报错。

- 命令格式

```
show partitions <table_name>;
```

- 参数说明

  table_name：必填。待查看分区信息的分区表名称。

- 使用示例

```
--列出sale_detail中的所有分区。
show partitions sale_detail;
```

返回结果如下。

```
sale_date=201310/region=beijing
sale_date=201312/region=shenzhen
sale_date=201312/region=xian
sale_date=2014/region=shenzhen
OK
```

2.8. 分区和列操作

分区和列操作为您提供了变更MaxCompute中表的分区或列的操作方法，您可以根据实际业务场景执行相应操作。本文为您介绍如何添加或删除分区、添加或修改列及注释常用操作。

分区和列操作详情请参见分区和列操作。分区和列操作常用命令如下。

类型	功能	角色	操作入口
添加分区	为已存在的分区表新增分区。	具备修改表权限（Alter）的用户	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
删除分区	为已存在的分区表删除分区。		
添加列名或注释	为已存在的非分区表或分区表添加列或注释。		
修改列名或注释	修改非分区表或分区表的列名或注释。		

添加分区

为已存在的分区表新增分区。

- 限制条件
  - MaxCompute单表支持的分区数量上限为6万个。
  - 对于有多级分区的表，如果需要添加新的分区值，必须指明全部的分区。
  - 仅支持新增分区值，不支持新增分区字段。
- 命令格式

```
alter table <table_name> add [if not exists] partition <pt_spec> [partition <pt_spec> partition <pt_spec>...];
```

- 参数说明
  - table_name：必填。待新增分区的分区表名称。
  - if not exists：可选。如果未指定if not exists而同名的分区已存在，会执行失败并返回报错。
  - pt_spec：必填。新增的分区，格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。分区字段不区分大小写，分区值区分大小写。
- 使用示例
  - 示例1：给表sale_detail添加一个分区，用来存储2013年12月杭州地区的销售记录。

```
alter table sale_detail add if not exists partition (sale_date='201312', region='hangzhou');
```

- 示例2：给表sale_detail同时添加两个分区，用来存储2013年12月北京和上海地区的销售记录。

```
alter table sale_detail add if not exists partition (sale_date='201312', region='beijing') partition (sale_date='201312', re
gion='shanghai');
```

- 示例3：给表sale_detail添加分区，仅指定一个分区字段sale_date，返回报错，需要同时指定2个分区字段sale_date和region。

```
alter table sale_detail add if not exists partition (sale_date='20111011');
```

删除分区

为已存在的分区表删除分区。

MaxCompute支持通过条件筛选方式删除分区。如果您希望一次性删除符合某个规则条件的多个分区，可以使用表达式指定筛选条件，通过筛选条件匹配分区并批量删除分区。

- 限制条件
  - 每个分区过滤子句只能访问一个分区列。
  - 表达式用到的函数必须是内建的Scalar函数。
- 注意事项
  - 删除分区之后，MaxCompute项目的存储量会降低。
  - 您可以结合MaxCompute提供的生命周期功能，实现自动回收旧分区的能力。更多生命周期信息，请参见生命周期。
- 命令格式
  - 未指定筛选条件

```
--一次删除一个分区。
alter table <table_name> drop [if exists] partition <pt_spec>;
--一次删除多个分区。
alter table <table_name> drop [if exists] partition <pt_spec>,partition <pt_spec>[,partition <pt_spec>....];
```

  - 指定筛选条件

```
alter table <table_name> drop [if exists] partition <partition_filtercondition>;
```

- 参数说明
  - table_name：必填。待删除分区的分区表名称。
  - if exists：可选。如果未指定if exists且分区不存在，则返回报错。
  - pec：必填。删除的分区。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。分区字段不区分大小写，分区值区分大小写。
  - partition_filtercondition：指定筛选条件时必填。分区筛选条件，不区分大小写。格式为：

```
partition_filtercondition
    : partition (<partition_col> <relational_operators> <partition_col_value>)
    | partition (scalar(<partition_col>) <relational_operators> <partition_col_value>)
    | partition (<partition_filtercondition1> AND|OR <partition_filtercondition2>)
    | partition (NOT <partition_filtercondition>)
    | partition (<partition_filtercondition1>)[,partition (<partition_filtercondition2>), ...]
```

    - partition_col：分区名称。
    - relational_operators：关系运算符，详情请参见运算符。
    - partition_col_value：分区列比较值或正则表达式，与分区列数据类型保持一致。
    - scalar()：Scalar函数。Scalar函数基于输入值生成对应的标量，对分区列的值（partition_col）进行处理后再按照指定的关系运算符relational_operators与partition_col_value做比较。
    - 分区过滤条件支持逻辑运算符NOT、AND和OR。支持通过NOT过滤条件子句，取过滤规则的补集。支持多个过滤条件子句以AND或OR的关系组成整体分区匹配规则。
    - 支持多个分区过滤子句，当多个分区过滤子句以英文逗号（,）分隔时，每个过滤子句的逻辑以OR的关系组成整体分区匹配规则。
- 使用示例
  - 未指定筛选条件

```
--从表sale_detail中删除一个分区，2013年12月杭州分区的销售记录。
alter table sale_detail drop if exists partition(sale_date='201312',region='hangzhou');
--从表sale_detail中同时删除两个分区，2013年12月杭州和上海分区的销售记录。
alter table sale_detail drop if exists partition(sale_date='201312',region='hangzhou'),partition(sale_date='201312',region='
shanghai');
```

- 指定筛选条件

```
--创建分区表。
create table if not exists sale_detail(
shop_name     STRING,
customer_id   STRING,
total_price   DOUBLE)
partitioned by (sale_date STRING);
--添加分区。
alter table sale_detail add if not exists
partition (sale_date= '201910')
partition (sale_date= '201911')
partition (sale_date= '201912')
partition (sale_date= '202001')
partition (sale_date= '202002')
partition (sale_date= '202003')
partition (sale_date= '202004')
partition (sale_date= '202005')
partition (sale_date= '202006')
partition (sale_date= '202007');
--批量删除分区。
alter table sale_detail drop if exists partition(sale_date < '201911');
alter table sale_detail drop if exists partition(sale_date >= '202007');
alter table sale_detail drop if exists partition(sale_date LIKE '20191%');
alter table sale_detail drop if exists partition(sale_date IN ('202002','202004','202006'));
alter table sale_detail drop if exists partition(sale_date BETWEEN '202001' AND '202007');
alter table sale_detail drop if exists partition(substr(sale_date, 1, 4) = '2020');
alter table sale_detail drop if exists partition(sale_date < '201912' OR sale_date >= '202006');
alter table sale_detail drop if exists partition(sale_date > '201912' AND sale_date <= '202004');
alter table sale_detail drop if exists partition(NOT sale_date > '202004');
--支持多个分区过滤表达式，表达式之间是OR的关系。
alter table sale_detail drop if exists partition(sale_date < '201911'), partition(sale_date >= '202007');
--添加其他格式分区。
alter table sale_detail add IF NOT EXISTS
partition (sale_date= '2019-10-05')
partition (sale_date= '2019-10-06')
partition (sale_date= '2019-10-07');
--批量删除分区，使用正则表达式匹配分区。
alter table sale_detail drop if exists partition(sale_date RLIKE '2019-\\d+-\\d+');
--创建多级分区表。
create table if not exists region_sale_detail(
shop_name     STRING,
customer_id   STRING,
total_price   DOUBLE)
partitioned by (sale_date STRING , region STRING );
--添加分区。
alter table region_sale_detail add IF NOT EXISTS
partition (sale_date= '201910',region = 'shanghai')
partition (sale_date= '201911',region = 'shanghai')
partition (sale_date= '201912',region = 'shanghai')
partition (sale_date= '202001',region = 'shanghai')
partition (sale_date= '202002',region = 'shanghai')
partition (sale_date= '201910',region = 'beijing')
partition (sale_date= '201911',region = 'beijing')
partition (sale_date= '201912',region = 'beijing')
partition (sale_date= '202001',region = 'beijing')
partition (sale_date= '202002',region = 'beijing');
--执行如下语句批量删除多级分区，两个匹配条件是或的关系，会将sale_date小于201911或region等于beijing的分区都删除掉。
alter table region_sale_detail drop if exists partition(sale_date < '201911'),partition(region = 'beijing');
--如果删除sale_date小于201911且region等于beijing的分区，可以使用如下方法。
alter table region_sale_detail drop if exists partition(sale_date < '201911', region = 'beijing');
```

批量删除多级分区时，在一个 `partition` 过滤子句中，不能根据多个分区列编写组合条件匹配分区，如下语句会报错 `FAILED: ODPS-0130071: [1,82] Semantic analysis exception - invalid column reference region, partition expression must have one and only one column reference` 。

```
--分区过滤子句只能访问一个分区列，如下语句报错。
alter table region_sale_detail drop if exists partition(sale_date < '201911' AND region = 'beijing');
```

添加列或注释

为已存在的非分区表或分区表添加列或注释。

> 说明 MaxCompute已支持添加STRUCT类型的列，例如 `struct<x: string, y: bigint>` 、 `map<string, struct<x: double, y: double>>` 。该功能正在试用中，试用操作请参见新功能测试申请。

- 命令格式

```
alter table <table_name> add columns (<col_name1> <type1> comment ['<col_comment>'][, <col_name2> <type2> comment '<col_comment>'...]);
```

- 参数说明
  - table_name：必填。待新增列的表名称。添加的新列不支持指定顺序，默认在最后一列。
  - col_name：必填。新增列的名称。
  - type：必填。新增列的数据类型。
  - col_comment：可选。新增列的注释。
- 使用示例
  - 示例1：给表sale_detail添加两个列。

```
alter table sale_detail add columns (customer_name STRING, education BIGINT);
```

  - 示例2：给表sale_detail添加两个列并同时添加列注释。

```
alter table sale_detail add columns (customer_name STRING comment '客户', education BIGINT comment '教育' );
```

  - 示例3：给表sale_detail添加一个复杂数据类型列。

```
alter table sale_detail add columns (region struct<province:string, area:string>);
```

修改列名及注释

修改非分区表或分区表的列名或注释。

- 命令格式

```
alter table <table_name> change column <old_col_name> <new_col_name> <column_type> comment '<col_comment>';
```

- 参数说明
  - table_name：必填。需要修改列名以及注释的表名称。
  - old_col_name：必填。需要修改的列名称。 `old_col_name` 必须是已存在的列。
  - new_col_name：必填。新的列名称。表中不能有名为 `new_col_name` 的列。
  - column_type：必填。列的数据类型。
  - col_comment：可选。修改后的注释信息。内容最长为1024字节。
- 使用示例

```
--修改表sale_detail的列名customer_name为customer_newname，注释“客户”为“customer”。
alter table sale_detail change column customer_name customer_newname STRING comment 'customer';
```

2.9. 实例操作

通过MaxCompute提交的SQL、Spark和MapReduce等作业会实例化，以MaxCompute实例（Instance）的形式存在。本文为您介绍如何查看指定的实例信息和实例状态、停止实例以及获得实例运行日志信息等常用实例操作。

MaxCompute实例有对应的实例ID，实例ID永久有效，且全局唯一。实例操作常用命令如下。

类型	功能	角色	操作入口
查看实例信息	查看实例信息。	具备查看实例权限（Read）的用户或具备项目查看对象列表权限（List）的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
查看实例状态	查看指定实例的状态。	具备查看实例权限（Read）的用户。	
查看正在执行的实例信息	查看项目中正在执行的实例信息。	具备查看实例权限（Read）的用户或具备项目查看对象列表权限（List）的用户。	
停止实例	停止指定的处于Running状态的实例。	具备更新实例权限（Write）的用户。	
获取实例对应的作业信息	根据具体的实例ID获取作业信息。	具备查看实例权限（Read）的用户。	
获取实例对应的作业运行日志信息	根据具体的实例ID获取作业的运行日志信息。		

查看实例信息

查看实例信息，包括StartTime（时间精确到秒）、RunTime（s）、Status（实例状态）、InstanceID、实例所有者、Query（执行语句）。

- 命令格式

```
show p|proc|processlist|instances [from <startdate>] [to <enddate>] [-p <project_name>] [-limit <number> | <number>] [-all];
ls|list instances [from <startdate>] [to <enddate>] [-p <project_name>] [-limit <number> | <number>] [-all];
```

`show p`、`show proc`、`show processlist`、`show instances`、`ls instances`和`list instances`命令等效。

- 参数说明
  - startdate、enddate：可选。返回从起始时间startdate到结束时间enddate期间用户提交的实例信息，startdate要小于enddate。不包含enddate当天提交的实例信息。格式必须为`yyyy-mm-dd`，精度为天。如果不指定，则返回用户当天提交的实例。
  - project_name：可选。待查询实例所属的MaxCompute项目名称，需要具备目标MaxCompute项目的查看实例权限。如果不指定，则表示查询当前所处的MaxCompute项目。
  - number：可选。指定返回实例的数量。
  - 依照时间排序，返回提交时间距离当前时间最近的number个实例信息。如果不指定number，则返回满足要求的所有实例信息。`-limit <number>`与`number`二者等效。
  - -all：可选。返回MaxCompute项目中所有成员执行过的实例。如果不指定，则表示返回MaxCompute项目中当前用户执行过的实例。

    不配置number时，默认最大返回50条；如果配置了number，返回number个实例信息。

- 使用示例
  - 示例1：查看当前MaxCompute项目下当前用户当天执行的所有实例信息。命令示例如下。

    ```
    show p;
    ```

    返回结果如下。

    ```
    StartTime             RunTime  Status   InstanceID                Owner                          Query
    2021-09-14 11:43:04   0s       Success  20210914**************3rw2  ALIYUN$****@test.aliyunid.com
    2021-09-14 11:43:05   1s       Success  20210914**************5t32  ALIYUN$****@test.aliyunid.com select date_sub(datetime
    '2005-03-01 00:00:00', 1);
    2021-09-14 11:58:13   0s       Success  20210914**************5pr2  ALIYUN$****@test.aliyunid.com
    2021-09-14 11:58:15   1s       Success  20210914**************5qgr  ALIYUN$****@test.aliyunid.com select date_sub(date '20
    05-02-28', -1);
    2021-09-14 12:02:15   1s       Success  20210914**************h8o7  ALIYUN$****@test.aliyunid.com select date_sub('2008-03
    -01 00:00:00', 2);
    2021-09-14 12:02:15   0s       Success  20210914**************5t32  ALIYUN$****@test.aliyunid.com
    2021-09-14 12:02:31   0s       Success  20210914**************5pr2  ALIYUN$****@test.aliyunid.com
    2021-09-14 12:02:32   0s       Success  20210914**************euq2  ALIYUN$****@test.aliyunid.com select date_sub('2005-03
    -01 00:00:00', 2);
    2021-09-14 13:35:42   0s       Success  20210914**************1ms2  ALIYUN$****@test.aliyunid.com
    2021-09-14 13:35:43   0s       Success  20210914**************j8o7  ALIYUN$****@test.aliyunid.com select date_sub(getdate(
    ),1);
    2021-09-14 13:40:40   1s       Success  20210914**************h3wz  ALIYUN$****@test.aliyunid.com select date_sub(getdate(
    ),0);
    2021-09-14 13:40:40   0s       Success  20210914**************9nm7  ALIYUN$****@test.aliyunid.com
    12 instances
    ```

  - 示例2：查看当前MaxCompute项目下当前用户在指定时间段内执行的实例，并指定返回实例数量。命令示例如下。

    ```
    show instances from 2021-09-14 to 2021-09-15 -limit 10;
    ```

    返回结果如下。

    ```
    StartTime             RunTime  Status   InstanceID                Owner                          Query
    2021-09-14 11:58:13   0s       Success  20210914**************5pr2  ALIYUN$****@test.aliyunid.com
    2021-09-14 11:58:15   1s       Success  20210914**************5qgr  ALIYUN$****@test.aliyunid.com    select date_sub(date
    '2005-02-28', -1);
    2021-09-14 12:02:15   1s       Success  20210914**************h8o7  ALIYUN$****@test.aliyunid.com    select date_sub('2008
    -03-01 00:00:00', 2);
    2021-09-14 12:02:15   0s       Success  20210914**************5t32  ALIYUN$****@test.aliyunid.com
    2021-09-14 12:02:31   0s       Success  20210914**************5pr2  ALIYUN$****@test.aliyunid.com
    2021-09-14 12:02:32   0s       Success  20210914**************euq2  ALIYUN$****@test.aliyunid.com    select date_sub('2005
    -03-01 00:00:00', 2);
    2021-09-14 13:35:42   0s       Success  20210914**************1ms2  ALIYUN$****@test.aliyunid.com
    2021-09-14 13:35:43   0s       Success  20210914**************j8o7  ALIYUN$****@test.aliyunid.com    select date_sub(getda
    te(),1);
    2021-09-14 13:40:40   1s       Success  20210914**************h3wz  ALIYUN$****@test.aliyunid.com    select date_sub(getda
    te(),0);
    2021-09-14 13:40:40   0s       Success  20210914**************9nm7  ALIYUN$****@test.aliyunid.com
    10 instances
    ```

- 示例3：查看另一MaxCompute项目下所有用户在指定时间段内执行的实例，并指定返回实例数量。命令示例如下。

```
ls instances from 2021-09-14 to 2021-09-15 -p doc_test_dev -all -limit 10;
```

返回结果如下。

```
StartTime             RunTime  Status   InstanceID                  Owner                          Query
2021-09-14 11:59:16   0s       Success  20210914**************6pr3   ALIYUN$****@test.aliyunid.com
2021-09-14 11:59:20   1s       Success  20210914**************6qgr   ALIYUN$****@test.aliyunid.com   select date_sub(date
'2007-02-26', -1);
2021-09-14 12:02:19   1s       Success  20210914**************h8o7   ALIYUN$****@test.aliyunid.com   select date_sub('2009
-03-01 00:00:00', 2);
2021-09-14 12:02:25   0s       Success  20210914**************7t42   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:37   0s       Success  20210914**************7pr2   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:40   0s       Success  20210914**************emq2   ALIYUN$****@test.aliyunid.com   select date_sub('2015
-03-01 00:00:00', 2);
2021-09-14 13:35:42   0s       Success  20210914**************1ms2   ALIYUN$****@test.aliyunid.com
2021-09-14 13:35:43   0s       Success  20210914**************68o7   ALIYUN$kiki                     select date_sub(getda
te(),1);
2021-09-14 13:45:40   1s       Success  20210914**************73wz   ALIYUN$kiki                     select date_sub(getda
te(),0);
2021-09-14 13:45:45   0s       Success  20210914**************9nm7   ALIYUN$dreak
10 instances
```

- 示例4：查看另一MaxCompute项目下所有用户在当天执行的实例，并指定返回实例数量。命令示例如下。

```
show p -p doc_test_dev -all 5;
```

返回结果如下。

```
StartTime             RunTime  Status   InstanceID                  Owner                          Query
2021-09-14 12:02:40   0s       Success  20210914**************emq2   ALIYUN$****@test.aliyunid.com   select date_sub('2015
-03-01 00:00:00', 2);
2021-09-14 13:35:42   0s       Success  20210914**************1ms2   ALIYUN$****@test.aliyunid.com
2021-09-14 13:35:43   0s       Success  20210914**************68o7   ALIYUN$kiki                     select date_sub(getda
te(),1);
2021-09-14 13:45:40   1s       Success  20210914**************73wz   ALIYUN$kiki                     select date_sub(getda
te(),0);
2021-09-14 13:45:45   0s       Success  20210914**************9nm7   ALIYUN$dreak
5 instances
```

查看实例状态

查看指定实例的状态。如果此实例并非当前用户创建，则返回报错。

- 命令格式

```
status <instance_id>;
```

- 参数说明

  instance_id：必填。实例的唯一标识符，指定查询哪个实例状态。

- 返回结果

  实例有以下6种状态：

  - Running：正在运行。
  - Success：成功结束。
  - Waiting：等待中。
  - Failed：作业失败，但是尚未改写目标表数据。
  - Cancelled：被取消。
  - Terminated：作业已执行结束。

- 使用示例

```
--查看ID为20131225123xxxxxxxxxxxxxx的实例状态。
status 20131225123xxxxxxxxxxxxxx;
```

返回结果如下。

```
Success
```

查看正在执行的实例信息

查看项目中正在执行的实例信息，包括InstanceID、Owner、Type、StartTime、Progress、Status、Priority、RuntimeUsage（CPU/MEM）、totalUsage（CPU/MEM）、QueueingInfo（POS/LEN）等。

- 命令格式

```
--查看当前项目中当前账号所提交的正在执行的实例信息。
top instance;
--返回当前项目下所有或指定数量的正在执行的实例信息。
top instance [-all][-limit <number>];
```

- 参数说明
  - -all：可选。返回当前项目下所有正在执行的作业，默认最大返回50条。
  - -limit number：可选。返回当前项目下指定数量的正在执行的作业。
- 使用示例

```
--返回当前项目下前5条正在执行的实例信息。
top instance -limit 5;
```

返回结果如下。

```
QueueingInstances: 0 total.
```

停止实例

停止指定的实例，实例的状态必须为Running。

> 说明 此命令是一个异步的过程，在系统接受此请求并返回结果后，并不意味着分布式的作业已经停止，只能代表系统已接收到请求，因此还要执行 `STATUS` 命令，查看此Instance的状态后才可以确定是否已停止实例。

- 命令格式

```
kill <instance_id>;
```

- 参数说明

  instance_id：实例的唯一标识符。状态必须是Running的实例，否则返回异常。
- 使用示例

```
--停止ID为20131225123xxxxxxxxxxxxxx的实例。
kill 20131225123xxxxxxxxxxxxxx;
```

获取实例对应的作业信息

根据具体的实例ID获取作业信息，包括具体的SQL、Owner、StartTime、EndTime和Status等信息。

- 命令格式

```
desc instance <instance_id>;
```

- 参数说明

  instance_id：实例的唯一标识符。
- 使用示例

```
--查询ID为20150715xxxxxxxxxxxxxx的实例对应的作业信息。
desc instance 20150715xxxxxxxxxxxxxx;
```

返回结果如下。

```
ID                                  20150715xxxxxxxxxxxxxx
Owner                               ALIYUN$XXXXXX@alibaba-inc.com
StartTime                           2015-07-15 18:34:41
EndTime                             2015-07-15 18:34:42
Status                              Terminated
console_select_query_task_1436956481295 Success
Query                               select * from mj_test;
```

获取实例对应的作业运行日志信息

根据具体的实例ID获取作业的运行日志信息，包含Logview链接。您可以通过查看Logview获得作业的详细日志。

> 说明 获取最近3天Instance对应的Logview可正常打开。超过3天的Instance对应Logview可获取，但因为过期被清除而无法打开。

- 命令格式

```
wait <instance_id>;
```

- 参数说明

  instance_id：实例的唯一标识符。

- 使用示例

```
--查看实例ID为20170925161122379gxxxxxx的作业运行日志信息。
wait 20170925161122379gxxxxxx;
```

返回结果如下。

```
ID = 20170925161122379g3xxxxxx
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=alian&i=20170925161122xxxxxxdqp&token=XXXXXXvbiI6IjEifQ==
Job Queueing...
Summary:
resource cost: cpu 0.05 Core * Min, memory 0.05 GB * Min
inputs:
        alian.bank_data: 41187 (588232 bytes)
outputs:
        alian.result_table: 8 (640 bytes)
Job run time: 2.000
Job run mode: service job
Job run engine: execution engine
M1:
        instance count: 1
        run time: 1.000
        instance time:
                min: 1.000, max: 1.000, avg: 1.000
        input records:
                TableScan_REL5213301: 41187  (min: 41187, max: 41187, avg: 41187
)
        output records:
                StreamLineWrite_REL5213305: 8  (min: 8, max: 8, avg: 8)
R2_1:
        instance count: 1
        run time: 2.000
        instance time:
                min: 2.000, max: 2.000, avg: 2.000
        input records:
                StreamLineRead_REL5213306: 8  (min: 8, max: 8, avg: 8)
        output records:
                TableSink_REL5213309: 8  (min: 8, max: 8, avg: 8)
```

2.10. 资源操作

资源（Resource）是MaxCompute的特有概念，如果您想使用MaxCompute的自定义函数（UDF）或MapReduce功能，需要依赖资源来完成。本文为您介绍添加、查看、下载及删除资源等常用资源操作。

资源操作常用命令如下。

类型	功能	角色	操作入口
添加资源	添加资源至MaxCompute项目中。	具备更新资源权限（Write）的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
查看资源信息	查看资源的详细信息。	具备读取资源权限（Read）的用户。	
查看资源列表	查看当前项目下所有的资源。	具备项目查看对象列表权限（List）的用户。	
为资源创建别名	为资源创建别名。	具备更新资源权限（Write）的用户。	
下载资源	下载MaxCompute项目中的资源到本地。	具备更新资源权限（Write）的用户。	
删除资源	删除MaxCompute项目中已经存在的资源。	具备删除资源权限（Delete）的用户。	

添加资源

添加资源至MaxCompute项目中。

- 使用限制
  - MaxCompute不支持添加外部表为资源。
  - 每个资源文件的大小不能超过500 MB。单个SQL、MapReduce作业所引用的资源总大小不能超过2048 MB。
- 命令格式

```
add file <local_file> [as <alias>] [comment '<comment>'][-f];
add archive <local_file> [as <alias>] [comment '<comment>'][-f];
add table <table_name> [partition (<spec>)] [as <alias>] [comment '<comment>'][-f];
add py|jar <localfile> [comment '<comment>'][-f];
```

- 参数说明
  - file|archive|table|py|jar：必填。资源类型。资源类型详情请参见资源。
  - local_file：必填。表示本地文件所在路径。并以文件名作为资源名，资源名是资源的唯一标识。
  - table_name：必填。表示MaxCompute中的表名。
  - spec：必填。当添加的资源为分区表时，MaxCompute仅支持将某个分区作为资源，不支持将整张分区表作为资源。
  - alias：可选。指定资源名，不加该参数时默认文件名为资源名。JAR及PYTHON类型资源不支持此参数。
  - comment：可选。为资源添加注释。
  - -f：可选。当存在同名的资源时，此操作会覆盖原有资源。如果不指定此选项，存在同名资源时，操作将失败。
- 使用示例
  - 示例1：添加一个FILE类型文件资源到MaxCompute。

    ```
    add file banking.txt;
    ```

    返回结果如下。

    ```
    OK: Resource 'banking.txt' have been created.
    ```

  - 示例2：添加一个别名为sale.res的分区表资源到MaxCompute。

    ```
    add table sale_detail partition (ds='20150602') as sale.res comment 'sale detail on 20150602' -f;
    ```

    返回结果如下。

    ```
    OK: Resource 'sale.res' have been updated.
    ```

  - 示例3：添加Python资源到MaxCompute。

    ```
    add py python.py [comment '<comment>'][-f];
    ```

    返回结果如下。

    ```
    OK: Resource 'python.py' have been created.
    ```

查看资源信息

查看当前项目下指定资源的信息，包含资源名、所有者、资源类型、资源大小、创建时间、上一次修改时间、资源文件的MD5值等信息。

- 命令格式

```
desc resource <resource_name>;
```

- 参数说明

  resource_name：必填。已存在的资源名称。
- 返回值说明
  - Name：资源的名称。
  - Owner：资源所有者账号。
  - Type：资源类型。
  - Comment：资源注释信息。
  - CreatedTime：资源创建时间。
  - LastModifiedTime：资源最近一次的更新时间。
  - LastUpdator：执行最近一次更新操作的账号。
  - Size：资源文件大小，单位为MB。
  - Md5sum：资源文件的MD5值。

> **说明** MaxCompute的资源名大小写不敏感，resource_A和 resource_a指代同一资源。

- 使用示例

```
--查看资源topn_new.jar的信息。
desc resource topn_new.jar;
```

返回结果如下。

```
Name                                topn_new.jar
Owner                               ALIYUN$****@test.aliyunid.com
Type                                JAR
Comment                             cloudopenapi
CreatedTime                         2020-12-29 13:55:11
LastModifiedTime                    2020-12-29 13:55:11
LastUpdator
Size                                11438795
Md5sum                              8bcf6aabf****56c0
```

查看资源列表

查看当前项目下所有的资源，包含资源名、所有者、创建时间、最近一次更新时间、资源类型等信息。

- 命令格式

```
list resources;
```

- 使用示例

```
list resources;
```

返回结果如下。

```
Resource Name         Owner                Creation Time         Last Modified Time    Type          Last Updator Resource Si
ze  Source         comment
getaddr.jar           ALIYUN$****          2020-06-18 15:47:28   2020-06-18 15:47:28   jar           1353716
cloudopenapi
ip.dat                ALIYUN$****          2020-06-18 15:49:46   2020-06-18 15:49:46   file          8525962
cloudopenapi
2 resources
```

为资源创建别名

为资源创建别名。alias命令可以在不修改代码的前提下，在MapReduce或自定义函数（UDF）代码中，通过某个固定的资源名读取不同资源（数据）。

- 命令格式

```
alias <alias>=<real>;
```

- 参数说明
  - alias：命名后的资源别名。
  - real：资源的原名。
- 使用示例

```
--新增资源res_20121208和res_20121209。
add table sale_detail partition (ds='20121208') as res_20121208;
add table sale_detail partition (ds='20121209') as res_20121209;
--命名资源res_20121208的别名为resName，并调用它。
alias resName=res_20121208;
jar -resources resName -libjars work.jar  -classpath ./work.jar com.company.MainClass args ...;
--命名资源res_20121209的别名为resName，并调用它。
alias resName=res_20121209;
jar -resources resName -libjars work.jar  -classpath ./work.jar com.company.MainClass args ...;
```

上例中的资源别名 `resName` 在两个作业里引用了不同的资源表，您无需修改代码便可读取到不同的数据。

下载资源

下载MaxCompute项目中的资源到本地。资源类型必须为FILE、JAR、ARCHIVE或PY，不支持Table类型。

- 命令格式

```
get resource <resource_name> <path>;
```

- 参数说明
  - resource_name：必填。要下载的资源名称。
  - path：必填。资源保存到本地的路径。
- 使用示例

```
get resource getaddr.jar D:\;
```

删除资源

删除MaxCompute项目中已经存在的资源。

- 命令格式

```
drop resource <resource_name>;
```

- 参数说明

  resource_name：待删除资源的名称。

- 使用示例

```
drop resource getaddr.jar;
```

返回结果如下。

```
Confirm to "drop resource getaddr.jar" (yes/no)? y
OK
```

2.11. 函数操作

您可以使用系统的内建函数或自定义函数（UDF）完成计算。内建函数可直接调用，自定义函数需要您自定义后才可调用。本文为您介绍如何注册、注销及查看自定义函数等常用函数操作。

函数操作常用命令如下。

类型	功能	角色	操作入口
注册函数	在MaxCompute项目中注册自定义函数。	具备更新函数权限（Write）的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
注销函数	在MaxCompute项目中注销已注册的自定义函数。	具备注销函数权限（Delete）的用户。	
查看函数	查看MaxCompute项目中指定自定义函数的信息。	具备查看函数权限（Read）的用户。	
查看自定义函数列表	查看MaxCompute项目中所有自定义函数的信息。	具备查看项目对象列表权限（List）的用户。	
查看内建函数列表	查看当前MaxCompute项目中所有内建函数的信息。		

注册函数

在MaxCompute项目中注册自定义函数。

- 使用限制
  - 函数名称全局唯一，同名函数只能注册一次。
  - 自定义函数无法覆盖系统内建函数。只有项目Owner才有权利覆盖内建函数。如果您使用了覆盖内建函数的自定义函数，在SQL执行结束后，会在作业相应Logview的Summary中打印出Warning信息。
- 命令格式

```
create function <function_name> as <'package_to_class'> using <'resource_list'>;
```

- 参数说明
  - function_name：必填。新建的自定义函数名称。
  - package_to_class：必填。自定义函数的类，区分大小写，需要用英文引号（''）引起来。
    - Java UDF：此名称为从顶层包名一直到实现UDF类的类名。
    - Python UDF：此名称为*Python脚本名.类名*。

      > **说明** Python脚本名为底层唯一标识的资源名。MaxCompute的资源名大小写不敏感，例如第一次上传资源时资源名为pyudf_test.py，后续在DataStudio重命名资源或用MaxCompute客户端覆盖资源时将资源名重命名为PYUDF_TEST.py，此时底层唯一标识的资源名仍然为pyudf_test.py。因此注册自定义函数时，类名仍然需要填pyudf_test.SampleUDF。您可以通过 `list resource;` 命令查看所有资源在底层唯一标识的资源名。

  - resource_list：必填。自定义函数所用到的资源列表。
    - 此资源列表必须包括自定义函数代码所在的资源，且资源已经上传至MaxCompute。
    - 如果您的代码通过Distributed Cache接口读取资源文件，此列表中还要包括自定义函数所读取的资源文件列表。
    - 资源列表由多个资源名组成，资源名之间用英文逗号（,）分隔，且资源列表必须用英文引号（''）引起来。
    - 如果需要指定资源所在的项目，格式为 `<project_name>/resources/<resource_name>` 。
- 使用示例
  - 示例1：创建函数 `my_lower` 。假设Java UDF类 `org.alidata.odps.udf.examples.Lower` 在*my_lower.jar*中。

```
create function my_lower as 'org.alidata.odps.udf.examples.Lower' using 'my_lower.jar';
```

- 示例2：创建函数 `my_lower` 。假设Python UDF类MyLower在项目 `test_project` 的脚本*pyudf_test.py*中。

```
create function my_lower as 'pyudf_test.MyLower' using 'test_project/resources/pyudf_test.py';
```

- 示例3：创建函数 `test_udtf` 。假设Java UDF类 `com.aliyun.odps.examples.udf.UDTFResource` 是在*udtfexample1.jar*中，同时该函数还依赖FILE资源*file_resource.txt*、Table资源*table_resource1*以及Archive资源*test_archive.zip*。

```
create function test_udtf as 'com.aliyun.odps.examples.udf.UDTFResource' using 'udtfexample1.jar, file_resource.txt, table_r
esource1, test_archive.zip';
```

注销函数

在MaxCompute项目中注销已注册的自定义函数。

- 命令格式

```
drop function <function_name>;
```

- 参数说明

  function_name：必填。已注册的自定义函数名称。

- 使用示例

```
--注销my_lower函数。
drop function my_lower;
```

查看函数

查看MaxCompute项目中指定自定义函数的信息，包含函数名称、所有者、创建时间、类名和资源列表信息。

- 命令格式

```
desc function <function_name>;
```

- 参数说明

  function_name：必填。已存在的函数名称。

- 返回值说明
  - Name：自定义函数名称。
  - Owner：自定义函数的所有者账号。
  - Created Time：自定义函数的注册时间。
  - Class：自定义函数的类，区分大小写。
  - Resources：自定义函数所用到的资源列表。

- 使用示例

```
--查看函数my_lower的信息。
desc function my_lower;
```

返回结果如下。

```
Name                                my_lower
Owner                               ALIYUN$****
Created Time                        2020-06-18 15:50:19
Class                               org.alidata.odps.udf.examples.Lower
Resources                           project_name/my_lower.jar
```

查看自定义函数列表

查看MaxCompute项目中所有自定义函数的信息。

- 命令格式

```
list functions [-p <project_name>];
```

- 参数说明

  project_name：可选。指定MaxCompute项目的名称。

- 使用示例

```
list functions;
```

返回结果如下。

```
Name             Owner                                  Create Time          Class                            Resourc
es
ipv4_ipv6_aton   ALIYUN$****@aliyun.com 2021-11-15 13:42:14   com.aliyun.odps.udf.udfFunction.IpLocation ipv4.txt,ipv6.txt,ud
f-1.0-SNAPSHOT.jar
```

```
Lower_test        ALIYUN$****@aliyun.com 2021-08-25 15:51:22   com.aliyun.odps.udf.example.Lower udf-1.0-SNAPSHOT.jar
my_add            ALIYUN$****@aliyun.com 2021-05-08 11:26:02
my_index          ALIYUN$****@aliyun.com 2021-08-25 12:01:05   com.aliyun.odps.examples.udf.UdfArray udf-1.0-SNAPSHOT.jar
my_sum            ALIYUN$****@aliyun.com 2021-05-08 10:24:58
my_udtf           ALIYUN$****@aliyun.com 2021-02-23 11:37:30   com.aliyun.odps.examples.udf.UDTFResource udf-1.0-SNAPSHOT.jar
numpy             ALIYUN$****@aliyun.com 2020-11-11 14:12:50   numpy.TryImport                  numpy.py,numpy-1.19.4-cp37-cp
37m-manylinux1_x86_64.zip
ST_Aggr_ConvexHull ALIYUN$****@aliyun.com 2021-03-18 17:06:29   com.esri.hadoop.hive.ST_Aggr_ConvexHull esri-geometry-api.jar,
spatial-sdk-hive.jar
ST_Aggr_Intersection ALIYUN$****@aliyun.com 2021-03-18 17:06:29   com.esri.hadoop.hive.ST_Aggr_Intersection esri-geometry-api.
jar,spatial-sdk-hive.jar
ST_Aggr_Union     ALIYUN$****@aliyun.com 2021-03-18 17:06:30   com.esri.hadoop.hive.ST_Aggr_Union esri-geometry-api.jar,spatia
l-sdk-hive.jar
ST_Area           ALIYUN$****@aliyun.com 2021-03-18 17:06:30   com.esri.hadoop.hive.ST_Area      esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_AsBinary       ALIYUN$****@aliyun.com 2021-03-18 17:06:30   com.esri.hadoop.hive.ST_AsBinary  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_AsGeoJson      ALIYUN$****@aliyun.com 2021-03-18 17:06:49   com.esri.hadoop.hive.ST_AsGeoJson esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_AsJson         ALIYUN$****@aliyun.com 2021-03-18 17:06:50   com.esri.hadoop.hive.ST_AsJson    esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_AsShape        ALIYUN$****@aliyun.com 2021-03-18 17:06:50   com.esri.hadoop.hive.ST_AsShape   esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_AsText         ALIYUN$****@aliyun.com 2021-03-18 17:06:50   com.esri.hadoop.hive.ST_AsText    esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Bin            ALIYUN$****@aliyun.com 2021-03-18 17:06:50   com.esri.hadoop.hive.ST_Bin       esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_BinEnvelope    ALIYUN$****@aliyun.com 2021-03-18 17:07:01   com.esri.hadoop.hive.ST_BinEnvelope esri-geometry-api.jar,spati
al-sdk-hive.jar
ST_Boundary       ALIYUN$****@aliyun.com 2021-03-18 17:07:01   com.esri.hadoop.hive.ST_Boundary  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Buffer         ALIYUN$****@aliyun.com 2021-03-18 17:07:01   com.esri.hadoop.hive.ST_Buffer    esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Centroid       ALIYUN$****@aliyun.com 2021-03-18 17:07:01   com.esri.hadoop.hive.ST_Centroid  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Contains       ALIYUN$****@aliyun.com 2021-03-18 17:07:01   com.esri.hadoop.hive.ST_Contains  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_ConvexHull     ALIYUN$****@aliyun.com 2021-03-18 17:07:13   com.esri.hadoop.hive.ST_ConvexHull esri-geometry-api.jar,spatia
l-sdk-hive.jar
ST_CoordDim       ALIYUN$****@aliyun.com 2021-03-18 17:07:14   com.esri.hadoop.hive.ST_CoordDim  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Crosses        ALIYUN$****@aliyun.com 2021-03-18 17:07:14   com.esri.hadoop.hive.ST_Crosses   esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Difference     ALIYUN$****@aliyun.com 2021-03-18 17:07:14   com.esri.hadoop.hive.ST_Difference esri-geometry-api.jar,spatia
l-sdk-hive.jar
ST_Dimension      ALIYUN$****@aliyun.com 2021-03-18 17:07:14   com.esri.hadoop.hive.ST_Dimension esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Disjoint       ALIYUN$****@aliyun.com 2021-03-18 17:07:31   com.esri.hadoop.hive.ST_Disjoint  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Distance       ALIYUN$****@aliyun.com 2021-03-18 17:07:31   com.esri.hadoop.hive.ST_Distance  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_EndPoint       ALIYUN$****@aliyun.com 2021-03-18 17:07:31   com.esri.hadoop.hive.ST_EndPoint  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_Envelope       ALIYUN$****@aliyun.com 2021-03-18 17:07:32   com.esri.hadoop.hive.ST_Envelope  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_EnvIntersects  ALIYUN$****@aliyun.com 2021-03-18 17:07:32   com.esri.hadoop.hive.ST_EnvIntersects esri-geometry-api.jar,spa
tial-sdk-hive.jar
ST_Equals         ALIYUN$****@aliyun.com 2021-03-18 17:07:44   com.esri.hadoop.hive.ST_Equals    esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_ExteriorRing   ALIYUN$****@aliyun.com 2021-03-18 17:07:44   com.esri.hadoop.hive.ST_ExteriorRing esri-geometry-api.jar,spat
ial-sdk-hive.jar
ST_GeodesicLengthWGS84 ALIYUN$****@aliyun.com 2021-03-18 17:07:44   com.esri.hadoop.hive.ST_GeodesicLengthWGS84 esri-geometry-
api.jar,spatial-sdk-hive.jar
ST_GeomCollection ALIYUN$****@aliyun.com 2021-03-18 17:07:44   com.esri.hadoop.hive.ST_GeomCollection esri-geometry-api.jar,sp
atial-sdk-hive.jar
ST_Geometry       ALIYUN$****@aliyun.com 2021-03-18 17:07:44   com.esri.hadoop.hive.ST_Geometry  esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_GeometryN      ALIYUN$****@aliyun.com 2021-03-18 17:07:55   com.esri.hadoop.hive.ST_GeometryN esri-geometry-api.jar,spatial
-sdk-hive.jar
ST_GeometryType   ALIYUN$****@aliyun.com 2021-03-18 17:07:55   com.esri.hadoop.hive.ST_GeometryType esri-geometry-api.jar,spat
ial-sdk-hive.jar
ST_GeomFromGeoJson ALIYUN$****@aliyun.com 2021-03-18 17:07:55   com.esri.hadoop.hive.ST_GeomFromGeoJson esri-geometry-api.jar,
spatial-sdk-hive.jar
ST_GeomFromJson   ALIYUN$****@aliyun.com 2021-03-18 17:07:55   com.esri.hadoop.hive.ST_GeomFromJson esri-geometry-api.jar,spat
ial-sdk-hive.jar
```

```
ST_GeomFromShape  ALIYUN$****@aliyun.com 2021-03-18 17:07:56   com.esri.hadoop.hive.ST_GeomFromShape esri-geometry-api.jar,spa
tial-sdk-hive.jar
ST_GeomFromText   ALIYUN$****@aliyun.com 2021-03-18 17:08:10   com.esri.hadoop.hive.ST_GeomFromText esri-geometry-api.jar,spat
ial-sdk-hive.jar
ST_GeomFromWKB    ALIYUN$****@aliyun.com 2021-03-18 17:08:10   com.esri.hadoop.hive.ST_GeomFromWKB esri-geometry-api.jar,spati
al-sdk-hive.jar
```

查看内建函数列表

查看当前MaxCompute项目中所有内建函数的信息。

- 命令格式

```
show functions;
```

- 使用示例

```
show functions;
```

返回结果如下。

```
ID = 20211214091641326gg0g****
::::ABS SCALAR  INT ABS(INT arg0),DOUBLE ABS(DOUBLE arg0),DECIMAL(?,?) ABS(DECIMAL(?,?) arg0),BIGINT ABS(BIGINT arg0)
::::ACOS        SCALAR  DOUBLE ACOS(DOUBLE arg0),DOUBLE ACOS(DECIMAL(?,?) arg0)
::::ADD_MONTHS  SCALAR  STRING ADD_MONTHS(TIMESTAMP arg0, BIGINT arg1),STRING ADD_MONTHS(STRING arg0, BIGINT arg1),STRING ADD_
MONTHS(DATE arg0, BIGINT arg1)
::::ALL_MATCH   SCALAR  BOOLEAN ALL_MATCH(ARRAY<T> arg0, java.util.function.Function<T, java.lang.Boolean> arg1)
::::ANY_MATCH   SCALAR  BOOLEAN ANY_MATCH(ARRAY<T> arg0, java.util.function.Function<T, java.lang.Boolean> arg1)
::::ANY_VALUE   AGGREGATOR      T ANY_VALUE([DISTINCT] T arg1)
::::APPROX_DISTINCT     AGGREGATOR      BIGINT APPROX_DISTINCT([DISTINCT] P arg1, DOUBLE arg2),BIGINT APPROX_DISTINCT([DISTINC
T] P arg1)
::::ARG_MAX     AGGREGATOR      R ARG_MAX([DISTINCT] T arg1, R arg2)
::::ARG_MIN     AGGREGATOR      R ARG_MIN([DISTINCT] T arg1, R arg2)
::::ARRAY       SCALAR  ARRAY<STRING> ARRAY(),ARRAY<T> ARRAY(T arg0...)
::::ARRAYS_OVERLAP      SCALAR  BOOLEAN ARRAYS_OVERLAP(ARRAY<T> arg0, ARRAY<T> arg1)
::::ARRAYS_ZIP  SCALAR  null
::::ARRAY_CONTAINS      SCALAR  BOOLEAN ARRAY_CONTAINS(ARRAY<T> arg0, T arg1)
::::ARRAY_DISTINCT      SCALAR  ARRAY<T> ARRAY_DISTINCT(ARRAY<T> arg0)
::::ARRAY_EXCEPT        SCALAR  ARRAY<T> ARRAY_EXCEPT(ARRAY<T> arg0, ARRAY<T> arg1)
::::ARRAY_INTERSECT     SCALAR  ARRAY<T> ARRAY_INTERSECT(ARRAY<T> arg0, ARRAY<T> arg1)
::::ARRAY_JOIN  SCALAR  STRING ARRAY_JOIN(ARRAY<STRING> arg0, STRING arg1),STRING ARRAY_JOIN(ARRAY<STRING> arg0, STRING arg1,
STRING arg2)
::::ARRAY_MAX   SCALAR  T ARRAY_MAX(ARRAY<T> arg0)
::::ARRAY_MIN   SCALAR  T ARRAY_MIN(ARRAY<T> arg0)
::::ARRAY_POSITION      SCALAR  BIGINT ARRAY_POSITION(ARRAY<T> arg0, T arg1)
::::ARRAY_REDUCE        SCALAR  OUT ARRAY_REDUCE(ARRAY<IN> arg0, BUF arg1, java.util.function.BiFunction<BUF, IN, BUF> arg2, j
ava.util.function.Function<BUF, OUT> arg3)
::::ARRAY_REMOVE        SCALAR  ARRAY<T> ARRAY_REMOVE(ARRAY<T> arg0, T arg1)
::::ARRAY_REPEAT        SCALAR  ARRAY<T> ARRAY_REPEAT(T arg0, BIGINT arg1)
::::ARRAY_SORT  SCALAR  ARRAY<T> ARRAY_SORT(ARRAY<T> arg0, java.util.function.BiFunction<T, T, java.lang.Long> arg1)
::::ARRAY_UNION SCALAR  ARRAY<T> ARRAY_UNION(ARRAY<T> arg0, ARRAY<T> arg1)
::::ASCII       SCALAR  BIGINT ASCII(STRING arg0)
::::ASIN        SCALAR  DOUBLE ASIN(DOUBLE arg0),DOUBLE ASIN(DECIMAL(?,?) arg0)
::::ATAN        SCALAR  DOUBLE ATAN(DOUBLE arg0),DOUBLE ATAN(DECIMAL(?,?) arg0)
::::AT_MOST_ONE_ROW     AGGREGATOR      T AT_MOST_ONE_ROW([DISTINCT] T arg1)
::::AVG AGGREGATOR      DECIMAL(?,?) AVG([DISTINCT] DECIMAL(?,?) arg1),DOUBLE AVG([DISTINCT] DOUBLE arg1)
::::AVG WINDOW  DOUBLE AVG([DISTINCT] DOUBLE arg0),DECIMAL(?,?) AVG([DISTINCT] DECIMAL(?,?) arg0)
::::BASE64      SCALAR  STRING BASE64(BINARY arg0)
::::BIN SCALAR  STRING BIN(BIGINT arg0)
::::BITAND      SCALAR  BIGINT BITAND(BIGINT arg0, BIGINT arg1)
::::BITNOT      SCALAR  BIGINT BITNOT(BIGINT arg0)
::::BITOR       SCALAR  BIGINT BITOR(BIGINT arg0, BIGINT arg1)
::::BITXOR      SCALAR  BIGINT BITXOR(BIGINT arg0, BIGINT arg1)
::::BROUND      SCALAR  DOUBLE BROUND(DOUBLE arg0, BIGINT arg1),DOUBLE BROUND(DOUBLE arg0)
::::CASE        SCALAR  null
......
```

2.12. Tunnel操作

MaxCompute通过Tunnel实现上传下载数据功能。本文为您介绍如何通过Tunnel上传、下载数据。

Tunnel操作详情请参见Tunnel命令。Tunnel操作常用命令如下。

类型	功能	角色	操作入口

类型	功能	角色	操作入口
上传数据	将本地文件的数据上传至MaxCompute的表中，以追加模式导入。	具备修改表权限（Alter）的用户。	本文中的命令您需要在MaxCompute客户端执行。
下载数据	将MaxCompute表数据或指定Instance的执行结果下载至本地。	具备读取表数据权限（Select）的用户。	

上传数据

将本地文件的数据上传至MaxCompute的表中，以追加模式导入。上传数据至MaxCompute不收取费用。

- 使用限制
  - 支持文件或目录（指一级目录）的上传，每一次上传只支持数据上传到一张表或表的一个分区。
  - 分区表一定要指定上传的分区，多级分区一定要指定到末级分区。
- 命令格式

```
Tunnel upload <path> [<project_name>.]<table_name>[/<pt_spc>];
```

- 参数说明
  - path：必填。上传数据文件的存放路径以及文件名称。默认上传的数据文件为TXT格式。

    数据文件的归档路径有两种选择：您可以将文件直接归档至MaxCompute客户端的 `bin` 目录中，此时path需要设置为 `文件名.后缀名` ；也可以将文件归档至其他路径下，例如D盘的test文件夹，path需要设置为 `D:\test\文件名.后缀名` 。
  - project_name：可选。目标表所属项目空间名称。跨项目空间访问表时需要指定该参数。
  - table_name：必填。目标表名。
  - pt_spc：可选。需要指定至最末级分区。格式为 `partition_col1=col1_value1, partition_col2=col2_value1...` 。
- 使用示例
  - 示例1：将log.txt中的数据上传至当前项目空间的表test_table中。log.txt文件归档在MaxCompute客户端的 `bin` 目录中。命令示例如下。

    ```
    Tunnel upload log.txt test_table;
    ```
  - 示例2：将log.txt中的数据上传至项目空间test_project的表test_table（二级分区表）中的 `p1="b1",p2="b2"` 分区。log.txt文件归档在D盘的test文件夹中。命令示例如下。

    ```
    Tunnel upload D:\test\log.txt test_project.test_table/p1="b1",p2="b2";
    ```

下载数据

将MaxCompute表数据或指定Instance的执行结果下载至本地。MaxCompute仅对公网的数据下载，按照下载的数据大小进行计费，一次下载费用=下载数据量（GB）×下载价格（0.8元/GB）。

- 使用限制
  - 只支持下载到单个文件，每一次下载只支持下载一张表或一个分区到一个文件。
  - 分区表一定要指定下载的分区，多级分区一定要指定到末级分区。
- 命令格式

```
Tunnel download [<project_name>.]<table_name>[/<pt_spc>] <path>;
```

- 参数说明
  - project_name：可选。目标表所属项目空间名称。跨项目空间访问表时需要指定该参数。
  - table_name：必填。目标表表名。
  - pt_spc：可选。需要指定至最末级分区。格式为 `partition_col1=col1_value1, partition_col2=col2_value1...` 。
  - path：必填。下载数据文件的存放路径以及文件名称。默认下载的数据文件为TXT格式。

    数据文件的保存路径有两种选择：您可以将文件直接下载至MaxCompute客户端的 `bin` 目录中，此时path需要设置为 `文件名.后缀名` ；也可以将文件下载至其他路径下，例如D盘的test文件夹，path需要设置为 `D:\test\文件名.后缀名` 。
- 使用示例
  - 示例1：将test_project.test_table表（二级分区表）中的数据下载到test_table.txt文件中。test_table.txt文件位于MaxCompute客户端的 `bin` 目录中。命令示例如下。

    ```
    Tunnel download test_project.test_table/p1="b1",p2="b2" test_table.txt;
    ```
  - 示例2：将test_project.test_table表（二级分区表）中的数据下载到test_table.txt文件中。test_table.txt文件位于D盘的test文件夹中。命令示例如下。

    ```
    Tunnel download test_project.test_table/p1="b1",p2="b2" D:\test\test_table.txt;
    ```

2.13. 时区配置操作

本文为您介绍如何使用SET命令配置MaxCompute Project的时区。

支持时区功能的作业如下：

- MapReduce支持时区功能。
- Spark支持时区功能。
  - 对于提交到MaxCompute计算集群的任务，可自动获取Project的时区。
  - 对于通过yarn-client模式启动（例如spark-shell，spark-sql，pyspark等）的设置，您需要手动配置Driver的启动参数（*spark-defaults.conf*），增加 `spark.driver.extraJavaOptions -Duser.timezone=America/Los_Angeles` ，timezone的值为将要使用的时区。
- PAI支持时区功能。
- Graph支持时区功能。

配置时区

MaxCompute Project时区默认是中国的东八区，DATETIME、TIMESTAMP、DATE类型字段以及相关时间内置函数按照东八区进行计算。您可以通过以下两种方式配置时区：

- Session级别：执行 `SET odps.sql.timezone=<timezoneid>;` 语句，需要与计算语句一起提交。

```
--设置时区为Asia/Tokyo。
SET odps.sql.timezone=Asia/Tokyo;
--查询当前时区。
SELECT getdate();
output:
+------------+
| _c0        |
+------------+
| 2018-10-30 23:49:50 |
+------------+
```

- Project级别：执行 `setproject odps.sql.timezone=<timezoneid>;` 语句，此命令需要项目所有者（Project Owner）执行。

> **注意** Project的时区一旦被设置，相关的时间计算会取设置后的时区，原有的作业数据将会受到影响。因此，请您谨慎考虑是否有必要设置时区。如果必要，建议只对新增的Project进行时区设置，不对已有数据的Project进行设置。

使用限制及注意事项

- SQL内置日期函数、UDF、UDT、UDJ、 `SELECT TRANSFORM` 支持获取Project Timezone属性来配置时区。
- 时区支持的格式类型为 `Asia/Shanghai` （存在夏令时跳变），不支持GMT+9格式。
- 当SDK时区与Project时区不同时，DATETIME类型转化为STRING类型的操作需设置GMT时区。
- 时区配置后，通过DataWorks执行相关SQL时，某些时间段的日期显示会存在差异。例如，1900~1928年的日期时间差异为5分52秒，1900年之前的日期时间差异为9秒。
- 为了保证MaxCompute在多个时区DATETIME类型数据的正确性，MaxCompute服务、Java SDK以及客户端将会进行版本更新（*-oversea*后缀的Java SDK或客户端版本），更新后可能影响MaxCompute中已经存储的早于1928年的DATETIME类型数据的显示。
- 对于非中国东八区的区域，建议您同步更新Java SDK或客户端版本，以保证在1900-01-01之后的SQL计算结果及Tunnel传输数据的准确性和一致性。对于早于1900-01-01的DATETIME数据，SQL的计算显示结果和Tunnel传输数据仍然可能存在343秒的差异。对于新版本SDK或客户端，之前已经上传的早于1928-01-01的DATETIME数据，在新版本中日期时间会减少352秒。
- 如果继续使用不带有*-oversea*后缀的SDK或客户端，SQL计算结果和Tunnel传输数据将存在差异。早于1900-01-01的数据差异为9秒，1900-01-01~1928-01-01的数据差异为352秒。

> **说明** Java SDK或客户端版本更新配置时区不影响DataWorks的时区配置，因此时区会存在差异，需要您对DataWorks中定时任务调度的影响进行计算评估。DataWorks服务器在日本区域的时区是GMT+9，在新加坡Region的时区是GMT+8。

- 通过JDBC连接的第三方客户端需要在客户端设置时区，保证与服务端时区设置的一致性。

2.14. SHOW

本文为您介绍不同操作中SHOW命令的用法以及示例。

不同操作中SHOW命令的用法以及示例如下表所示。

操作	说明
SET操作	显示SET命令设置。
Tunnel操作	查看上传或下载数据的详细命令或日志。
安全操作	在当前项目中查看项目的安全配置属性。

操作	说明
Package操作	在MaxCompute项目中查看已创建或已安装的Package列表。
备份操作	• 查看当前项目内的表和处于备份状态的表信息，包括表名、表ID、创建时间和删除时间等。 • 查看指定表的备份数据，获取保留周期内备份的各个数据版本信息。 • 查看已删除表的备份数据，获取保留周期内备份的各个数据版本信息。 • 查看指定分区的备份数据，获取保留周期内备份的各个数据版本信息。 • 查看已删除分区的备份数据，获取保留周期内备份的各个数据版本信息。
表操作	• 查看建表语句。 • 查看项目下所有的表和视图，或符合某规则的表和视图。 • 查看某张表下的所有分区。
函数操作	查看当前MaxCompute项目中所有内建函数的信息。
实例操作	查看实例信息，包括StartTime（时间精确到秒）、RunTime（s）、Status（实例状态）、InstanceID、实例所有者、Query（执行语句）。
查询权限信息	查询用户、角色、对象等的权限信息。
优化器信息收集	测试Column Stats收集结果。

SET操作

显示set命令设置的属性。命令格式如下：

```
show flags;
```

> 说明 更多关于SET操作的介绍，请参见SET操作。

Tunnel操作

- 显示历史记录。
  - 命令格式

    ```
    tunnel show history [-n <number>];
    ```

    -n <number>：tunnel执行的次数。
  - 命令示例

    示例1：显示历史记录，默认保存500条数据。

    ```
    tunnel show history;
    ```

    示例2：显示前5次上传或下载数据的详细命令。

    ```
    tunnel show history -n 5;
    ```
- 显示最后一次上传或下载数据的日志。

  ```
  tunnel show log;
  ```

> 说明 更多关于Tunnel命令的介绍，请参见Tunnel命令。

安全操作

在当前项目中查看项目的安全配置属性。命令格式如下：

```
show SecurityConfiguration;
```

> 说明 更多关于安全操作的命令介绍，请参见安全操作。

Package操作

在MaxCompute项目中查看已创建或已安装的Package列表。

- 命令格式

```
show packages;
```

- 使用示例

  查看test_project_a项目中创建或安装的Package信息。命令示例如下。

  ```
  --查看Package列表。
  show packages;
  ```

  返回结果如下。包含如下信息：
  - 创建或安装Package的名称（PackageName）
  - 创建时间（CreateTime）
  - Package所属MaxCompute项目（SourceProject）
  - 安装时间（InstallTime）
  - 状态（Status）

  ```
  +-------------+--------------------------+
  | PackageName | CreateTime               |
  +-------------+--------------------------+
  | datashare   | 2021-12-28T18:10:39+0800 |
  +-------------+--------------------------+
  +-------------+--------------------+--------------------------+--------+
  | PackageName | SourceProject      | InstallTime              | Status |
  +-------------+--------------------+--------------------------+--------+
  | systables   | information_schema | 2020-11-24T14:11:23+0800 | OK     |
  +-------------+--------------------+--------------------------+--------+
  ```

> **说明** 更多关于Package命令操作介绍，请参见基于Package跨项目访问资源。

备份操作

- 查看当前项目内的表和处于备份状态的表信息，包括表名、表ID、创建时间和删除时间等。

  ```
  show history for tables;
  ```

- 查看指定表的备份数据，获取保留周期内备份的各个数据版本信息。

  ```
  show history for table <table_name>;
  ```

- 查看已删除表的备份数据，获取保留周期内备份的各个数据版本信息。

  ```
  show history for table table_name ('id'='xxxx');
  ```

- 查看指定分区的备份数据，获取保留周期内备份的各个数据版本信息。

  ```
  show history for table table_name partition_spec;
  ```

- 查看已删除分区的备份数据，获取保留周期内备份的各个数据版本信息。id可以通过 `show history for table <table_name>;` 命令返回结果中的ObjectId字段获取。

  ```
  show history for table table_name PARTITION('id'='xxxx');
  ```

> **说明** 更多关于备份与恢复命令的介绍，请参见备份与恢复。

表操作

- 查看建表语句。
  - 命令格式

    ```
    show create table <table_name>;
    ```

  - 参数说明

    table_name：必填。待查看建表语句的表的名称。

  - 使用示例

    ```
    --查看表sale_detail的建表语句。
    show create table sale_detail;
    ```

    返回结果如下。

    ```
    CREATE TABLE IF NOT EXISTS doc_test_dev.sale_detail(shop_name STRING,customer_id STRING,total_price DOUBLE) PARTITIONED BY (
    sale_date STRING,region STRING) STORED AS ALIORC;
    ```

- 列出项目下所有的表和视图，或符合某规则的表和视图。
  - 命令格式

```
--列出项目下所有的表和视图。
show tables;
--列出项目下表名或视图名与chart匹配的表。
show tables like '<chart>';
```

  - 使用示例

```
--列出项目下表名与sale*匹配的表。*表示任意字段。
show tables like 'sale*';
```

    返回结果如下。

```
ALIYUN$account_name:sale_detail
......
--ALIYUN是系统提示符，表示您是阿里云主账号用户。如果您是阿里云RAM用户，系统提示符为RAM。
```

- 列出一张表中的所有分区，表不存在或为非分区表时，返回报错。
  - 命令格式

```
show partitions <table_name>;
```

  - 参数说明

    table_name：必填。待查看分区信息的分区表名称。

  - 使用示例

```
--列出sale_detail中的所有分区。
show partitions sale_detail;
```

    返回结果如下。

```
sale_date=201310/region=beijing
sale_date=201312/region=shenzhen
sale_date=201312/region=xian
sale_date=2014/region=shenzhen
OK
```

> 说明 更多关于表操作的命令介绍，请参见表操作。

函数操作

查看当前MaxCompute项目中所有内建函数的信息。

- 命令格式

```
show functions;
```

- 使用示例

```
show functions;
```

  返回结果如下。

```
ID = 20211214091641326gg0g****
::::ABS SCALAR  INT ABS(INT arg0),DOUBLE ABS(DOUBLE arg0),DECIMAL(?,?) ABS(DECIMAL(?,?) arg0),BIGINT ABS(BIGINT arg0)
::::ACOS        SCALAR  DOUBLE ACOS(DOUBLE arg0),DOUBLE ACOS(DECIMAL(?,?) arg0)
::::ADD_MONTHS  SCALAR  STRING ADD_MONTHS(TIMESTAMP arg0, BIGINT arg1),STRING ADD_MONTHS(STRING arg0, BIGINT arg1),STRING ADD_
MONTHS(DATE arg0, BIGINT arg1)
::::ALL_MATCH   SCALAR  BOOLEAN ALL_MATCH(ARRAY<T> arg0, java.util.function.Function<T, java.lang.Boolean> arg1)
::::ANY_MATCH   SCALAR  BOOLEAN ANY_MATCH(ARRAY<T> arg0, java.util.function.Function<T, java.lang.Boolean> arg1)
::::ANY_VALUE   AGGREGATOR      T ANY_VALUE([DISTINCT] T arg1)
::::APPROX_DISTINCT     AGGREGATOR      BIGINT APPROX_DISTINCT([DISTINCT] P arg1, DOUBLE arg2),BIGINT APPROX_DISTINCT([DISTINC
T] P arg1)
::::ARG_MAX     AGGREGATOR      R ARG_MAX([DISTINCT] T arg1, R arg2)
::::ARG_MIN     AGGREGATOR      R ARG_MIN([DISTINCT] T arg1, R arg2)
::::ARRAY       SCALAR  ARRAY<STRING> ARRAY(),ARRAY<T> ARRAY(T arg0...)
::::ARRAYS_OVERLAP      SCALAR  BOOLEAN ARRAYS_OVERLAP(ARRAY<T> arg0, ARRAY<T> arg1)
::::ARRAYS_ZIP  SCALAR  null
::::ARRAY_CONTAINS      SCALAR  BOOLEAN ARRAY_CONTAINS(ARRAY<T> arg0, T arg1)
::::ARRAY_DISTINCT      SCALAR  ARRAY<T> ARRAY_DISTINCT(ARRAY<T> arg0)
::::ARRAY_EXCEPT        SCALAR  ARRAY<T> ARRAY_EXCEPT(ARRAY<T> arg0, ARRAY<T> arg1)
::::ARRAY_INTERSECT     SCALAR  ARRAY<T> ARRAY_INTERSECT(ARRAY<T> arg0, ARRAY<T> arg1)
::::ARRAY_JOIN  SCALAR  STRING ARRAY_JOIN(ARRAY<STRING> arg0, STRING arg1),STRING ARRAY_JOIN(ARRAY<STRING> arg0, STRING arg1,
STRING arg2)
::::ARRAY_MAX   SCALAR  T ARRAY_MAX(ARRAY<T> arg0)
::::ARRAY_MIN   SCALAR  T ARRAY_MIN(ARRAY<T> arg0)
::::ARRAY_POSITION      SCALAR  BIGINT ARRAY_POSITION(ARRAY<T> arg0, T arg1)
::::ARRAY_REDUCE        SCALAR  OUT ARRAY_REDUCE(ARRAY<IN> arg0, BUF arg1, java.util.function.BiFunction<BUF, IN, BUF> arg2, j
ava.util.function.Function<BUF, OUT> arg3)
::::ARRAY_REMOVE        SCALAR  ARRAY<T> ARRAY_REMOVE(ARRAY<T> arg0, T arg1)
::::ARRAY_REPEAT        SCALAR  ARRAY<T> ARRAY_REPEAT(T arg0, BIGINT arg1)
::::ARRAY_SORT  SCALAR  ARRAY<T> ARRAY_SORT(ARRAY<T> arg0, java.util.function.BiFunction<T, T, java.lang.Long> arg1)
::::ARRAY_UNION SCALAR  ARRAY<T> ARRAY_UNION(ARRAY<T> arg0, ARRAY<T> arg1)
::::ASCII       SCALAR  BIGINT ASCII(STRING arg0)
::::ASIN        SCALAR  DOUBLE ASIN(DOUBLE arg0),DOUBLE ASIN(DECIMAL(?,?) arg0)
::::ATAN        SCALAR  DOUBLE ATAN(DOUBLE arg0),DOUBLE ATAN(DECIMAL(?,?) arg0)
::::AT_MOST_ONE_ROW     AGGREGATOR      T AT_MOST_ONE_ROW([DISTINCT] T arg1)
::::AVG AGGREGATOR      DECIMAL(?,?) AVG([DISTINCT] DECIMAL(?,?) arg1),DOUBLE AVG([DISTINCT] DOUBLE arg1)
::::AVG WINDOW  DOUBLE AVG([DISTINCT] DOUBLE arg0),DECIMAL(?,?) AVG([DISTINCT] DECIMAL(?,?) arg0)
::::BASE64      SCALAR  STRING BASE64(BINARY arg0)
::::BIN SCALAR  STRING BIN(BIGINT arg0)
::::BITAND      SCALAR  BIGINT BITAND(BIGINT arg0, BIGINT arg1)
::::BITNOT      SCALAR  BIGINT BITNOT(BIGINT arg0)
::::BITOR       SCALAR  BIGINT BITOR(BIGINT arg0, BIGINT arg1)
::::BITXOR      SCALAR  BIGINT BITXOR(BIGINT arg0, BIGINT arg1)
::::BROUND      SCALAR  DOUBLE BROUND(DOUBLE arg0, BIGINT arg1),DOUBLE BROUND(DOUBLE arg0)
::::CASE        SCALAR  null
......
```

> **说明** 更多关于函数操作的命令介绍，请参见函数操作。

实例操作

查看实例信息，包括StartTime（时间精确到秒）、RunTime（s）、Status（实例状态）、InstanceID、实例所有者、Query（执行语句）。

- 命令格式

```
show p|proc|processlist|instances [from <startdate>] [to <enddate>] [-p <project_name>] [-limit <number> | <number>] [-all];
ls|list instances [from <startdate>] [to <enddate>] [-p <project_name>] [-limit <number> | <number>] [-all];
```

`show p`、`show proc`、`show processlist`、`show instances`、`ls instances`和`list instances`命令等效。

- 参数说明
  - startdate、enddate：可选。返回从起始时间startdate到结束时间enddate期间用户提交的实例信息，startdate要小于enddate。不包含enddate当天提交的实例信息。格式必须为`yyyy-mm-dd`，精度为天。如果不指定，则返回用户当天提交的实例。
  - project_name：可选。待查询实例所属的MaxCompute项目名称，需要具备目标MaxCompute项目的查看实例权限。如果不指定，则表示查询当前所处的MaxCompute项目。
  - number：可选。指定返回实例的数量。
  - 依照时间排序，返回提交时间距离当前时间最近的number个实例信息。如果不指定number，则返回满足要求的所有实例信息。`-limit <number>`与`number`二者等效。
  - -all：可选。返回MaxCompute项目中所有成员执行过的实例。如果不指定，则表示返回MaxCompute项目中当前用户执行过的实例。

    不配置number时，默认最大返回50条；如果配置了number，返回number个实例信息。
- 使用示例

- 示例1：查看当前MaxCompute项目下当前用户当天执行的所有实例信息。命令示例如下。

```
show p;
```

返回结果如下。

```
StartTime             RunTime  Status   InstanceID                Owner                          Query
2021-09-14 11:43:04   0s       Success  20210914**************3rw2   ALIYUN$****@test.aliyunid.com
2021-09-14 11:43:05   1s       Success  20210914**************5t32   ALIYUN$****@test.aliyunid.com select date_sub(datetime
'2005-03-01 00:00:00', 1);
2021-09-14 11:58:13   0s       Success  20210914**************5pr2   ALIYUN$****@test.aliyunid.com
2021-09-14 11:58:15   1s       Success  20210914**************5qgr   ALIYUN$****@test.aliyunid.com select date_sub(date '20
05-02-28', -1);
2021-09-14 12:02:15   1s       Success  20210914**************h8o7   ALIYUN$****@test.aliyunid.com select date_sub('2008-03
-01 00:00:00', 2);
2021-09-14 12:02:15   0s       Success  20210914**************5t32   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:31   0s       Success  20210914**************5pr2   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:32   0s       Success  20210914**************euq2   ALIYUN$****@test.aliyunid.com select date_sub('2005-03
-01 00:00:00', 2);
2021-09-14 13:35:42   0s       Success  20210914**************1ms2   ALIYUN$****@test.aliyunid.com
2021-09-14 13:35:43   0s       Success  20210914**************j8o7   ALIYUN$****@test.aliyunid.com select date_sub(getdate(
),1);
2021-09-14 13:40:40   1s       Success  20210914**************h3wz   ALIYUN$****@test.aliyunid.com select date_sub(getdate(
),0);
2021-09-14 13:40:40   0s       Success  20210914**************9nm7   ALIYUN$****@test.aliyunid.com
12 instances
```

- 示例2：查看当前MaxCompute项目下当前用户在指定时间段内执行的实例，并指定返回实例数量。命令示例如下。

```
show instances from 2021-09-14 to 2021-09-15 -limit 10;
```

返回结果如下。

```
StartTime             RunTime  Status   InstanceID                Owner                          Query
2021-09-14 11:58:13   0s       Success  20210914**************5pr2   ALIYUN$****@test.aliyunid.com
2021-09-14 11:58:15   1s       Success  20210914**************5qgr   ALIYUN$****@test.aliyunid.com    select date_sub(date
'2005-02-28', -1);
2021-09-14 12:02:15   1s       Success  20210914**************h8o7   ALIYUN$****@test.aliyunid.com    select date_sub('2008
-03-01 00:00:00', 2);
2021-09-14 12:02:15   0s       Success  20210914**************5t32   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:31   0s       Success  20210914**************5pr2   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:32   0s       Success  20210914**************euq2   ALIYUN$****@test.aliyunid.com    select date_sub('2005
-03-01 00:00:00', 2);
2021-09-14 13:35:42   0s       Success  20210914**************1ms2   ALIYUN$****@test.aliyunid.com
2021-09-14 13:35:43   0s       Success  20210914**************j8o7   ALIYUN$****@test.aliyunid.com    select date_sub(getda
te(),1);
2021-09-14 13:40:40   1s       Success  20210914**************h3wz   ALIYUN$****@test.aliyunid.com    select date_sub(getda
te(),0);
2021-09-14 13:40:40   0s       Success  20210914**************9nm7   ALIYUN$****@test.aliyunid.com
10 instances
```

- 示例3：查看另一MaxCompute项目下所有用户在指定时间段内执行的实例，并指定返回实例数量。命令示例如下。

```
ls instances from 2021-09-14 to 2021-09-15 -p doc_test_dev -all -limit 10;
```

返回结果如下。

```
StartTime             RunTime  Status   InstanceID                Owner                          Query
2021-09-14 11:59:16   0s       Success  20210914**************6pr3   ALIYUN$****@test.aliyunid.com
2021-09-14 11:59:20   1s       Success  20210914**************6qgr   ALIYUN$****@test.aliyunid.com    select date_sub(date
'2007-02-26', -1);
2021-09-14 12:02:19   1s       Success  20210914**************h8o7   ALIYUN$****@test.aliyunid.com    select date_sub('2009
-03-01 00:00:00', 2);
2021-09-14 12:02:25   0s       Success  20210914**************7t42   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:37   0s       Success  20210914**************7pr2   ALIYUN$****@test.aliyunid.com
2021-09-14 12:02:40   0s       Success  20210914**************emq2   ALIYUN$****@test.aliyunid.com    select date_sub('2015
-03-01 00:00:00', 2);
2021-09-14 13:35:42   0s       Success  20210914**************1ms2   ALIYUN$****@test.aliyunid.com
2021-09-14 13:35:43   0s       Success  20210914**************68o7   ALIYUN$kiki                      select date_sub(getda
te(),1);
2021-09-14 13:45:40   1s       Success  20210914**************73wz   ALIYUN$kiki                      select date_sub(getda
te(),0);
2021-09-14 13:45:45   0s       Success  20210914**************9nm7   ALIYUN$dreak
10 instances
```

- 示例4：查看另一MaxCompute项目下所有用户在当天执行的实例，并指定返回实例数量。命令示例如下。

```
show p -p doc_test_dev -all 5;
```

返回结果如下。

```
StartTime             RunTime  Status   InstanceID                  Owner                          Query
2021-09-14 12:02:40   0s       Success  20210914**************emq2   ALIYUN$****@test.aliyunid.com  select date_sub('2015
-03-01 00:00:00', 2);
2021-09-14 13:35:42   0s       Success  20210914**************1ms2   ALIYUN$****@test.aliyunid.com
2021-09-14 13:35:43   0s       Success  20210914**************68o7   ALIYUN$kiki                    select date_sub(getda
te(),1);
2021-09-14 13:45:40   1s       Success  20210914**************73wz   ALIYUN$kiki                    select date_sub(getda
te(),0);
2021-09-14 13:45:45   0s       Success  20210914**************9nm7   ALIYUN$dreak
5 instances
```

> **说明** 更多关于实例操作的命令，请参见实例操作。

查询权限信息

MaxCompute支持通过SHOW命令查询用户、角色、对象等的权限信息。详情请参见通过MaxCompute SQL查询权限信息或权限命令集。

优化器信息收集

执行 `show statistic` 命令测试Column Stats收集结果。

- 命令示例

```
--测试tinyint1列的Column Stats收集结果。
show statistic analyze2_test columns (tinyint1);
--测试smallint1、string1、boolean1和timestamp1列的Column Stats收集结果。
show statistic analyze2_test columns (smallint1, string1, boolean1, timestamp1);
--测试全部列的Column Stats收集结果。
show statistic analyze2_test columns;
```

- 返回结果

```
--tinyint1列的Column Stats收集结果。
ID = 20201126085225150gnqo****
tinyint1:MaxValue:      20                  --对应max。
tinyint1:DistinctNum:   4.0                 --对应ndv。
tinyint1:MinValue:      1                   --对应min。
tinyint1:NullNum:       1.0                 --对应nNulls。
tinyint1:TopK:  {1=1.0, 10=1.0, 20=1.0}     --对应topK。10=1.0表示列值10出现的频次为1。topK最多显示前20个最高频次的值。
--smallint1、string1、boolean1和timestamp1列的Column Stats收集结果。
ID = 20201126091636149gxgf****
smallint1:MaxValue:     20
smallint1:DistinctNum:  4.0
smallint1:MinValue:     2
smallint1:NullNum:      1.0
smallint1:TopK:         {2=1.0, 7=1.0, 20=1.0}
string1:MaxLength       6.0                 --对应maxColLen。
string1:AvgLength:      3.0                 --对应avgColLen。
string1:DistinctNum:    4.0
string1:NullNum:        1.0
string1:TopK:   {str1=1.0, str12=1.0, str123=1.0}
boolean1:DistinctNum:   3.0
boolean1:NullNum:       1.0
boolean1:TopK:  {false=2.0, true=1.0}
timestamp1:DistinctNum:         3.0
timestamp1:NullNum:     1.0
timestamp1:TopK:        {2018-09-17 00:00:00.0=2.0, 2018-09-18 00:00:00.0=1.0}
--全部列的Column Stats收集结果。
ID = 20201126092022636gzm1****
tinyint1:MaxValue:      20
tinyint1:DistinctNum:   4.0
tinyint1:MinValue:      1
tinyint1:NullNum:       1.0
tinyint1:TopK:  {1=1.0, 10=1.0, 20=1.0}
smallint1:MaxValue:     20
smallint1:DistinctNum:  4.0
smallint1:MinValue:     2
smallint1:NullNum:      1.0
smallint1:TopK:         {2=1.0, 7=1.0, 20=1.0}
int1:MaxValue:  7
```

```
int1:DistinctNum:       3.0
int1:MinValue:  4
int1:NullNum:   1.0
int1:TopK:      {4=2.0, 7=1.0}
bigint1:MaxValue:       11111118
bigint1:DistinctNum:    4.0
bigint1:MinValue:       8
bigint1:NullNum:        1.0
bigint1:TopK:   {8=1.0, 2222228=1.0, 11111118=1.0}
double1:MaxValue:       123452.3
double1:DistinctNum:    4.0
double1:MinValue:       12.3
double1:NullNum:        1.0
double1:TopK:   {12.3=1.0, 67892.3=1.0, 123452.3=1.0}
decimal1:MaxValue:      22.4
decimal1:DistinctNum:   4.0
decimal1:MinValue:      2.4
decimal1:NullNum:       1.0
decimal1:TopK:  {2.4=1.0, 12.4=1.0, 22.4=1.0}
decimal2:MaxValue:      52.5
decimal2:DistinctNum:   4.0
decimal2:MinValue:      2.57
decimal2:NullNum:       1.0
decimal2:TopK:  {2.57=1.0, 42.5=1.0, 52.5=1.0}
string1:MaxLength       6.0
string1:AvgLength:      3.0
string1:DistinctNum:    4.0
string1:NullNum:        1.0
string1:TopK:   {str1=1.0, str12=1.0, str123=1.0}
varchar1:MaxLength      6.0
varchar1:AvgLength:     3.0
varchar1:DistinctNum:   4.0
varchar1:NullNum:       1.0
varchar1:TopK:  {str2=1.0, str200=1.0, str21=1.0}
boolean1:DistinctNum:   3.0
boolean1:NullNum:       1.0
boolean1:TopK:  {false=2.0, true=1.0}
timestamp1:DistinctNum:         3.0
timestamp1:NullNum:     1.0
timestamp1:TopK:        {2018-09-17 00:00:00.0=2.0, 2018-09-18 00:00:00.0=1.0}
datetime1:DistinctNum:  3.0
datetime1:NullNum:      1.0
datetime1:TopK:         {1537117199000=2.0, 1537030799000=1.0}
```

> **说明** 更多关于MaxCompute收集元数据的方法介绍，请参见优化器信息收集。

2.15. 其它操作

本文为您介绍使用MaxCompute SQL开发过程中会使用的其它常用命令。

计量预估

基于一条SQL作业的输入数据量大小、UDF个数以及SQL复杂等级估算作业执行费用。需要注意的是，如果UDF涉及分区裁剪，由于无法确定最终会扫描多少个分区，该场景下的费用无法预估。

> **说明** 该信息不能作为实际计费标准，仅供参考，实际费用请以账单为准。

- 命令格式

```
cost sql <sql_sentence>;
```

- 参数说明

  sql_sentence：待估算费用的SQL语句。

- 使用示例

```
cost sql select * from sale_detail;
```

返回结果如下。

```
ID = 20150715113033121xxxxxxx
UDF:0
Complexity:1.0
Input:0 Bytes
```

3.开发参考：SQL

3.1. SQL概述

MaxCompute SQL采用的是类似于SQL的语法。它的语法是标准语法ANSI SQL92的一个子集，并有自己的扩展。本文为您介绍MaxCompute SQL使用场景、使用向导、支持的工具信息，帮助您快速了解MaxCompute SQL，为后续使用MaxCompute SQL提供帮助。

应用场景

MaxCompute SQL适用于海量数据（GB、TB、EB级别）、离线批量计算的场景。提交MaxCompute作业后，会存在几十秒到数分钟不等的排队调度，所以适合处理批作业，提交一次作业批量处理海量数据。不适合直接对接需要每秒处理几千至数万笔事务的前台业务系统。

使用向导

如果您需要...	参考文档	说明
了解MaxCompute SQL	与其他SQL语法的差异	了解与主流数据库的语法差异。
	保留字与关键字	了解MaxCompute SQL命令中的保留字与关键字。
	数据类型	了解MaxCompute SQL支持处理的数据类型、版本及各版本间的区别。
	数据类型转换	了解MaxCompute SQL支持的数据类型转换方式。
	数据类型映射表	了解与主流数据库的数据类型差异。
	转义字符	了解MaxCompute SQL中的转义字符。
	LIKE字符匹配	了解MaxCompute SQL中LIKE匹配支持的字符。
	正则表达式规范	了解MaxCompute SQL支持的正则表达式规则。
	运算符	了解MaxCompute中的关系运算符、算术运算符、位运算符和逻辑运算符。
使用MaxCompute SQL	使用限制	介绍MaxCompute SQL的限制项，助力编写满足规则的脚本。
	DDL语句	介绍MaxCompute SQL的数据定义语法。例如，表、生命周期、分区和列。
	DML语句	介绍MaxCompute SQL中更新数据的语法。例如 `insert into` 、 `insert overwrite` 。
	DQL语句	介绍MaxCompute SQL中查询数据的 `select` 语法。
	SQL增强语法	介绍提升MaxCompute SQL可读性与执行效率的语法。例如 `load` 、 `unload` 。
	MaxCompute UDF	介绍MaxCompute支持的自定义函数，包含UDF（User Defined Scalar Function）、UDTF（User Defined Table Valued Function）和UDAF（User Defined Aggregation Function），并指导您如何创建自定义函数。
	MaxCompute UDT	介绍如何在SQL中直接调用第三方语言的类及方法、或直接使用第三方对象获取数据内容。
	MaxCompute UDJ	介绍如何实现灵活的跨表、多表自定义操作。
	通过脚本模式运行MaxCompute SQL	介绍如何使用脚本模式编译SQL脚本。

相关工具

您可以使用如下工具开发MaxCompute SQL作业：

- 如果开发简单作业，推荐您使用MaxCompute客户端或MaxCompute控制台（查询编辑器）。
- 如果开发流程比较复杂的作业，推荐您使用MaxCompute Studio或DataWorks控制台。

3.2. 与其他SQL语法的差异

本文将MaxCompute SQL与Hive、MySQL、Oracle、SQL Server的SQL语法进行对比，为您介绍MaxCompute不支持的语法。

MaxComputeDDL语法差异对比

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
CREATE TABLE—PRIMARY KEY	N	N	Y	Y	Y
CREATE TABLE—NOT NULL	Y	N	Y	Y	Y
CREATE TABLE—CLUSTER BY	Y	Y	N	Y	Y
CREATE TABLE—EXTERNAL TABLE	Y (OSS, OTS, TDDL)	Y	N	Y	N
CREATE TABLE—TEMPORARY TABLE	N	Y	Y	Y	Y （with #prefix）
INDEX—CREATE INDEX	N	Y	Y	Y	Y
VIRTUAL COLUMN	N	N	N	Y	Y

MaxComputeDML语法差异对比

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
CTE	Y	Y	Y	Y	Y
SELECT—recursive CTE	N	N	N	Y	Y
SELECT—GROUP BY ROLL UP	Y	Y	Y	Y	Y
SELECT—GROUP BY CUBE	Y	Y	N	Y	Y
SELECT—GROUPING SET	Y	Y	N	Y	Y
SELECT—IMPLICT JOIN	Y	Y	N	Y	Y
SELECT—PIVOT	N	N	N	Y	Y
SEMI JOIN	Y	Y	Y	N	N
SELECT TRANSFORM	Y	Y	N	N	N
SELECT—corelated subquery	Y	Y	Y	Y	Y
ORDER BY NULLS FIRST/LAST	N	Y	Y	Y	Y
LATERAL VIEW	Y	Y	N	Y	Y (CROSS APPLY keyword)
SET OPERATOR—UNION (disintct)	Y	Y	Y	Y	Y
SET OPERATOR—INTERSECT	Y	N	N	Y	Y
SET OPERATOR—MINUS/EXCEPT	Y	N	N	Y	Y （keyword EXCEPT）
INSERT INTO ... VALUES	Y	Y	Y	Y	Y
INSERT INTO (ColumnList)	Y	Y	Y	Y	Y

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
UPDATE ... WHERE	Y	Y	Y	Y	Y
DELETE ... WHERE	Y	Y	Y	Y	Y
MERGE INTO	Y	Y	N	Y	Y
ANALYTIC—reusable WINDOWING CLAUSE	Y	Y	N	N	N
ANALYTIC—CURRENT ROW	Y	Y	N	Y	Y
ANALYTIC—UNBOUNDED	Y	N	Y	Y	Y
ANALYTIC—RANGE ...	N	Y	N	Y	Y
WHILE DO	N	N	Y	Y	Y
VIEW WITH PARAMETERS	Y	N	N	N **说明** SQL Function/Storage Procedure一定程度上能满足MaxCompute参数化VIEW的功能。	N

MaxCompute SCRIPTING语法差异对比

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
TABLE VARIABLE	Y	Y	Y	Y	Y
SCALER VARIABLE	Y	Y	Y	Y	Y
ERROR HANDLING—RAISE ERROR	N	N	Y	Y	Y
ERROR HANDLING—TRY CATCH	N	N	N	Y	Y
FLOW CONTROL—LOOP	N	N	Y	Y	Y
CURSOR	N	N	Y	Y	Y

3.3. SQL使用限制项

本文为您介绍MaxCompute SQL的限制项。

限制项	最大值/限制条件	分类	说明
表名长度	128字节	长度限制	表名、列名中不能有特殊字符，以字母开头，且只能用英文小写字母（a-z）、英文大写字母（A-Z）、数字和下划线（_）。
注释长度	1024字节	长度限制	长度不超过1024字节的有效字符串。
表的列定义	1200个	数量限制	单表的列定义个数最多为1200个。
单表分区数	60000个	数量限制	单表的分区个数最多为60000个。
表的分区层级	6级	数量限制	在表中创建的分区层次不能超过6级。
屏显	10000行	数量限制	SELECT语句屏显最多输出10000行。
`INSERT` 目标个数	256个	数量限制	`MULTI-INSERT` 场景，目标表的数量限制为256个。

限制项	最大值/限制条件	分类	说明
UNION ALL	256个	数量限制	UNION ALL 场景，最多允许合并256个表。
MAPJOIN	128个	数量限制	MAPJOIN 场景，最多允许连接128个小表。
MAPJOIN 内存限制	512 MB	数量限制	MAPJOIN 场景，所有小表的内存不能超过512 MB。
ptinsubq	1000行	数量限制	子查询中存在分区列时，子查询的返回结果不能超过1000行。
SQL语句长度	2 MB	长度限制	SQL语句的最大长度为2 MB。包括您使用SDK调用SQL的场景。
WHERE 子句条件个数	256个	数量限制	WHERE 子句中的条件个数最大为256个。
列记录长度	8 MB	数量限制	表中单个单元的最大长度为8 MB。
IN的参数个数	1024	数量限制	IN的最大参数限制，例如 in (1,2,3….,1024) 。如果 in(…) 的参数过多，会影响编译性能。1024为建议值，不是限制值。
jobconf.json	1 MB	长度限制	jobconf.json 的大小为1 MB。当表包含的分区数量较多时，大小可能超过 jobconf.json ，超过1 MB。
视图	不可写	操作限制	视图不支持写入，不支持 INSERT 操作。
列的数据类型	不可修改	操作限制	不允许修改列的数据类型及列位置。
Java UDF函数	不允许为 ABSTRACT 或者 STATIC 。	操作限制	Java UDF函数不能为 ABSTRACT 或 STATIC 。
最多查询分区个数	10000个	数量限制	最多查询分区个数不能超过10000个。
SQL执行计划长度	1 MB	长度限制	MaxCompute SQL生成的执行计划不能超过1 MB，否则会触发 FAILED: ODPS-0010000:System internal error - The Size of Plan is too large 报错。

> 说明 上述MaxCompute SQL限制项均无法修改。

3.4. DDL语句

3.4.1. 表操作

表是MaxCompute的数据存储单元。数据仓库的开发、分析及运维都需要对表数据进行处理。本文为您详细介绍表操作相关命令。

表操作命令如下。

类型	功能	角色	操作入口
创建表	创建非分区表、分区表、外部表或聚簇表。	具备项目创建表权限（CreateTable）的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
修改表的所有人	修改表的所有人，即表Owner。	项目Owner	
修改表的注释	修改表的注释内容。	具备修改表权限（Alter）的用户。	
修改表的修改时间	修改表的 LastDataModifiedTime 为当前时间。		
修改表的聚簇属性	增加或去除表的聚簇属性。		
重命名表	重命名表的名称。		
清空非分区表里的数据	清空指定的非分区表中的数据。		
删除表	删除分区表或非分区表。	具备删除表权限（Drop）的用户。	

类型	功能	角色	操作入口
查看表或视图信息	查看MaxCompute内部表、视图、外部表、聚簇表或Transactional表的信息。	具备读取表元信息权限（Describe）的用户。	
查看分区信息	查看表中分区的详细信息。		
查看建表语句	查看表的SQL DDL语句。		
列出项目下的表和视图	列出项目下所有的表、视图或符合某规则（支持正则表达式）的表、视图。	具备项目查看对象列表权限（List）的用户。	
列出所有分区	列出一张表中的所有分区，表不存在或为非分区表时，返回报错。		

创建表

创建非分区表、分区表、外部表或聚簇表。

- 限制条件
  - 分区表的分区层级不能超过6级。例如某张表以日期为分区列，分区层级为 `年/月/周/日/时/分` 。
  - 一张表允许的分区个数支持按照具体的项目配置，默认为6万个。

  更多表的限制条件，请参见SQL使用限制项。
- 命令格式

```
--创建新表。
 create [external] table [if not exists] <table_name>
 [(<col_name> <data_type> [not null] [default <default_value>] [comment <col_comment>], ...)]
 [comment <table_comment>]
 [partitioned by (<col_name> <data_type> [comment <col_comment>], ...)]
 --用于创建聚簇表时设置表的Shuffle和Sort属性。
 [clustered by | range clustered by (<col_name> [, <col_name>, ...]) [sorted by (<col_name> [asc | desc] [, <col_name> [asc | desc] ...])] into <number_of_buckets> buckets]
 --仅限外部表。
 [stored by StorageHandler]
 --仅限外部表。
 [with serdeproperties (options)]
 --仅限外部表。
 [location <osslocation>]
 --指定表为Transactional表，后续可以对该表执行更新或删除表数据操作，但是Transactional表有部分使用限制，请根据需求创建。
 [tblproperties("transactional"="true")]
 [lifecycle <days>];
--基于已存在的表创建新表并复制数据，但不复制分区属性。
create table [if not exists] <table_name> [lifecycle <days>] as <select_statement>;
--基于已存在的表创建具备相同结构的新表但不复制数据。
create table [if not exists] <table_name> like <existing_table_name> [lifecycle <days>];
```

- 参数说明
  - external：可选。表示创建的表为外部表。
  - if not exists：可选。如果不指定if not exists选项而存在同名表，会报错。如果指定if not exists，只要存在同名表，即使原表结构与要创建的目标表结构不一致，均返回成功。已存在的同名表的元数据信息不会被改动。
  - table_name：必填。表名。表名大小写不敏感，不能有特殊字符，只能包含a~z、A~Z、数字和下划线（_）。建议以字母开头，名称的长度不超过128字节，否则报错。
  - col_name：可选。表的列名。列名大小写不敏感，不能有特殊字符，只能包含a~z、A~Z、数字和下划线（_）。建议以字母开头，名称的长度不超过128字节，否则报错。
  - col_comment：可选。列的注释内容。注释内容为长度不超过1024字节的有效字符串，否则报错。
  - data_type：可选。列的数据类型，包含BIGINT、DOUBLE、BOOLEAN、DATETIME、DECIMAL和STRING等多种数据类型，详情请参见数据类型版本说明。
  - not null：可选。禁止该列的值为NULL。更多修改非空属性信息，请参见修改表的列非空属性。
  - default_value：可选。指定列的默认值，当 `insert` 操作不指定该列时，该列写入默认值。
  - table_comment：可选。表注释内容。注释内容为长度不超过1024字节的有效字符串，否则报错。

- partitioned by (<col_name> <data_type> [comment <col_comment>], ...)：可选。指定分区表的分区字段。
  - col_name：表的分区列名。列名大小写不敏感，不能有特殊字符，只能包含a~z、A~Z、数字和下划线（_）。建议以字母开头，名称的长度不超过128字节，否则报错。
  - data_type：分区列的数据类型。MaxCompute 1.0版本仅支持STRING类型。MaxCompute 2.0版本扩充了分区类型，包含TINYINT、SMALLINT、INT、BIGINT、VARCHAR和STRING类型。详情请参见数据类型版本说明。当使用分区字段对表进行分区时，新增分区、更新分区内数据和读取分区数据均不需要做全表扫描，提高处理效率。
  - col_comment：分区列的注释内容。注释内容为长度不超过1024字节的有效字符串，否则报错。

    说明 分区值不能包含双字节字符（如中文），必须以字母开头，包含字母、数字和允许的字符，长度不超过128字节。允许的字符包括空格、冒号（:）、下划线（_）、美元符号（$）、井号（#）、英文句点（.）、感叹号（!）和at（@），其他字符的行为未定义，例如转义字符 `\t` 、 `\n` 和 `/` 。

- clustered by | range clustered by (<col_name> [, <col_name>, ...]) [sorted by (<col_name> [asc | desc] [, <col_name> [asc | desc] ...])] into <number_of_buckets> buckets：可选。用于创建聚簇表时设置表的Shuffle和Sort属性。

  聚簇表分为Hash聚簇表和Range聚簇表两种：

  - Hash聚簇表
    - clustered by：指定Hash Key。MaxCompute将对指定列进行Hash运算，按照Hash值分散到各个Bucket中。为避免数据倾斜和热点，并取得较好的并行执行效果， `clustered by` 列适宜选择取值范围大，重复键值少的列。此外，为了达到 `join` 优化的目的，也应该考虑选取常用的Join或Aggregation Key，即类似于传统数据库中的主键。
    - sorted by：指定Bucket内字段的排序方式。建议sorted by和clustered by保持一致，以取得较好的性能。此外，当指定sorted by子句后，MaxCompute将自动生成索引，并且在查询时利用索引来加快执行。
    - number_of_buckets：指定哈希桶的数量。该值必须填写，且由数据量大小决定。此外，MaxCompute默认支持最多1111个Reducer，所以此处最多只支持1111个哈希桶。您可以使用 `set odps.sql.reducer.instances=xxx;` 来提升这个限制，但最大不得超过4000，否则会影响性能。

      选择哈希桶数目时，请您遵循以下两个原则：

      - 哈希桶大小适中：建议每个Bucket的大小为500 MB左右。例如，分区大小估计为500 GB，初略估算Bucket数目应该设为1000，这样平均每个Bucket大小约为500 MB。对于特别大的表，500 MB的限制可以突破，每个Bucket在2 GB~3 GB左右比较合适。同时，可以结合 `set odps.sql.reducer.instances=xxx;` 来突破1111个桶的限制。
      - 对于 `join` 优化场景，去除Shuffle和Sort步骤能显著提升性能。因此要求两个表的哈希桶数目成倍数关系，例如256和512。建议哈希桶的数量统一使用2^n，例如512、1024、2048或4096，这样系统可以自动进行哈希桶的分裂和合并，也可以去除Shuffle和Sort的步骤，提升执行效率。
  - Range聚簇表
    - range clustered by：指定范围聚簇列。MaxCompute将对指定列进行分桶运算，按照分桶编号分散到各个Bucket中。
    - sorted by：指定Bucket内字段的排序方式，使用方法与Hash聚簇表相同。
    - number_of_buckets：指定哈希桶的数量。Range聚簇表的Bucket桶数没有Hash聚簇表的2^n最佳实践，在数据量分布合理的情况下，任何桶数都可以。Range聚簇表的Bucket数不是必须的，可以省略，此时系统会根据数据量自动决定最佳的Bucket数目。
    - 当Join或Aggregation的对象是Range聚簇表，且Join Key或Group Key是Range Clustering Key或其前缀时，可以通过控制Flag消除数据的重分布，即Shuffle Remove，提升执行效率。您可以通过 `set odps.optimizer.enable.range.partial.repartitioning=true/flase;` 进行设置，默认关闭。

      说明
      - 聚簇表的优点：
        - 优化Bucket Pruning。
        - 优化Aggregation。
        - 优化存储。
      - 聚簇表的限制：
        - 不支持 `insert into` ，只能通过 `insert overwrite` 来添加数据。
        - 不支持Tunnel直接Upload到Range聚簇表，因为Tunnel上传数据是无序的。

- stored by StorageHandler：可选。按照外部表数据格式指定StorageHandler。
- with serdeproperties (options)：可选。外部表的授权、压缩、字符解析等相关参数。
- osslocation：可选。外部表数据OSS存储位置，详情请参见创建OSS外部表。

- tblproperties("transactional"="true")：可选。设置表为Transactional表。后续可以对Transactional表执行 `update` 、 `delete` 操作实现行级更新或删除数据。更多信息，请参见更新或删除数据（UPDATE|DELETE）。

  Transactional表的使用限制如下：

  - MaxCompute只允许在创建表时设置Transactional属性。已创建的表不允许通过 `alter table` 方式修改Transactional属性，执行如下语句会报错：

    ```
    alter table not_txn_tbl set tblproperties("transactional"="true");
    --报错。
    FAILED: Catalog Service Failed, ErrorCode: 151, Error Message: Set transactional is not supported
    ```

  - 在创建表时，不支持将聚簇表、外部表设置为Transactional表。
  - 不支持MaxCompute内部表、外部表、聚簇表与Transactional表互转。
  - 不支持其他系统的作业（例如MaxCompute Spark、PAI、Graph）访问Transactional表。
  - 不支持 `clone table` 、 `merge partition` 操作。
  - 不支持通过备份与恢复功能备份数据，因此在对Transactional表的重要数据执行 `update` 、 `delete` 或 `insert overwrite` 操作前需要手动通过 `select` + `insert` 操作将数据备份至其他表中。

- lifecycle：可选。表的生命周期，仅支持正整数。单位：天。
  - 非分区表：自最后一次修改表数据开始计算，经过days天后数据无改动，则您无需干预此表，MaxCompute会自动回收（类似drop table操作）。
  - 分区表：系统根据各分区的LastDataModifiedTime判断是否需要回收分区。不同于非分区表，分区表的最后一个分区被回收后，该表不会被删除。生命周期只能设定到表级别，不支持在分区级别设置生命周期。

- 通过 `create table [if not exists] <table_name> [lifecycle <days>] as <select_statement>;` 语句可以再创建一个表，并在建表的同时将数据复制到新表中。但通过该语句创建的表不会复制分区属性，只会把源表的分区列作为目标表的一般列处理，也不会复制源表的生命周期属性。您还可以通过lifecycle参数回收表。
- 通过 `create table [if not exists] <table_name> like <existing_table_name> [lifecycle <days>];` 语句可以再创建一个表，使目标表和源表具有相同的表结构。但通过该语句创建的表不复制数据，也不会复制源表的生命周期属性。您还可以通过lifecycle参数回收表。

- 使用示例
  - 示例1：创建非分区表test1。

    ```
    create table test1 (key STRING);
    ```

  - 示例2：创建一张分区表sale_detail。

    ```
    create table if not exists sale_detail(
     shop_name     STRING,
     customer_id   STRING,
     total_price   DOUBLE)
    partitioned by (sale_date STRING, region STRING);
    ```

  - 示例3：创建一个新表sale_detail_ctas1，将sale_detail的数据复制到sale_detail_ctas1中，并设置生命周期。

    ```
    create table sale_detail_ctas1 lifecycle 10 as select * from sale_detail;
    ```

    您可以通过 `desc extended sale_detail_ctas1;` 命令查看到表的结构及生命周期等详细信息。

    此处 `sale_detail` 是一张分区表，而通过 `create table ... as select_statement ...` 语句创建的表 `sale_detail_ctas1` 不会复制分区属性，只会把源表的分区列作为目标表的一般列处理。即 `sale_detail_ctas1` 是一个含有5列的非分区表。

  - 示例4：创建一个新表sale_detail_ctas2，在 `select` 子句中使用常量作为列的值。

    ```
    --指定列的名字。
    create table sale_detail_ctas2
    as
    select shop_name, customer_id, total_price, '2013' as sale_date, 'China' as region
    from sale_detail;
    --不指定列的名字。
    create table sale_detail_ctas3
    as
    select shop_name, customer_id, total_price, '2013', 'China'
    from sale_detail;
    ```

    > **说明** 如果在 `select` 子句中使用常量作为列的值，建议您指定列的名字。创建的表sale_detail_ctas3的第四、五列类似于 `_c4` 、 `_c5` 。

- 示例5：创建一个新表sale_detail_like，与sale_detail具有相同的表结构，并设置生命周期。

```
create table sale_detail_like like sale_detail lifecycle 10;
```

您可以通过 `desc extended sale_detail_like;` 命令查看到表的结构及生命周期等详细信息。

sale_detail_like的表结构与sale_detail完全相同。除生命周期属性外，列名、列注释以及表注释等均相同。但sale_detail中的数据不会被复制到sale_detail_like表中。

- 示例6：创建使用新数据类型的表test_newtype。

```
set odps.sql.type.system.odps2=true;
CREATE TABLE test_newtype (
    c1 TINYINT
    ,c2 SMALLINT
    ,c3 INT
    ,c4 BIGINT
    ,c5 FLOAT
    ,c6 DOUBLE
    ,c7 DECIMAL
    ,c8 BINARY
    ,c9 TIMESTAMP
    ,c10 ARRAY<MAP<BIGINT,BIGINT>>
    ,c11 MAP<STRING,ARRAY<BIGINT>>
    ,c12 STRUCT<s1:STRING,s2:BIGINT>
    ,c13 VARCHAR(20))
LIFECYCLE 1
;
```

- 示例7：创建Hash聚簇非分区表t1。

```
create table t1 (a STRING, b STRING, c BIGINT) clustered by (c) sorted by (c) into 1024 buckets;
```

- 示例8：创建Hash聚簇分区表t2。

```
create table t2 (a STRING, b STRING, c BIGINT) partitioned by (dt STRING) clustered by (c) sorted by (c) into 1024 buckets;
```

- 示例9：创建Range聚簇非分区表t3。

```
create table t3 (a STRING, b STRING, c BIGINT) range clustered by (c) sorted by (c) into 1024 buckets;
```

- 示例10：创建Range聚簇分区表t4。

```
create table t4 (a STRING, b STRING, c BIGINT) partitioned by (dt STRING) range clustered by (c) sorted by (c);
```

- 示例11：创建Transactional非分区表t5。

```
create table t5(id bigint) tblproperties("transactional"="true");
```

- 示例12：创建Transactional分区表t6。

```
create table if not exists t6(id bigint) partitioned by(ds string) tblproperties ("transactional"="true");
```

修改表的所有人

修改表的所有人，即表Owner。

- 命令格式

```
alter table <table_name> changeowner to <new_owner>;
```

- 参数说明
  - table_name：必填。待修改Owner的表名。
  - new_owner：必填。修改后的Owner账号。
- 使用示例

```
--将表test1的所有人修改为ALIYUN$xxx@aliyun.com。
alter table test1 changeowner to 'ALIYUN$xxx@aliyun.com';
```

修改表的注释

修改表的注释内容。

- 命令格式

```
alter table <table_name> set comment '<new_comment>';
```

- 参数说明
  - table_name：必填。待修改注释的表的名称。

- new_comment：必填。修改后的注释名称。

- 使用示例

```
alter table sale_detail set comment 'new coments for table sale_detail';
```

您可以通过MaxCompute的 `desc table_name` 命令查看表中 `comment` 的修改结果。

修改表的修改时间

MaxCompute SQL提供 `touch` 操作用来修改表的 `LastDataModifiedTime` ，可将表的 `LastDataModifiedTime` 修改为当前时间。此操作会改变表的 `LastDataModifiedTime` 的值，MaxCompute会认为表的数据有变动，生命周期的计算会重新开始。

- 命令格式

```
alter table <table_name> touch;
```

- 参数说明

table_name：必填。待修改表的修改时间的表名称。

- 使用示例

```
alter table sale_detail touch;
```

修改表的聚簇属性

对于分区表，MaxCompute支持通过 `alter table` 语句增加或者去除聚簇属性。

- 命令格式
  - 增加表的Hash聚簇属性的语法格式如下：

```
alter table <table_name> [clustered by (<col_name> [, <col_name>, ...]) [sorted by (<col_name> [asc | desc] [, <col_name> [asc | desc] ...])] into <number_of_buckets> buckets];
```

  - 去除表的Hash聚簇属性的语法格式如下：

```
alter table <table_name> not clustered;
```

  - 增加表的Range聚簇属性，Bucket数不是必须的，可以省略，此时系统会根据数据量自动决定最佳的Bucket数目。语法格式如下：

```
alter table <table_name> [range clustered by (<col_name> [, <col_name>, ...]) [sorted by (<col_name> [asc | desc] [, <col_name> [asc | desc] ...])] into <number_of_buckets> buckets];
```

  - 去除表或分区的Range聚簇属性的语法格式如下：

```
alter table <table_name> not clustered;
alter table <table_name> <pt_spec> not clustered;
```

> **说明**
>
> - 通过 `alter table` 改变聚簇属性，只对分区表有效，非分区表一旦建立聚簇属性就无法改变。 `alter table` 语句适用于存量表，在增加了新的聚簇属性后，新的分区将按设置的聚簇属性存储。
> - `alter table` 只会影响分区表的新建分区（包括 `insert overwrite` 生成的），新分区将按新的聚簇属性存储，老数据分区的聚簇属性和存储保持不变。即在一张曾经做过聚簇属性设置的表上，关闭了聚簇属性，再增加聚簇设置，可以在新分区设置不同于之前的聚簇列、排序列及分桶数。
> - 由于 `alter table` 只影响新分区，所以该语句不可以再指定分区。

- 参数说明

详情请参见创建表。

重命名表

重命名表的名称。仅修改表的名字，不改动表中的数据。

- 命令格式

```
alter table <table_name> rename to <new_table_name>;
```

- 参数说明
  - table_name：必填。待修改名称的表。
  - new_table_name：必填。修改后的表名称。如果已存在与new_table_name同名的表，会返回报错。

- 使用示例

```
alter table sale_detail rename to sale_detail_rename;
```

清空非分区表里的数据

将指定的非分区表中的数据清空。如果您需要清空分区表中单个或多个分区的数据，请参见清空分区数据。

- 命令格式

```
truncate table <table_name>;
```

- 参数说明

  table_name：必填。待清空数据的非分区表的名称。

删除表

删除非分区表或分区表。

- 注意事项
  - 请谨慎操作，确认表可以删除后，再执行删除操作。如果误删了表，当项目开启了备份恢复功能，且删除操作未超过项目设置的备份数据保留天数时，则可以恢复表。更多备份恢复信息，请参见备份与恢复。
  - 删除表之后，MaxCompute项目的存储量会降低。
- 命令格式

```
drop table [if exists] <table_name>;
```

- 参数说明
  - if exists：可选。如果不指定if exists且表不存在，则返回异常。如果指定if exists，无论表是否存在，均返回成功。
  - table_name：必填。待删除的表名。
- 使用示例

```
--删除表sale_detail。无论sale_detail表是否存在，均返回成功。
drop table if exists sale_detail;
```

查看表或视图信息

查看MaxCompute内部表、视图、外部表、聚簇表或Transactional表的信息。如果您需要查看表的详细数据信息，请参见SELECT语法。

- 命令格式

```
--查看表或视图信息。
desc <table_name|view_name> [partition (<pt_spec>)];
--查看外部表、聚簇表或Transactional表信息。也可以查看内部表的扩展信息。
desc extended <table_name>;
```

- 参数说明
  - table_name：必填。待查看表的名称。
  - view_name：必填。待查看视图的名称。
  - pt_spec：可选。待查看分区表的指定分区。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。
  - extended：如果表为外部表、聚簇表或Transactional表，需要包含此参数。显示表的扩展信息。也可以查看内部表的扩展信息，例如列的非空属性。
- 使用示例
  - 示例1：查看创建的test1表的信息。

```
desc test1;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $project_name                      |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 17:47:48                                      |
| LastDDLTime:              2020-11-16 17:47:48                                      |
| LastModifiedTime:         2020-11-16 17:47:48                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                                     |
+------------------------------------------------------------------------------------+
| key             | string     |       |                                             |
+------------------------------------------------------------------------------------+
```

- 示例2：查看创建的sale_detail表的信息。

```
desc sale_detail;
```

返回结果如下。

```
+------------------------------------------------------------------------+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $project_name          |
| TableComment:                                                          |
+------------------------------------------------------------------------+
| CreateTime:               2017-06-28 15:05:17                          |
| LastDDLTime:              2017-06-28 15:05:17                          |
| LastModifiedTime:         2017-06-28 15:05:17                          |
+------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                      |
+------------------------------------------------------------------------+
| Native Columns:                                                        |
+------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                         |
+------------------------------------------------------------------------+
| shop_name       | string     |       |                                 |
| customer_id     | string     |       |                                 |
| total_price     | double     |       |                                 |
+------------------------------------------------------------------------+
| Partition Columns:                                                     |
+------------------------------------------------------------------------+
| sale_date       | string     |                                         |
| region          | string     |                                         |
+------------------------------------------------------------------------+
```

- 示例3：查看创建的sale_detail_ctas1表的详细信息。

```
desc extended sale_detail_ctas1;
```

返回结果如下。

```
+-------------------------------------------------------------------------------------------+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $project_name                             |
| TableComment:                                                                             |
+-------------------------------------------------------------------------------------------+
| CreateTime:               2021-07-07 15:29:53                                             |
| LastDDLTime:              2021-07-07 15:29:53                                             |
| LastModifiedTime:         2021-07-07 15:29:53                                             |
| Lifecycle:                10                                                              |
+-------------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                         |
+-------------------------------------------------------------------------------------------+
| Native Columns:                                                                           |
+-------------------------------------------------------------------------------------------+
| Field    | Type   | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+-------------------------------------------------------------------------------------------+
| shop_name | string |       |               | true     | NULL         |              |
| customer_id | string |       |               | true     | NULL         |              |
| total_price | double |       |               | true     | NULL         |              |
| sale_date | string |       |               | true     | NULL         |              |
| region    | string |       |               | true     | NULL         |              |
+-------------------------------------------------------------------------------------------+
| Extended Info:                                                                            |
+-------------------------------------------------------------------------------------------+
| TableID:                  98cb8a38733c49eabed4735173818147                                |
| IsArchived:               false                                                           |
| PhysicalSize:             0                                                               |
| FileNum:                  0                                                               |
| StoredAs:                 AliOrc                                                          |
| CompressionStrategy:      normal                                                          |
+-------------------------------------------------------------------------------------------+
```

sale_date和region两个字段仅会作为普通列存在，而不是表的分区。

- 示例4：查看创建的sale_detail_ctas2表的信息。

```
desc sale_detail_ctas2;
```

返回结果如下。

```
+------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name           |
| TableComment:                                                          |
+------------------------------------------------------------------------+
| CreateTime:               2017-06-28 15:42:17                          |
| LastDDLTime:              2017-06-28 15:42:17                          |
| LastModifiedTime:         2017-06-28 15:42:17                          |
+------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                      |
+------------------------------------------------------------------------+
| Native Columns:                                                        |
+------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                         |
+------------------------------------------------------------------------+
| shop_name       | string     |       |                                 |
| customer_id     | string     |       |                                 |
| total_price     | double     |       |                                 |
| sale_date       | string     |       |                                 |
| region          | string     |       |                                 |
+------------------------------------------------------------------------+
```

- 示例5：查看创建的sale_detail_like表的详细信息。

```
desc extended sale_detail_like;
```

返回结果如下。

```
+-------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                        |
| TableComment:                                                                       |
+-------------------------------------------------------------------------------------+
| CreateTime:               2021-07-07 15:40:38                                       |
| LastDDLTime:              2021-07-07 15:40:38                                       |
| LastModifiedTime:         2021-07-07 15:40:38                                       |
| Lifecycle:                10                                                        |
+-------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                   |
+-------------------------------------------------------------------------------------+
| Native Columns:                                                                     |
+-------------------------------------------------------------------------------------+
| Field    | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+-------------------------------------------------------------------------------------+
| shop_name | string |       |               | true     | NULL         |              |
| customer_id | string |       |               | true     | NULL         |              |
| total_price | double |       |               | true     | NULL         |              |
+-------------------------------------------------------------------------------------+
| Partition Columns:                                                                  |
+-------------------------------------------------------------------------------------+
| sale_date       | string     |                                                      |
| region          | string     |                                                      |
+-------------------------------------------------------------------------------------+
| Extended Info:                                                                      |
+-------------------------------------------------------------------------------------+
| TableID:                  61782ff7713f426e9d6f91d5deeac99a                          |
| IsArchived:               false                                                     |
| PhysicalSize:             0                                                         |
| FileNum:                  0                                                         |
| StoredAs:                 AliOrc                                                    |
| CompressionStrategy:      normal                                                    |
+-------------------------------------------------------------------------------------+
```

除生命周期属性外，sale_detail_like的其它属性（字段类型、分区类型等）均与sale_detail完全一致。

> **说明** 通过 `desc table_name` 查看到的Size包含了在回收站的数据Size。如果您需要清空回收站，可以先执行 `purge table table_name` ，然后再执行 `desc table_name` 查看除回收站以外的数据大小。您也可以执行 `show recyclebin` 查看本项目中回收站内的数据明细。

- 示例6：查看创建的test_newtype表的信息。

```
desc test_newtype;
```

返回结果如下。

```
| Native Columns:                                                                 |
+---------------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                                  |
+---------------------------------------------------------------------------------+
| c1              | tinyint    |       |                                          |
| c2              | smallint   |       |                                          |
| c3              | int        |       |                                          |
| c4              | bigint     |       |                                          |
| c5              | float      |       |                                          |
| c6              | double     |       |                                          |
| c7              | decimal    |       |                                          |
| c8              | binary     |       |                                          |
| c9              | timestamp  |       |                                          |
| c10             | array<map<bigint,bigint>> |       |                                  |
| c11             | map<string,array<bigint>> |       |                                  |
| c12             | struct<s1:string,s2:bigint> |       |                                |
| c13             | varchar(20) |       |                                          |
+---------------------------------------------------------------------------------+
OK
```

- 示例7：查看创建的Hash聚簇非分区表t1的信息。聚簇属性将显示在Extended Info中。

```
desc extended t1;
```

返回结果如下。

```
+---------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                    |
| TableComment:                                                                   |
+---------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 18:00:56                                   |
| LastDDLTime:              2020-11-16 18:00:56                                   |
| LastModifiedTime:         2020-11-16 18:00:56                                   |
+---------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                               |
+---------------------------------------------------------------------------------+
| Native Columns:                                                                 |
+---------------------------------------------------------------------------------+
| Field    | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+---------------------------------------------------------------------------------+
| a        | string |       |               | true     | NULL         |              |
| b        | string |       |               | true     | NULL         |              |
| c        | bigint |       |               | true     | NULL         |              |
+---------------------------------------------------------------------------------+
| Extended Info:                                                                  |
+---------------------------------------------------------------------------------+
| TableID:                  e6b06f705dc34a36a5b72e5af486cab7                      |
| IsArchived:               false                                                 |
| PhysicalSize:             0                                                     |
| FileNum:                  0                                                     |
| StoredAs:                 AliOrc                                                |
| CompressionStrategy:      normal                                                |
| ClusterType:              hash                                                  |
| BucketNum:                1024                                                  |
| ClusterColumns:           [c]                                                   |
| SortColumns:              [c ASC]                                               |
+---------------------------------------------------------------------------------+
OK
```

- 示例8：查看创建的Hash聚簇分区表t2的信息。聚簇属性将显示在Extended Info中。

```
desc extended t2;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                       |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime: 2017-12-25 11:18:26                                                    |
| LastDDLTime: 2017-12-25 11:18:26                                                   |
| LastModifiedTime: 2017-12-25 11:18:26                                              |
| Lifecycle: 2                                                                       |
+------------------------------------------------------------------------------------+
| InternalTable: YES | Size: 0                                                       |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field | Type   | Label | Comment                                                   |
+------------------------------------------------------------------------------------+
| a     | string |       |                                                           |
| b     | string |       |                                                           |
| c     | bigint |       |                                                           |
+------------------------------------------------------------------------------------+
| Partition Columns:                                                                 |
+------------------------------------------------------------------------------------+
| dt    | string |                                                                   |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
| TableID: 91a3395d3ef64b4d9ee1d2852755                                              |
| IsArchived: false                                                                  |
| PhysicalSize: 0                                                                    |
| FileNum: 0                                                                         |
| ClusterType: hash                                                                  |
| BucketNum: 1024                                                                    |
| ClusterColumns: [c]                                                                |
| SortColumns: [c ASC]                                                               |
+------------------------------------------------------------------------------------+
OK
```

- 示例9：查看Range聚簇非分区表t3的信息。聚簇属性将显示在Extended Info中。

```
desc extended t3;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                       |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 18:01:05                                      |
| LastDDLTime:              2020-11-16 18:01:05                                      |
| LastModifiedTime:         2020-11-16 18:01:05                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field    | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+------------------------------------------------------------------------------------+
| a        | string |       |               | true     | NULL         |              |
| b        | string |       |               | true     | NULL         |              |
| c        | bigint |       |               | true     | NULL         |              |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
| TableID:                  38d170aca2684f4baadbbe1931a6ae1f                         |
| IsArchived:               false                                                    |
| PhysicalSize:             0                                                        |
| FileNum:                  0                                                        |
| StoredAs:                 AliOrc                                                   |
| CompressionStrategy:      normal                                                   |
| ClusterType:              range                                                    |
| BucketNum:                1024                                                     |
| ClusterColumns:           [c]                                                      |
| SortColumns:              [c ASC]                                                  |
+------------------------------------------------------------------------------------+
OK
```

- 示例10：查看Range聚簇分区表t4的信息。聚簇属性将显示在Extended Info中。

```
desc extended t4;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@alibaba-inc.com | Project: $project_name                       |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2020-11-16 19:17:48                                      |
| LastDDLTime:              2020-11-16 19:17:48                                      |
| LastModifiedTime:         2020-11-16 19:17:48                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field     | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+------------------------------------------------------------------------------------+
| a         | string |       |               | true     | NULL         |              |
| b         | string |       |               | true     | NULL         |              |
| c         | bigint |       |               | true     | NULL         |              |
+------------------------------------------------------------------------------------+
| Partition Columns:                                                                 |
+------------------------------------------------------------------------------------+
| dt              | string     |                                                     |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
| TableID:                  6ebc3432e283449188c861427bcd6ee4                         |
| IsArchived:               false                                                    |
| PhysicalSize:             0                                                        |
| FileNum:                  0                                                        |
| StoredAs:                 AliOrc                                                   |
| CompressionStrategy:      normal                                                   |
| ClusterType:              range                                                    |
| BucketNum:                0                                                        |
| ClusterColumns:           [c]                                                      |
| SortColumns:              [c ASC]                                                  |
+------------------------------------------------------------------------------------+
OK
```

- 示例11：查看非分区表t5是否为Transactional表。

> **说明** 推荐您使用MaxCompute客户端查看表是否为Transactional表，需要将MaxCompute客户端升级到0.35.4版本。其他工具可能会存在版本升级未就绪的问题，导致查询结果不显示Transactional信息。

```
desc extended t5;
```

返回结果如下。

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$xxxxx@aliyun.com | Project: $project_name                            |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2021-02-18 10:56:27                                      |
| LastDDLTime:              2021-02-18 10:56:27                                      |
| LastModifiedTime:         2021-02-18 10:56:27                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field     | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+------------------------------------------------------------------------------------+
| id        | bigint |       |               | true     | NULL         |              |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
...
| Transactional:            true                                                     |
+------------------------------------------------------------------------------------+
```

- 示例12：查看分区表t6是否为Transactional表。

  > **说明** 推荐您使用MaxCompute客户端查看表是否为Transactional表，需要将MaxCompute客户端升级到0.35.4版本。其他工具可能会存在版本升级未就绪的问题，导致查询结果不显示Transactional信息。

  ```
  desc extended t6;
  ```

  返回结果如下。

  ```
  +------------------------------------------------------------------------------------+
  | Owner: ALIYUN$xxxxx@test.aliyunid.com | Project: $project_name                     |
  | TableComment:                                                                      |
  +------------------------------------------------------------------------------------+
  | CreateTime:               2021-02-18 15:34:54                                      |
  | LastDDLTime:              2021-02-18 15:34:54                                      |
  | LastModifiedTime:         2021-02-18 15:34:54                                      |
  +------------------------------------------------------------------------------------+
  | InternalTable: YES      | Size: 0                                                  |
  +------------------------------------------------------------------------------------+
  | Native Columns:                                                                    |
  +------------------------------------------------------------------------------------+
  | Field           | Type       | Label | Comment                                     |
  +------------------------------------------------------------------------------------+
  | id              | bigint     |       |                                             |
  +------------------------------------------------------------------------------------+
  | Partition Columns:                                                                 |
  +------------------------------------------------------------------------------------+
  | ds              | string     |                                                     |
  +------------------------------------------------------------------------------------+
  | Extended Info:                                                                     |
  +------------------------------------------------------------------------------------+
  ...
  | Transactional:            true                                                     |
  +------------------------------------------------------------------------------------+
  ```

查看分区信息

查看某个分区表具体的分区的信息。

- 命令格式

  ```
  desc <table_name> partition (<pt_spec>);
  ```

- 参数说明
  - table_name：必填。待查看分区信息的分区表名称。
  - pt_spec：必填。待查看的分区信息。格式为 `partition_col1=col1_value1, partition_col2=col2_value1...` 。对于有多级分区的表，必须指明全部的分区值。
- 使用示例

  ```
  --查询分区表sale_detail的分区信息。
  desc sale_detail partition (sale_date='201310',region='beijing');
  ```

  返回结果如下。

  ```
  +------------------------------------------------------------------------------------+
  | PartitionSize: 2109112                                                             |
  +------------------------------------------------------------------------------------+
  | CreateTime:               2015-10-10 08:48:48                                      |
  | LastDDLTime:              2015-10-10 08:48:48                                      |
  | LastModifiedTime:         2015-10-11 01:33:35                                      |
  +------------------------------------------------------------------------------------+
  OK
  ```

查看建表语句

生成创建表的SQL DDL语句，方便您通过SQL重建Schema。

- 命令格式

  ```
  show create table <table_name>;
  ```

- 参数说明

  table_name：必填。待查看建表语句的表的名称。
- 使用示例

```
--查看表sale_detail的建表语句。
show create table sale_detail;
```

返回结果如下。

```
CREATE TABLE IF NOT EXISTS doc_test_dev.sale_detail(shop_name STRING,customer_id STRING,total_price DOUBLE) PARTITIONED BY (sa
le_date STRING,region STRING) STORED AS ALIORC;
```

列出项目下的表和视图

列出项目下所有的表和视图，或符合某规则的表和视图。

- 命令格式

```
--列出项目下所有的表和视图。
show tables;
--列出项目下表名或视图名与chart匹配的表。
show tables like '<chart>';
```

- 使用示例

```
--列出项目下表名与sale*匹配的表。*表示任意字段。
show tables like 'sale*';
```

返回结果如下。

```
ALIYUN$account_name:sale_detail
......
--ALIYUN是系统提示符，表示您是阿里云主账号用户。如果您是阿里云RAM用户，系统提示符为RAM。
```

列出所有分区

列出一张表中的所有分区，表不存在或为非分区表时，返回报错。

- 命令格式

```
show partitions <table_name>;
```

- 参数说明

  table_name：必填。待查看分区信息的分区表名称。

- 使用示例

```
--列出sale_detail中的所有分区。
show partitions sale_detail;
```

返回结果如下。

```
sale_date=201310/region=beijing
sale_date=201312/region=shenzhen
sale_date=201312/region=xian
sale_date=2014/region=shenzhen
OK
```

3.4.2. 分区和列操作

分区和列操作为您提供了变更MaxCompute中表的分区或列的操作方法，您可以根据实际业务场景执行相应操作。

MaxCompute SQL的分区和列操作命令如下。

类型	操作	功能	角色	操作入口
分区操作	添加分区	为已存在的分区表新增分区。		本文中的命令您可以在如下工具平台执行：
	删除分区	为已存在的分区表删除分区。		
	修改分区的更新时间	修改分区表中分区的 `LastDataModifiedTime` 。		
	修改分区值	修改分区表的分区值。		
	合并分区	对分区表的分区进行合并，即同一个分区表下的多个分区合并成一个分区，同时删除被合并的分区维度的信息，把数据移动到指定分区。		
	清空分区数据	清空指定分区的数据。		

类型	操作	功能	角色 具备修改表权限（Alter）的用户	操作入口 • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
列操作	添加列或注释	为已存在的非分区表或分区表添加列或注释。		
	删除列	删除已存在的非分区表或分区表的列。		
	更改列数据类型	更改已存在列的数据类型。		
	修改列的顺序	调整表中指定列的顺序。		
	修改列名	为已存在的非分区表或分区表修改列名称。		
	修改列注释	为已存在的非分区表或分区表修改列注释。		
	修改列名及注释	为已存在的非分区表或分区表同时修改列名称和列注释。		
	修改表的列非空属性	修改非分区列的非空属性。		

添加分区

为已存在的分区表新增分区。

- 限制条件
  - MaxCompute单表支持的分区数量上限为6万个。
  - 对于有多级分区的表，如果需要添加新的分区值，必须指明全部的分区。
  - 仅支持新增分区值，不支持新增分区字段。
- 命令格式

```
alter table <table_name> add [if not exists] partition <pt_spec> [partition <pt_spec> partition <pt_spec>...];
```

- 参数说明
  - table_name：必填。待新增分区的分区表名称。
  - if not exists：可选。如果未指定if not exists而同名的分区已存在，会执行失败并返回报错。
  - pt_spec：必填。新增的分区，格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。分区字段不区分大小写，分区值区分大小写。
- 使用示例
  - 示例1：给表sale_detail添加一个分区，用来存储2013年12月杭州地区的销售记录。

    ```
    alter table sale_detail add if not exists partition (sale_date='201312', region='hangzhou');
    ```

  - 示例2：给表sale_detail同时添加两个分区，用来存储2013年12月北京和上海地区的销售记录。

    ```
    alter table sale_detail add if not exists partition (sale_date='201312', region='beijing') partition (sale_date='201312', region='shanghai');
    ```

  - 示例3：给表sale_detail添加分区，仅指定一个分区字段sale_date，返回报错，需要同时指定2个分区字段sale_date和region。

    ```
    alter table sale_detail add if not exists partition (sale_date='20111011');
    ```

删除分区

为已存在的分区表删除分区。

MaxCompute支持通过条件筛选方式删除分区。如果您希望一次性删除符合某个规则条件的多个分区，可以使用表达式指定筛选条件，通过筛选条件匹配分区并批量删除分区。

- 限制条件
  - 每个分区过滤子句只能访问一个分区列。
  - 表达式用到的函数必须是内建的Scalar函数。
- 注意事项
  - 删除分区之后，MaxCompute项目的存储量会降低。
  - 您可以结合MaxCompute提供的生命周期功能，实现自动回收旧分区的能力。更多生命周期信息，请参见生命周期。
- 命令格式

- 未指定筛选条件

```
--一次删除一个分区。
alter table <table_name> drop [if exists] partition <pt_spec>;
--一次删除多个分区。
alter table <table_name> drop [if exists] partition <pt_spec>,partition <pt_spec>[,partition <pt_spec>....];
```

- 指定筛选条件

```
alter table <table_name> drop [if exists] partition <partition_filtercondition>;
```

- 参数说明
  - table_name：必填。待删除分区的分区表名称。
  - if exists：可选。如果未指定if exists且分区不存在，则返回报错。
  - pec：必填。删除的分区。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。分区字段不区分大小写，分区值区分大小写。
  - partition_filtercondition：指定筛选条件时必填。分区筛选条件，不区分大小写。格式为：

```
partition_filtercondition
    : partition (<partition_col> <relational_operators> <partition_col_value>)
    | partition (scalar(<partition_col>) <relational_operators> <partition_col_value>)
    | partition (<partition_filtercondition1> AND|OR <partition_filtercondition2>)
    | partition (NOT <partition_filtercondition>)
    | partition (<partition_filtercondition1>)[,partition (<partition_filtercondition2>), ...]
```

    - partition_col：分区名称。
    - relational_operators：关系运算符，详情请参见运算符。
    - partition_col_value：分区列比较值或正则表达式，与分区列数据类型保持一致。
    - scalar()：Scalar函数。Scalar函数基于输入值生成对应的标量，对分区列的值（partition_col）进行处理后再按照指定的关系运算符relational_operators与partition_col_value做比较。
    - 分区过滤条件支持逻辑运算符NOT、AND和OR。支持通过NOT过滤条件子句，取过滤规则的补集。支持多个过滤条件子句以AND或OR的关系组成整体分区匹配规则。
    - 支持多个分区过滤子句，当多个分区过滤子句以英文逗号（,）分隔时，每个过滤子句的逻辑以OR的关系组成整体分区匹配规则。

- 使用示例
  - 未指定筛选条件

```
--从表sale_detail中删除一个分区，2013年12月杭州分区的销售记录。
alter table sale_detail drop if exists partition(sale_date='201312',region='hangzhou');
--从表sale_detail中同时删除两个分区，2013年12月杭州和上海分区的销售记录。
alter table sale_detail drop if exists partition(sale_date='201312',region='hangzhou'),partition(sale_date='201312',region='shanghai');
```

  - 指定筛选条件

```
--创建分区表。
create table if not exists sale_detail(
shop_name     STRING,
customer_id   STRING,
total_price   DOUBLE)
partitioned by (sale_date STRING);
--添加分区。
alter table sale_detail add if not exists
partition (sale_date= '201910')
partition (sale_date= '201911')
partition (sale_date= '201912')
partition (sale_date= '202001')
partition (sale_date= '202002')
partition (sale_date= '202003')
partition (sale_date= '202004')
partition (sale_date= '202005')
partition (sale_date= '202006')
partition (sale_date= '202007');
--批量删除分区。
alter table sale_detail drop if exists partition(sale_date < '201911');
alter table sale_detail drop if exists partition(sale_date >= '202007');
alter table sale_detail drop if exists partition(sale_date LIKE '20191%');
alter table sale_detail drop if exists partition(sale_date IN ('202002','202004','202006'));
alter table sale_detail drop if exists partition(sale_date BETWEEN '202001' AND '202007');
alter table sale_detail drop if exists partition(substr(sale_date, 1, 4) = '2020');
alter table sale_detail drop if exists partition(sale_date < '201912' OR sale_date >= '202006');
alter table sale_detail drop if exists partition(sale_date > '201912' AND sale_date <= '202004');
alter table sale_detail drop if exists partition(NOT sale_date > '202004');
--支持多个分区过滤表达式，表达式之间是OR的关系。
alter table sale_detail drop if exists partition(sale_date < '201911'), partition(sale_date >= '202007');
--添加其他格式分区。
alter table sale_detail add IF NOT EXISTS
partition (sale_date= '2019-10-05')
partition (sale_date= '2019-10-06')
partition (sale_date= '2019-10-07');
--批量删除分区，使用正则表达式匹配分区。
alter table sale_detail drop if exists partition(sale_date RLIKE '2019-\\d+-\\d+');
--创建多级分区表。
create table if not exists region_sale_detail(
shop_name     STRING,
customer_id   STRING,
total_price   DOUBLE)
partitioned by (sale_date STRING , region STRING );
--添加分区。
alter table region_sale_detail add IF NOT EXISTS
partition (sale_date= '201910',region = 'shanghai')
partition (sale_date= '201911',region = 'shanghai')
partition (sale_date= '201912',region = 'shanghai')
partition (sale_date= '202001',region = 'shanghai')
partition (sale_date= '202002',region = 'shanghai')
partition (sale_date= '201910',region = 'beijing')
partition (sale_date= '201911',region = 'beijing')
partition (sale_date= '201912',region = 'beijing')
partition (sale_date= '202001',region = 'beijing')
partition (sale_date= '202002',region = 'beijing');
--执行如下语句批量删除多级分区，两个匹配条件是或的关系，会将sale_date小于201911或region等于beijing的分区都删除掉。
alter table region_sale_detail drop if exists partition(sale_date < '201911'),partition(region = 'beijing');
--如果删除sale_date小于201911且region等于beijing的分区，可以使用如下方法。
alter table region_sale_detail drop if exists partition(sale_date < '201911', region = 'beijing');
```

批量删除多级分区时，在一个 `partition` 过滤子句中，不能根据多个分区列编写组合条件匹配分区，如下语句会报错 `FAILED: ODPS-0130071: [1,82] Semantic analysis exception - invalid column reference region, partition expression must have one and only one column reference` 。

```
--分区过滤子句只能访问一个分区列，如下语句报错。
alter table region_sale_detail drop if exists partition(sale_date < '201911' AND region = 'beijing');
```

修改分区的更新时间

MaxCompute SQL提供 `touch` 操作，用于修改分区表中分区的 `LastDataModifiedTime` 。此操作会将 `LastDataModifiedTime` 修改为当前时间。此时，MaxCompute会认为数据有变动，重新计算生命周期。

- 使用限制

  对于有多级分区的表，必须指明全部的分区。

- 命令格式

```
alter table <table_name> touch partition (<pt_spec>);
```

- 参数说明
  - table_name：必填。待修改分区更新时间的分区表名称。如果表不存在，则返回报错。
  - pt_spec：必填。需要修改更新时间的分区信息。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。如果指定的分区字段或分区值不存在，则返回报错。
- 使用示例

```
--修改表sale_detail的分区sale_date='201312', region='shanghai'的LastDataModifiedTime。
alter table sale_detail touch partition (sale_date='201312', region='shanghai');
```

修改分区值

MaxCompute SQL支持通过 `rename` 操作更改分区表的分区值。

- 使用限制
  - 不支持修改分区列的列名，只能修改分区列对应的值。
  - 对于有多级分区的表，必须指明全部的分区。
- 命令格式

```
alter table <table_name> partition (<pt_spec>) rename to partition (<new_pt_spec>);
```

- 参数说明
  - table_name：必填。待修改分区值的表名称。
  - pt_spec：必填。需要修改分区值的分区信息。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。如果指定的分区字段或分区值不存在，则返回报错。
  - new_pt_spec：必填。修改后的分区信息。格式为 `(partition_col1 = new_partition_col_value1, partition_col2 = new_partition_col_value2, ...)` 。partition_col是分区字段，new_partition_col_value是新分区值。
- 使用示例

```
--修改表sale_detail的分区值。
alter table sale_detail partition (sale_date = '201312', region = 'hangzhou') rename to partition (sale_date = '201310', region = 'beijing');
```

合并分区

MaxCompute SQL提供 `merge partition` 对分区表的分区进行合并，即将同一个分区表下的多个分区合并成一个分区，同时删除被合并的分区维度的信息，把数据移动到指定分区。

- 使用限制
  - 不支持外部表，聚簇表合并后的分区会消除聚簇属性。
  - 一次性合并分区数量限制为4000个。
- 命令格式

```
alter table <table_name> merge [if exists] partition (<predicate>) [, partition(<predicate2>) ...] overwrite partition (<fullpartitionSpec>) [purge];
```

- 参数说明
  - table_name：必填。待合并分区的分区表名称。
  - if exists：可选。如果未指定if exists，且分区不存在，会执行失败并返回报错。如果指定if exists后不存在满足 `merge` 条件的分区，则不生成新分区。如果运行过程中出现源数据被并发修改（包括 `insert` 、 `rename` 或 `drop` ）时，即使指定if exists也会报错。
  - predicate：必填。筛选待合并分区需要满足的条件。
  - fullpartitionSpec：必填。目标分区信息。
  - purge：可选关键字。选择该字段，则会清理session目录，默认清理3天内的日志。详情请参见Purge。
- 使用示例

- 示例1：合并满足指定条件的分区到目标分区。

```
--查看分区表的分区。
show partitions intpstringstringstring;
ds=20181101/hh=00/mm=00
ds=20181101/hh=00/mm=10
ds=20181101/hh=10/mm=00
ds=20181101/hh=10/mm=10
--合并所有满足hh='00'的分区到hh='00', mm='00'中。
alter table intpstringstringstring merge partition(hh='00') overwrite partition(ds='20181101', hh='00', mm='00');
--查看合并后的分区。
show partitions intpstringstringstring;
ds=20181101/hh=00/mm=00
ds=20181101/hh=10/mm=00
ds=20181101/hh=10/mm=10
```

- 示例2：合并指定的多个分区到目标分区。

```
--合并多个指定分区。
alter table intpstringstringstring merge if exists partition(ds='20181101', hh='00', mm='00'), partition(ds='20181101', hh='
10', mm='00'),  partition(ds='20181101', hh='10', mm='10') overwrite partition(ds='20181101', hh='00', mm='00') purge;
--查看分区表的分区。
show partitions intpstringstringstring;
ds=20181101/hh=00/mm=00
```

清空分区数据

清空分区表中指定分区的数据。

MaxCompute支持通过条件筛选方式清空分区数据。如果您希望一次性删除符合某个规则条件的一个或多个分区，可以使用表达式指定筛选条件，通过筛选条件匹配分区并批量清空分区数据。

- 命令格式
  - 未指定筛选条件

```
truncate table <table_name> partition <pt_spec>[, partition <pt_spec>....];
```

  - 指定筛选条件

```
truncate table <table_name> partition <partition_filtercondition>;
```

- 参数说明
  - table_name：必填。待清空分区数据的分区表名称。
  - pt_spec：必填。待清空数据的分区。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。分区字段不区分大小写，分区值区分大小写。
  - partition_filtercondition：指定筛选条件时必填。分区筛选条件，不区分大小写。格式为：

```
partition_filtercondition
    : partition (<partition_col> <relational_operators> <partition_col_value>)
    | partition (scalar(<partition_col>) <relational_operators> <partition_col_value>)
    | partition (<partition_filtercondition1> AND|OR <partition_filtercondition2>)
    | partition (NOT <partition_filtercondition>)
    | partition (<partition_filtercondition1>)[,partition (<partition_filtercondition2>), ...]
```

    - partition_col：分区名称。
    - relational_operators：关系运算符，详情请参见运算符。
    - partition_col_value：分区列比较值或正则表达式，与分区列数据类型保持一致。
    - scalar()：Scalar函数。Scalar函数基于输入值生成对应的标量，对分区列的值（partition_col）进行处理后再按照指定的关系运算符relational_operators与partition_col_value做比较。
    - 分区过滤条件支持逻辑运算符NOT、AND和OR。支持通过NOT过滤条件子句，取过滤规则的补集。支持多个过滤条件子句以AND或OR的关系组成整体分区匹配规则。
    - 支持多个分区过滤子句，当多个分区过滤子句以英文逗号（,）分隔时，每个过滤子句的逻辑以OR的关系组成整体分区匹配规则。

- 使用示例
  - 未指定筛选条件

```
--从表sale_detail中清空一个分区，清空2013年12月杭州地域的销售记录。
truncate table sale_detail partition(sale_date='201312',region='hangzhou');
--从表sale_detail中同时清空两个分区，清空2013年12月杭州和上海地域的销售记录。
truncate table sale_detail partition(sale_date='201312',region='hangzhou'),  partition(sale_date='201312',region='shanghai')
;
```

- 指定筛选条件

```
--从表sale_detail中清空多个分区，清空杭州地域下sale_date以2013开头的销售记录。
truncate table sale_detail partition(sale_date like '2013%' and region='hangzhou');
```

添加列或注释

为已存在的非分区表或分区表添加列或注释。

> 说明 MaxCompute已支持添加STRUCT类型的列，例如 `struct<x: string, y: bigint>` 、 `map<string, struct<x: double, y: double>>` 。该功能正在试用中，试用操作请参见新功能测试申请。

- 命令格式

```
alter table <table_name> add columns (<col_name1> <type1> comment ['<col_comment>'][, <col_name2> <type2> comment '<col_comment>'...]);
```

- 参数说明
  - table_name：必填。待新增列的表名称。添加的新列不支持指定顺序，默认在最后一列。
  - col_name：必填。新增列的名称。
  - type：必填。新增列的数据类型。
  - col_comment：可选。新增列的注释。
- 使用示例
  - 示例1：给表sale_detail添加两个列。

```
alter table sale_detail add columns (customer_name STRING, education BIGINT);
```

  - 示例2：给表sale_detail添加两个列并同时添加列注释。

```
alter table sale_detail add columns (customer_name STRING comment '客户', education BIGINT comment '教育' );
```

  - 示例3：给表sale_detail添加一个复杂数据类型列。

```
alter table sale_detail add columns (region struct<province:string, area:string>);
```

删除列

为已存在的非分区表或分区表删除指定的单个或多个列。

> 说明 该功能正在试用中，试用操作请参见新功能测试申请。
>
> 在下列场景中，如果执行了删除列操作，会使表的读写行为发生变化：
>
> - 作业类型是MapReduce 1.0时，Graph任务无法读写修改的表。
> - CUPID作业只有Spark以下版本可以读表，但是不可以写表：
>   - Spark-2.3.0-odps0.34.0
>   - Spark-3.1.1-odps0.34.0
> - PAI作业可以读表，但不可以写表。
> - Hologres作业在1.3版本之前，Hologres引用修改的表作为外部表时，无法读写该表。
>
> 此外，Streaming Tunnel在写入表时，不可以修改表结构。

- 命令格式

```
alter table <table_name> drop columns <col_name1>[, <col_name2>...];
```

- 参数说明
  - table_name：必填。待删除列的表名称。
  - col_name：必填。待删除的列名称。
- 使用示例

```
--删除表sale_detail的列customer_id。输入yes确认后，即可删除列。
alter table sale_detail drop columns customer_id;
--删除表sale_detail的列shop_name和customer_id。输入yes确认后，即可删除列。
alter table sale_detail drop columns shop_name, customer_id;
```

更改列数据类型

为已存在的列更改数据类型。

> 说明 该功能正在试用中，试用操作请参见新功能测试申请。
>
> 在下列场景中，如果执行了更改列数据类型操作，会使表的读写行为发生变化：
>
> - 作业类型是MapReduce 1.0时，Graph任务无法读写修改的表。
> - CUPID作业只有Spark以下版本可以读表，但是不可以写表：
> - Spark-2.3.0-odps0.34.0
> - Spark-3.1.1-odps0.34.0
> - PAI作业可以读表，但不可以写表。
> - Hologres作业在1.3版本之前，Hologres引用修改的表作为外部表时，无法读写该表。
>
> 此外，Streaming Tunnel在写入表时，不可以修改表结构。

- 命令格式。

```
alter table <table_name> change [column] <old_column_name> <new_column_name> <new_data_type>;
```

- 参数说明。
  - table_name：必填。待修改列数据类型的表名称。
  - old_column_name：必填。待修改列数据类型的列名称。
  - new_column_name：必填。修改列数据类型后的列名称。old_column_name可以与new_column_name保持一致，表示不修改列名称。但是new_column_name不能与除old_column_name之外的列名称相同。
  - new_data_type：必填。待修改的列修改后的数据类型。
- 使用示例。

```
--将mf_evol_t3表的id字段由int转化为bigint
alter table mf_evol_t3 change id id bigint;
--将mf_evol_t3表的id字段类型由bigint转化为string
alter table mf_evol_t3 change column id id string;
```

- 数据类型支持转换表。

> 说明 Y表示支持转换；N表示不支持转换；-表示不涉及；Y（）表示满足括号内的条件支持转换。

数据类型	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	BINARY	FLOAT	DOUBLE	DECIMAL V1	DECIMAL(p2,s2) V2	VARCHAR(n2)	CHAR(n2)	STRING	DATE	DATETIME	TIMESTAMP
BOOLEAN	Y	Y	Y	Y	Y	N	-	-	N	Y	-	-	Y	-	-	-
TINYINT	-	Y	Y	Y	Y	N	-	-	N	Y	-	-	Y	-	-	-
SMALLINT	-	-	Y	Y	Y	N	-	-	N	Y	-	-	Y	-	-	-
INT	-	-	-	Y	Y	N	-	-	N	Y	-	-	Y	-	-	-
BIGINT	-	-	-	-	Y	N	-	-	N	Y	-	-	Y	-	-	-
BINARY	N	N	N	N	N	Y	N	N	N	N	N	N	N	N	N	N
FLOAT	-	-	-	-	-	N	Y	Y	N	Y(truncate to s2)	-	-	-	-	-	-
DOUBLE	-	-	-	-	-	N	-	Y	N	Y(truncate to s2)	-	-	-	-	-	-
DECIMAL V1	N	N	N	N	N	N	N	N	Y	Y(truncate to s2)	-	-	Y	-	-	-
DECIMAL(p1,s1) V2	-	-	-	-	-	N	-	-	N	Y(s1<s2&&(p1-s1)<=(p2-s2))	-	-	Y	-	-	-
VARCHAR(n1)	-	-	-	-	-	N	-	-	N	-	n1<=n2	n1<=n2	Y	-	-	-
CHAR(n1)	-	-	-	-	-	N	-	-	N	-	n1<=n2	n1<=n2	Y	-	-	-
STRING	-	-	-	-	-	N	-	-	N	-	-	-	Y	-	-	-
DATE	-	-	-	-	-	N	-	-	N	-	-	-	-	Y	-	-
DATETIME	-	-	-	-	-	N	-	-	N	-	-	-	-	-	Y	-
TIMESTAMP	-	-	-	-	-	N	-	-	N	-	-	-	-	-	-	Y

修改列的顺序

为已存在的非分区表或分区表修改列顺序。

> 说明 该功能正在试用中，试用操作请参见新功能测试申请。
>
> 在下列场景中，如果执行了更改表的列顺序、添加新列并修改列顺序操作，会使表的读写行为发生变化：
>
> - 作业类型是MapReduce 1.0时，Graph任务无法读写修改的表。
> - CUPID作业只有Spark以下版本可以读表，但是不可以写表：
> - Spark-2.3.0-odps0.34.0
> - Spark-3.1.1-odps0.34.0
> - PAI作业可以读表，但不可以写表。
> - Hologres作业在1.3版本之前，Hologres引用修改的表作为外部表时，无法读写该表。
>
> 此外，Streaming Tunnel在写入表时，不可以修改表结构。

- 命令格式

```
alter table <table_name> change <old_column_name> <new_column_name> <column_type> after <column_name>;
```

- 参数说明
  - table_name：必填。待修改列顺序的表名称。
  - old_column_name：必填。待修改顺序的列的原始名称。
  - new_col_name：必填。修改后的列名称。new_col_name可以与old_column_name保持一致，表示不修改列名称。但new_col_name不能与除old_column_name的之外的列名称相同。
  - column_type：必填。待修改的列的原始数据类型。不可修改。

- column_name：必填。将待调整顺序的列调整至column_name之后。

- 使用示例

```
--修改表sale_detail的列customer_id为customer并位于total_price之后。
alter table sale_detail change customer_id customer string after total_price;
--修改表sale_detail的列customer_id位于total_price之后，不修改列名称。
alter table sale_detail change customer_id customer_id string after total_price;
```

修改列名

为已存在的非分区表或分区表修改列名称。

- 命令格式

```
alter table <table_name> change column <old_col_name> rename to <new_col_name>;
```

- 参数说明
  - table_name：必填。待修改列名的表名称。
  - old_col_name：必填。待修改的列名称。old_col_name必须是已存在的列。
  - new_col_name：必填。修改后的列名称。表中不能有名为new_col_name的列。

- 使用示例

```
--修改表sale_detail的列名customer_name为customer。
alter table sale_detail change column customer_name rename to customer;
```

修改列注释

为已存在的非分区表或分区表修改列注释。

- 语法格式

```
alter table <table_name> change column <col_name> comment '<col_comment>';
```

- 参数说明
  - table_name：必填。待修改列注释的表名称。
  - col_name：必填。待修改注释的列名称。col_name必须是已存在的列。
  - col_comment：必填。修改后的注释信息。注释内容为长度不超过1024字节的有效字符串，否则报错。

- 使用示例

```
--修改表sale_detail的列customer的注释。
alter table sale_detail change column customer comment 'customer';
```

修改列名及注释

修改非分区表或分区表的列名或注释。

- 命令格式

```
alter table <table_name> change column <old_col_name> <new_col_name> <column_type> comment '<col_comment>';
```

- 参数说明
  - table_name：必填。需要修改列名以及注释的表名称。
  - old_col_name：必填。需要修改的列名称。`old_col_name`必须是已存在的列。
  - new_col_name：必填。新的列名称。表中不能有名为`new_col_name`的列。
  - column_type：必填。列的数据类型。
  - col_comment：可选。修改后的注释信息。内容最长为1024字节。

- 使用示例

```
--修改表sale_detail的列名customer_name为customer_newname，注释“客户”为“customer”。
alter table sale_detail change column customer_name customer_newname STRING comment 'customer';
```

修改表的列非空属性

修改表的非分区列的非空属性。即如果表的非分区列值禁止为NULL，您可以通过本命令修改分区列值允许为NULL。

您可以通过`desc extended table_name;`命令查看`Nullable`属性值，判断列的非空属性。如果`Nullable`为`true`，表示允许为NULL；如果`Nullable`为`false`，表示禁止为NULL。

- 使用限制

  修改分区列值允许为NULL后，不可回退，不支持再修改分区列值禁止为NULL，请谨慎操作。

- 命令格式

```
alter table <table_name> change column <old_col_name> null;
```

- 参数说明
  - table_name：必填。待修改列非空属性的表名称。
  - old_col_name：必填。待修改的非分区列的名称。old_col_name必须是已存在的非分区列。
- 使用示例

```
--创建一张分区表，id列禁止为NULL。
create table null_test(id int not null, name string) partitioned by (ds string);
--修改id列允许为NULL。
alter table null_test change column id null;
```

3.4.3. 生命周期操作

MaxCompute提供了表数据生命周期管理功能，方便您释放存储空间，简化回收数据的流程。

生命周期

您可以在创建表时，通过 `lifecycle` 关键字指定生命周期。

在MaxCompute中，每当表的数据被修改后，表的 `LastDataModifiedTime` 将会被更新。MaxCompute会根据每张表的 `LastDataModifiedTime` 以及生命周期的设置来判断是否要回收此表：

- 如果表是非分区表，自最后一次数据被修改开始计算，经过 `days` 天后数据仍未被改动，则此表无需您干预，MaxCompute会自动回收，类似 `drop table` 操作。
- 如果表是分区表，则根据各分区的 `LastDataModifiedTime` 判断该分区是否该被回收。分区表的最后一个分区被回收后，该表不会被删除。

生命周期操作命令如下。

类型	功能	角色	操作入口
创建表的生命周期	在创建表时指定表的生命周期。	具备修改表权限（Alter）的用户	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
修改表的生命周期	修改已存在的分区表或非分区表的生命周期。		
禁止或恢复生命周期	禁止或恢复指定表或分区的生命周期。		

使用限制

- 生命周期只能在表级别设置，不能在分区级别设置。分区表设置生命周期后，生命周期也会在分区级别生效。
- 非分区表不支持取消生命周期，只能修改生命周期。
- 分区表可以取消某个具体分区的生命周期，只能修改表级别的生命周期。

创建表的生命周期

在创建表时指定表的生命周期。

- 命令格式

```
--创建新表。
 create [external] table [if not exists] <table_name>
 [(<col_name> <data_type> [default <default_value>] [comment <col_comment>], ...)]
 [comment <table_comment>]
 [partitioned by (<col_name> <data_type> [comment <col_comment>], ...)]
 --用于创建聚簇表时设置表的Shuffle和Sort属性。
 [clustered by | range clustered by (<col_name> [, <col_name>, ...]) [sorted by (<col_name> [asc | desc] [, <col_name> [asc |
desc] ...])] into <number_of_buckets> buckets]
 --仅限外部表。
 [stored by StorageHandler]
 --仅限外部表。
 [with serdeproperties (options)]
 --仅限外部表。
 [location <osslocation>]
 lifecycle <days>;
--基于已存在的表创建新表。
create table [if not exists] <table_name> [as <select_statement> | like <existing_table_name>];
```

- 参数说明
  - table_name：必填。需要设置生命周期的表名。

- days：必填。修改后的生命周期时间，只能为正整数，单位为天。

  > **说明** 创建表的详细参数说明请参见创建表。

- 使用示例

```
--新建test_lifecycle表，生命周期为100天。
create table test_lifecycle (key string) lifecycle 100;
```

修改表的生命周期

修改已存在的分区表或非分区表的生命周期。

- 命令格式

```
alter table <table_name> set lifecycle <days>;
```

- 参数说明
  - table_name：必填。需要修改生命周期的表名。
  - days：必填。修改后的生命周期时间，只能为正整数，单位为天。
- 使用示例

```
--修改test_lifecycle表，将生命周期设为50天。
alter table test_lifecycle set lifecycle 50;
```

禁止或恢复生命周期

禁止或恢复指定表或分区的生命周期。

- 命令格式

```
alter table <table_name> [<pt_spec>] {enable|disable} lifecycle;
```

- 参数说明
  - table_name：必填。待禁止或恢复生命周期的表的名称。
  - pt_spec：可选。待禁止或恢复生命周期的表的分区信息。格式为 `partition_col1=col1_value1, partition_col2=col2_value1...` 。对于有多级分区的表，必须指明全部的分区值。
  - enable：恢复表或指定分区的生命周期功能。
    - 表及其分区重新参与生命周期回收，默认使用当前表及分区上的生命周期配置。
    - 开启表生命周期前可以修改表及分区的生命周期配置，防止开启表生命周期后因使用之前的配置导致数据被误回收。
  - disable：禁止表或指定分区的生命周期功能。
    - 禁止表本身及其所有分区被生命周期回收，优先级高于恢复表分区生命周期。即当使用 `table disable lifecycle` 时， `pt_spec enable lifecycle` 设置无效。
    - 禁止表的生命周期功能后，表的生命周期配置及其分区的enable和disable标记会被保留。
    - 禁止表的生命周期功能后，仍然可以修改表及分区的生命周期配置。
- 使用示例
  - 示例1：禁止表trans的生命周期功能。

    ```
    alter table trans disable lifecycle;
    ```

  - 示例2：禁止表trans中时间为20141111分区的生命周期功能。

    ```
    alter table trans partition (dt='20141111') disable lifecycle;
    ```

3.4.4. 视图操作

视图（View）是在表之上建立的虚拟表，它的结构和内容都来自表。一个视图可以对应一个表或多个表。如果您想保留查询结果，但不想创建表占用存储，可以通过视图实现。

视图操作命令如下。

类型	功能	角色	操作入口
创建或更新视图	基于查询语句创建视图或更新已存在的视图。	具备项目空间创建表权限（CreateTable）的用户。	本文中的命令您可以在如下工具平台执行：
重命名视图	修改已创建视图的名称。	具备修改表权限（Alter）的用户。	

类型	功能	角色	操作入口
查看视图	查看已创建的视图的信息。	具备读取表元信息权限（Describe）的用户。	• MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
删除视图	删除已创建的视图。	具备删除表权限（Drop）的用户。	
修改视图的所有人	修改已创建的视图的所有人。	具备修改表权限（Alter）的用户。	

创建或更新视图

基于查询语句创建视图或更新已存在的视图。

- 使用限制
  - 视图可以引用其它视图，但不能引用自己，也不能循环引用。
  - 不允许向视图写入数据，例如使用 `insert into` 或者 `insert overwrite` 操作视图。
- 命令格式

```
create [or replace] view [if not exists] <view_name>
    [(<col_name> [comment <col_comment>], ...)]
    [comment <view_comment>]
    as <select_statement>;
```

- 参数说明
  - or replace：可选。更新视图需要携带该字段。
  - if not exists：可选。如果没有指定if not exists，在视图已经存在时用 `create view` 会导致异常。这种情况可以用 `create or replace view` 重建视图，重建后视图本身的权限保持不变。
  - view_name：必填。待创建或更新的视图的名称。
  - col_name：必填。待创建视图包含的列名称。
  - col_comment：可选，待创建视图的列的注释。
  - view_comment：可选。待创建视图的注释。
  - select_statement：必填。 `select` 查询语句，是视图的数据来源，您必须有视图所引用表的读权限。视图只能包含一个有效的 `select` 语句。

    > **说明** 创建或更新好视图后，如果视图引用的表发生了变更，有可能导致视图无法访问，例如删除了引用的表。您需要自行维护引用表及视图之间的对应关系。

- 使用示例
  - 示例1：基于表sale_detail创建视图sale_detail_view。

```
create view if not exists sale_detail_view
(store_name, customer_id, price, sale_date, region)
comment 'a view for table sale_detail'
as select * from sale_detail;
```

  - 示例2：基于表sale_detail更新视图sale_detail_view。

```
create or replace view if not exists sale_detail_view
(store_name, customer_id, price)
comment 'a view for table sale_detail'
as select shop_name, customer_id, total_price from sale_detail;
```

重命名视图

修改已创建视图的名称。

- 命令格式

```
alter view <view_name> rename to <new_view_name>;
```

- 参数说明
  - view_name：必填。需要重命名视图的名称。
  - new_view_name：必填。新的视图名称。如果已存在同名视图，会返回报错。
- 使用示例

```
--将视图view sale_detail_view重命名为market。
alter view sale_detail_view rename to market;
```

查看视图

命令详情请参见查看表或视图信息。

删除视图

删除已创建的视图。

- 命令格式

```
drop view [if exists] <view_name>;
```

- 参数说明
  - if exists：可选。如果视图不存在且没有指定if exists，会返回报错。
  - view_name：必填。待删除的视图的名称。
- 使用示例

```
--删除视图sale_detail_view。
drop view if exists sale_detail_view;
```

修改视图的所有人

修改已创建的视图的所有人，即视图Owner。

- 命令格式

```
alter view <view_name> changeowner to <new_owner>;
```

- 参数说明
  - view_name：必填。待修改Owner的视图的名称。
  - new_owner：必填。修改后的Owner的账号。
- 使用示例

```
--将视图sale_detail_view的所有人修改为ALIYUN$xxx@aliyun.com。
alter view sale_detail_view changeowner to 'ALIYUN$xxx@aliyun.com';
```

3.4.5. 物化视图操作

物化视图（Materialized View）本质是一种预计算，即把某些耗时的操作（例如JOIN、AGGREGATE）的结果保存下来，以便在查询时直接复用，从而避免这些耗时的操作，最终达到加速查询的目的。本文为您介绍物化视图相关命令、语法格式并提供使用示例。

背景信息

视图是一种虚拟表，任何对视图的查询，都会转换为视图SQL语句的查询。而物化视图是一种特殊的物理表，物化视图会存储实际的数据，占用存储资源。更多物化视图计费信息，请参见计费规则。

物化视图适用于如下场景：

- 模式固定、且执行频次高的查询。
- 查询包含非常耗时的操作，比如聚合、连接操作等。
- 查询仅涉及表中的很小部分数据。

物化视图与传统查询的对比如下。

对比项	传统查询方式	物化视图查询方式

对比项	传统查询方式	物化视图查询方式

查询语句	直接使用SQL语句查询数据。
	`select empid, deptname`
	`from emps join depts`
	`on emps.deptno=depts.deptno`
	`where hire_date >= '2018-01-01';`
		您需要创建物化视图，然后基于物化视图查询数据。
		创建物化视图语句如下：
		`create materialized view mv`
		`as select empid, deptname, hire_date`
		`from emps join depts`
		`on emps.deptno=depts.deptno`
		`where hire_date >= '2016-01-01';`
		基于创建的物化视图查询数据：
		`select empid, deptname from mv`
		`where hire_date >= '2018-01-01';`
		如果物化视图开启了查询改写功能，使用如下SQL语句查询数据时会直接从物化视图中查询数据：
		`select empid, deptname`
		`from emps join depts`
		`on emps.deptno=depts.deptno`
		`where hire_date >= '2018-01-01';`
		`--相当于如下语句。`
		`select empid, deptname from mv`
		`where hire_date >= '2018-01-01';`

查询特点	查询涉及读表、JOIN、过滤（where）操作。当源表数据量很大时，查询速度会很慢。操作复杂度较高，运行效率低。	查询涉及读表、过滤操作。不涉及JOIN操作。MaxCompute会自动匹配到最优物化视图，并直接从物化视图中读取数据，从而大大提高查询效率。

物化视图相关操作命令如下。

类型	功能	角色	操作入口
创建物化视图支持分区和聚簇	基于查询语句创建物化视图。	具备项目创建表权限（CreateTable）的用户。	本文中的命令您可以在如下工具平台执行： • MaxCompute客户端 • MaxCompute控制台（查询编辑器） • DataWorks控制台 • MaxCompute Studio
更新物化视图	更新已创建的物化视图。	具备修改表权限（Alter）的用户。	
修改物化视图的生命周期	修改已创建的物化视图的生命周期。	具备修改表权限（Alter）的用户。	
开启或禁用物化视图的生命周期	开启或禁用已创建的物化视图的生命周期。	具备修改表权限（Alter）的用户。	
查询物化视图信息	查询物化视图的基本信息。	具备读取表元信息权限（Describe）的用户。	
查询物化视图状态	查看物化视图有效或无效。	具备读取表元信息权限（Describe）的用户。	
删除物化视图	删除已创建的物化视图。	具备删除表权限（Drop）的用户。	
删除物化视图分区	删除已创建的物化视图的分区。	具备删除表权限（Drop）的用户。	
物化视图支持查询穿透	当查询的分区数据不存在时，需要自动实现到原始分区表去查询数据。	具备项目写权限（Write）及创建表权限（CreateTable）的用户。	

使用限制

物化视图的使用限制如下：

- 不支持窗口函数。
- 不支持UDTF函数。
- 默认不支持非确定性函数（例如UDF、UDAF等）。当您的业务场景必须要使用非确定性函数时，请在Session级别设置属性 `set odps.sql.materialized.view.support.nondeterministic.function=true;` 。

创建物化视图支持分区和聚簇

基于满足物化视图场景的数据创建物化视图，支持分区和聚簇场景。

- 使用限制
  - 物化视图的名称不允许和当前已有表、视图、物化视图名称重复。您可以通过 `show tables;` 命令查看项目中的全部表及物化视图名称。
  - 不支持基于存在的物化视图创建新的物化视图。
  - 不支持基于外部表创建物化视图。
- 注意事项
  - 当查询语句执行失败时，物化视图也会创建失败。
  - 物化视图分区列必须来源于某张源表，其顺序和列数目必须和源表一样，列名称可以不一样。
  - 列注释需要指定所有列，包含分区列。如果只指定部分列，会报错。
  - 可以同时指定分区和聚簇，此时每个分区中的数据都有指定的聚簇属性
  - 当查询语句中包含不支持的算子时会报错。物化视图支持的算子列表，请参见基于物化视图执行查询改写操作。
  - MaxCompute默认不支持使用非确定性函数（例如UDF、UDAF等）创建物化视图。当您的业务场景必须要使用非确定性函数时，请在Session级别设置属性 `set odps.sql.materialized.view.support.nondeterministic.function=true;` 。
- 命令格式

```
set odps.sql.materialized.view.initial.partition={"分区列": "分区值"};
--可选项，指定初始分区，避免生成过多无用分区。
create materialized view [if not exists] [<project_name>.]<mv_name>
[lifecycle <days>]
[(<col_name> [comment <col_comment>], ...)]
[{disable|enable rewrite]
[partitioned on (<col_name> [, <col_name>, ...])
[clustered by|range clustered by (<col_name> [, <col_name>, ...])
     [sorted by (<col_name> [asc | desc] [, <col_name> [asc | desc] ...])]
           into <number_of_buckets> buckets]
[tblproperties("compressionstrategy"="normal/high/extreme","enable_auto_substitute"="true")]
as <select_statement>;
```

- 参数说明
  - if not exists：可选。如果没有指定if not exists且物化视图已经存在会返回报错。
  - project_name：可选。物化视图所属目标MaxCompute项目名称。不填写时表示当前所在MaxCompute项目。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
  - mv_name：必填。新建物化视图的名称。
  - days：可选。指定物化视图的生命周期，单位为天。
  - col_name：可选。指定物化视图的列名称。
  - col_comment：可选。指定物化视图的列的注释。
  - disable|enable rewrite：可选。设置禁止或允许通过物化视图执行查询改写操作。不指定时表示允许通过物化视图执行查询改写操作，您可以执行 `alter materialized view [project_name.]<mv_name> disable rewrite;` 命令禁止通过物化视图执行查询改写操作。同理您可以执行 `alter materialized view [project_name.]<mv_name> enable rewrite;` 命令允许通过物化视图执行查询改写操作。
  - partitioned on：可选。指定物化视图分区字段，表示创建的物化视图表为分区表。
  - clustered by|range clustered by：可选。用于创建聚簇表时设置表的Shuffle属性。
  - sorted by：可选。用于创建聚簇表时设置表的Sort属性。
  - number_of_buckets：可选。用于创建聚簇表时设置表分桶数。
  - tblproperties：可选。compressionstrategy指定表数据的存储压缩策略，可以选normal、high或extreme。enable_auto_substitute指定当分区不存在时是否自动穿透到原始分区表去查询数据，详细信息请参见物化视图支持查询穿透。
  - select_statement：必填。查询语句，详细格式请参见SELECT语法。
- 使用示例
  - 示例一：创建以ds为分区列的物化视图。命令示例如下。

```
create materialized view mv lifecycle 7
(
  key comment 'unique id',
  ds comment 'date'
)
partitioned on (ds)
as select t1.id, t1.value, t1.ds as ds from t1 join t2 on t1.id = t2.id;
```

- 示例二：创建带有聚簇属性的非分区物化视图。命令示例如下。

```
create materialized view mv lifecycle 7
clustered by id sorted by value into 1024 buckets
as select t1.id, t1.value, t1.ds as ds from t1 join t2 on t1.id = t2.id;
```

- 示例三：创建带有聚簇属性的分区物化视图。命令示例如下。

```
create materialized view mv lifecycle 7
partitioned on (ds)
clustered by id sorted by value into 1024 buckets
as select t1.id, t1.value, t1.ds as ds from t1 join t2 on t1.id = t2.id;
```

更新物化视图

当物化视图的数据对应的表或分区产生插入、覆写、更新、删除等操作时，物化视图会自动失效，无法用于查询改写。您可以先查看物化视图状态，当物化视图失效时，执行更新操作。查看物化视图状态操作，请参见查询物化视图状态。

- 注意事项
  - 物化视图的更新操作只支持全量更新，不支持增量更新。
  - 您可以借助于DataWorks调度能力实现定时更新。DataWorks的调度操作信息，请参见调度配置。
- 命令格式

```
alter materialized view [<project_name>.]<mv_name> rebuild [partition(<expression1>, <expressio2>...)];
```

- 参数说明
  - project_name：可选。物化视图所属目标MaxCompute项目名称。不填写时表示当前所在MaxCompute项目。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
  - mv_name：必填。待更新物化视图的名称。
  - expression：可选。当更新分区物化视图时，需要指定待更新分区信息，支持表达式。
- 使用示例
  - 示例一：更新非分区物化视图。命令示例如下。

```
alter materialized view count_mv rebuild;
```

  - 示例二：更新分区物化视图的某个分区。命令示例如下。

```
alter materialized view mv rebuild partition (ds='20210101');
```

  - 示例三：更新分区物化视图的满足指定条件的分区。命令示例如下。

```
alter materialized view mv rebuild partition(ds>='20210101', ds<='20210105');
```

修改物化视图的生命周期

修改已创建的物化视图的生命周期。

- 命令格式

```
alter materialized view [<project_name>.]<mv_name> set lifecycle <days>;
```

- 参数说明
  - project_name：可选。物化视图所属目标MaxCompute项目名称。不填写时表示当前所在MaxCompute项目。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
  - mv_name：必填。待更新物化视图的名称。
  - days：必填。设置物化视图的新生命周期。单位为天。
- 使用示例

```
--修改物化视图的生命周期为10天。
alter materialized view mv set lifecycle 10;
```

开启或禁用物化视图的生命周期

开启或禁用已创建的物化视图的生命周期。

- 命令格式

```
alter materialized view [<project_name>.]<mv_name> [<pt_spec>] enable|disable lifecycle;
```

- 参数说明
  - project_name：可选。物化视图所属目标MaxCompute项目名称。不填写时表示当前所在MaxCompute项目。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
  - mv_name：必填。待开启或禁用生命周期的物化视图的名称。

- pt_spec：可选。待开启或禁用生命周期的物化视图的分区。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。
- enable|disable：必填。 `enable` 代表开启， `disable` 代表禁用，禁用后该分区或表就不涉及生命周期管理。

- 使用示例
  - 示例一：开启物化视图的生命周期管理。命令示例如下。

    ```
    alter materialized view mv partition (ds='20210101') enable lifecycle;
    ```

  - 示例二：禁用物化视图的生命周期管理。命令示例如下。

    ```
    alter materialized view mv partition (ds='20210101') disable lifecycle;
    ```

查询物化视图信息

查看物化视图的结构、修改时间等信息。

- 命令格式

```
desc extended [<project_name>.]<mv_name>;
```

- 参数说明
  - project_name：可选。物化视图所属目标MaxCompute项目名称。不填写时表示当前所在MaxCompute项目。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
  - mv_name：必填。待查询物化视图的名称。

- 使用示例

```
desc extended mv;
```

返回结果如下：

```
+------------------------------------------------------------------------------------+
| Owner: ALIYUN$****@aliyunid.com | Project: ****                                    |
| TableComment:                                                                      |
+------------------------------------------------------------------------------------+
| CreateTime:               2021-08-01 17:50:15                                      |
| LastDDLTime:              2021-08-01 17:50:15                                      |
| LastModifiedTime:         2021-08-01 17:50:15                                      |
+------------------------------------------------------------------------------------+
| InternalTable: YES      | Size: 0                                                  |
+------------------------------------------------------------------------------------+
| Native Columns:                                                                    |
+------------------------------------------------------------------------------------+
| Field    | Type    | Label | ExtendedLabel | Nullable | DefaultValue | Comment      |
+------------------------------------------------------------------------------------+
| page_id  | string |       |               | true     | NULL         |              |
| _c1      | bigint |       |               | true     | NULL         |              |
+------------------------------------------------------------------------------------+
| Extended Info:                                                                     |
+------------------------------------------------------------------------------------+
| TableID:                  e4a7f1169588400ab39bc3076426****                         |
| IsArchived:               false                                                    |
| PhysicalSize:             0                                                        |
| FileNum:                  0                                                        |
+------------------------------------------------------------------------------------+
```

查询物化视图状态

查询物化视图状态，以便及时知晓源表变更，确保物化视图有效。物化视图状态分为如下两种：

- 物化视图有效

  执行查询语句时，MaxCompute会从物化视图中直接查询数据，不会从源数据中查询数据。

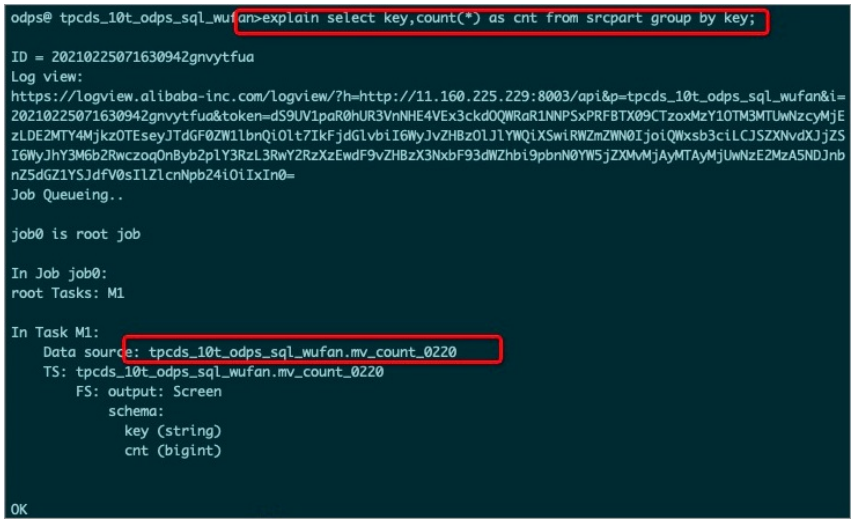
- 物化视图无效

  执行查询语句时，MaxCompute无法从物化视图中直接查询数据，会从源数据中查询数据，无法实现查询加速。

MaxCompute暂不支持通过命令方式查询物化视图状态，您可以通过如下方式查看查询语句中是否使用了物化视图来判断物化视图是否有效：

- 方式一：通过EXPLAIN命令查看。

  如下图所示，在Data source中出现物化视图证明查询使用了物化视图。



- 方式二：通过Logview的作业执行图或**Json Summary**页签查看。

  如果您可以在作业执行图或**Json Summary**页签搜索到物化视图名称，表明物化视图有效；否则物化视图失效。

删除物化视图

删除已创建的物化视图。

- 命令格式

```
drop materialized view [if exists] [<project_name>.]<mv_name>;
```

- 参数说明
  - if exists：可选。如果没有指定if exists且物化视图不存在会返回报错。
  - project_name：可选。物化视图所属目标MaxCompute项目名称。不填写时表示当前所在MaxCompute项目。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
  - mv_name：必填。待删除物化视图的名称。
- 使用示例

```
drop materialized view mv;
```

删除物化视图分区

删除已创建的物化视图的单个或多个分区。

- 命令格式

```
alter materialized view [<project_name>.]<mv_name> drop [if exists] partition <pt_spec> [partition <pt_spec>, partition <pt_sp
ec>....];
```

- 参数说明
  - project_name：可选。物化视图所属目标MaxCompute项目名称。不填写时表示当前所在MaxCompute项目。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
  - mv_name：必填。待删除分区的分区物化视图的名称。
  - if exists：可选。如果没有指定if exists且物化视图不存在会返回报错。
  - pt_spec：至少要指定一个分区。待删除的分区。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。
- 使用示例
  - 示例一：删除分区物化视图的某个分区。命令示例如下。

    ```
    alter materialized view mv drop partition (ds='20210101');
    ```

  - 示例二：删除分区物化视图的满足指定条件的分区。命令示例如下。

    ```
    alter materialized view mv drop partition (ds>='20210101', ds<='20210105');
    ```

基于物化视图执行查询改写操作

物化视图最重要的作用就是对查询语句进行查询改写，如果期望查询语句能利用物化视图进行查询改写，则需要在查询语句前添加 `set odps.sql.materialized.view.enable.auto.rewriting=true;` 配置。当物化视图处于失效状态时不支持查询改写，查询语句会直接查询源表而无法获得加速作用。

> 说明 默认每个MaxCompute项目只能利用自身的物化视图进行查询改写，如果需要利用其他项目中的物化视图进行改写，您需要在查询语句前添加 `set odps.sql.materialized.view.source.project.white.list=<project_name1>,<project_name2>,<project_name3>;` 配置指定其他MaxCompute项目列表。

MaxCompute中物化视图的查询改写支持的算子类型及与其他产品的对照关系如下。

算子类型	分类	MaxCompute	BigQuery	Amazon RedShift	Hive
FILTER	表达式完全匹配	支持	支持	支持	支持
	表达式部分匹配	支持	支持	支持	支持
AGGREGATE	单个AGGREGATE	支持	支持	支持	支持
	多个AGGREGATE	不支持	不支持	不支持	不支持
JOIN	JOIN类型	INNER JOIN	不支持	INNER JOIN	INNER JOIN
	单个JOIN	支持	不支持	支持	支持
	多个JOIN	支持	不支持	支持	支持
AGGREGATE+JOIN	-	支持	不支持	支持	支持

使用物化视图查询改写的原则是查询语句中需要的数据必须从物化视图中得到，包括输出列、筛选条件中需要的列、聚合函数需要的列、JOIN条件需要的列。如果查询语句中需要的列不包含在物化视图中或聚合函数不支持，则无法基于物化视图进行查询改写。具体示例如下：

- 改写带过滤条件的查询语句。假设创建的物化视图如下。

  ```
  create materialized view mv as select a,b,c from src where a>5;
  ```

  基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select a,b from src where a>5;`	`select a,b from mv;`
`select a, b from src where a=10;`	`select a,b from mv where a=10;`
`select a, b from src where a=10 and b=3;`	`select a,b from mv where a=10 and b=3;`
`select a, b from src where a>3;`	`(select a,b from src where a>3 and a<=5) union (select a,b from mv);`
`select a, b from src where a=10 and d=4;`	改写不成功，因为mv中没有d列。
`select d, e from src where a=10;`	改写不成功，因为mv中没有d、e列。
`select a, b from src where a=1;`	改写不成功，因为mv中没有a=1的数据。

- 改写带聚合函数的查询语句
  - 如果物化视图的SQL语句和查询语句的聚合Key相同，那么所有聚合函数都可以改写，如果聚合Key不相同，只支持SUM、MIN和MAX。

    假设创建的物化视图如下。

    ```
    create materialized view mv as
    select a, b, sum(c) as sum, count(d) as cnt from src group by a, b;
    ```

    基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select a, sum(c) from src group by a;`	`select a, sum(sum) from mv group by a;`
`select a, count(d) from src group by a, b;`	`select a, cnt from mv;`
`select a, count(b) from (select a, b from src group by a, b) group by a;`	`select a,count(b) from mv group by a;`
`select a, count(b) from src group by a;`	改写不成功，视图对a、b列进行过聚合，不能再对b进行聚合。
`select a, count(c) from src group by a;`	改写不成功，对于COUNT函数不支持重新聚合。

- 如果聚合函数中有 `distinct` ，当物化视图语句和查询语句聚合Key相同，可以改写，否则不可以改写。

  假设创建的物化视图如下。

  ```
  create materialized view mv as
  select a, b, sum(distinct c) as sum, count(distinct d) as cnt from src group by a, b;
  ```

  基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select a, count(distinct d) from src group by a, b;`	`select a, cnt from mv;`
`select a, count(c) from src group by a, b;`	改写不成功，对于COUNT函数不支持重新聚合。
`select a, count(distinct c) from src group by a;`	改写不成功，因为需要对a再进行聚合。

- 改写带 `join` 的查询语句
  - 改写 `join` 输入

    假设创建的物化视图如下。

    ```
    create materialized view mv1 as select a, b from j1 where b > 10;
    create materialized view mv2 as select a, b from j2 where b > 10;
    ```

    基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select j1.a,j1.b,j2.a from (select a,b from j1 where b > 10) j1 join j2 on j1.a=j2.a;`	`select mv1.a, mv1.b, j2.a from mv1 join j2 on mv1.a=j2.a;`
`select j1.a,j1.b,j2.a from(select a,b from j1 where b > 10) j1join(select a,b from j2 where b > 10) j2on j1.a=j2.a;`	`select mv1.a,mv1.b,mv2.a from mv1 join mv2 on mv1.a=mv2.a;`

  - `join` 带过滤条件

    假设创建的物化视图如下。

    ```
    --创建非分区物化视图。
    create materialized view mv1 as select j1.a, j1.b from j1 join j2 on j1.a=j2.a;
    create materialized view mv2 as select j1.a, j1.b from j1 join j2 on j1.a=j2.a where j1.a > 10;
    --创建分区物化视图。
    create materialized view mv lifecycle 7 partitioned on (ds) as select t1.id, t1.ds as ds from t1 join t2 on t1.id = t2.id;
    ```

    基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select j1.a,j1.b from j1 join j2 on j1.a=j2.a where j1.a=4;`	`select a, b from mv1 where a=4;`
`select j1.a,j1.b from j1 join j2 on j1.a=j2.a where j1.a > 20;`	`select a,b from mv2 where a>20;`
`select j1.a,j1.b from j1 join j2 on j1.a=j2.a where j1.a > 5;`	`(select j1.a,j1.b from j1 join j2 on j1.a=j2.a where j1.a > 5 and j1.a <= 10) union select * from mv2;`
`select key from t1 join t2 on t1.id= t2.id where t1.ds='20210306';`	`select key from mv where ds='20210306';`
`select key from t1 join t2 on t1.id= t2.id where t1.ds>='20210306';`	`select key from mv where ds>='20210306';`
`select j1.a,j1.b from j1 join j2 on j1.a=j2.a where j2.a=4;`	改写不成功，因为物化视图没有j2.a列。

- `join` 增加表

  假设创建的物化视图如下。

  ```
  create materialized view mv as select j1.a, j1.b from j1 join j2 on j1.a=j2.a;
  ```

  基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select j1.a, j1.b from j1 join j2 join j3 on j1.a=j2.a and j1.a=j3.a;`	`select mv.a, mv.b from mv join j3 on mv.a=j3.a;`
`select j1.a, j1.b from j1 join j2 join j3 on j1.a=j2.a and j2.a=j3.a;`	`select mv.a,mv.b from mv join j3 on mv.a=j3.a;`

  > 说明
  > - 以上三种语句可以相互结合，如果查询语句符合改写条件，则可以改写。
  > - MaxCompute会选择最佳的改写规则运行，如果改写后增加了一些操作，不是最优运行计划，最终也不会被选中。

- 改写带 `left join` 的查询语句

  假设创建的物化视图如下。

  ```
  create materialized view mv lifecycle 7(
          user_id,
          job,
          total_amount
  ) as select t1.user_id, t1.job, sum(t2.order_amout) as total amount
       from user_info as t1 left join sale_order as t2 on t1.user_id=t2.user_id group by t1.user_id;
  ```

  基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select t1.user_id, sum(t2.order_amout) as total amount from user_info as t1 left join sale_order as t2 on t1.user_id=t2.user_id group by t1.user_id;`	`select user_id, total_amount from mv;`

- 改写带 `union all` 的查询语句

  假设创建的物化视图如下。

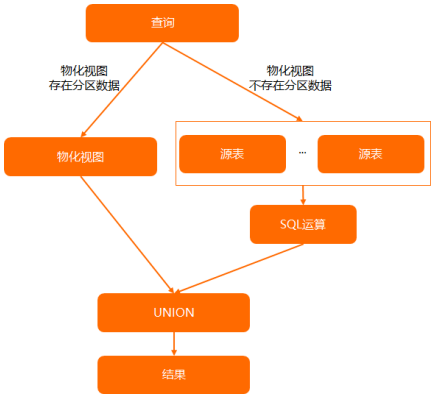
  ```
  create materialized view mv lifecycle 7(
          user_id,
          tran_amount,
          tran_date
  ) as (select user_id, tran_amount, tran_date from alipay_tran union all
   select user_id, tran_amount, tran_date from unionpay_tran);
  ```

  基于创建的物化视图执行查询语句，查询改写对照如下。

原始查询语句	改写后的查询语句
`select user_id, tran_amount from alipay_tran union all select user_id, tran_amount from unionpay_tran;`	`select user_id, total_amount from mv;`

物化视图支持查询穿透

对于分区物化视图，不一定所有分区都有数据，可能只刷新了最新的一些分区数据。但用户查询数据时，实际并不知道查询的所有分区数据都存在，当查询的分区数据不存在时，需要自动实现到原始分区表去查询数据，流程如下图所示。



如果需要物化视图支持穿透查询能力，您需要设置如下参数：

- 在Session级别设置 `set odps.optimizer.cbo.rule.filter.black=re;` ，分区列会被折叠成常量。
- 创建物化视图时，在tblproperties属性中添加 `"enable_auto_substitute"="true"` 配置。

物化视图支持穿透查询示例如下。

1. 创建物化视图支持分区并且支持查询穿透。命令示例如下。

```
--将分区列折叠为常量，实现分区动态可变。
set odps.optimizer.cbo.rule.filter.black=re;
--初始化20210101分区数据。
set odps.sql.materialized.view.initial.partition={"dt": "20210101"};
--创建分区物化视图支持穿透。
create materialized view if not exists mv lifecycle 7
tblproperties("enable_auto_substitute"="true")
partitioned on (dt)
as select id, name, dt from src;
```

2. 查询表src中的分区为20210101的数据。命令示例如下。

```
select * from mv where dt='20210101';
```

3. 查询物化视图mv中的分区为20210102的数据，自动穿透到源表查询数据。命令示例如下。

```
select * from mv where dt = '20210102';
--因为20210102的数据没有物化，则需要把查询转化到对应的源表，等价于
select * from (select id, name, dt from src where dt='20210102') t;
```

4. 查询物化视图mv中分区为20201230~20210102的数据，自动穿透到源表查询的数据与物化视图的数据执行UNION操作后再返回结果。

```
select * from mv where dt >= '20201230' and dt<='20210102' and id='5';
--因为20210102的数据没有物化，则需要把查询转化到对应的源表。等价于：
select * from
(select id, name, dt from src where dt='20211231' or dt='20210102';
 union all
 select * from mv where dt='20210101'
) t where id = '5';
```

使用示例

场景：假设有一张页面访问表visit_records，记录了各个用户访问的页面ID、用户ID、访问时间。用户经常要对不同页面的访问量进行查询分析。visit_records的结构如下。

```
+------------------------------------------------------------------------------------+
| Field           | Type       | Label | Comment                                     |
+------------------------------------------------------------------------------------+
| page_id         | string     |       |                                             |
| user_id         | string     |       |                                             |
| visit_time      | string     |       |                                             |
+------------------------------------------------------------------------------------+
```

此时，可以给visit_records表创建一个以页面ID分组，统计各个页面访问次数的物化视图，并基于物化视图执行后续查询操作：

1. 执行如下语句创建物化视图。

```
create materialized view count_mv as select page_id, count(*) from visit_records group by page_id;
```

2. 执行查询语句如下。

```
set odps.sql.materialized.view.enable.auto.rewriting=true;
select page_id, count(*) from visit_records group by page_id;
```

执行该查询语句时，MaxCompute能自动匹配到物化视图count_mv，从count_mv中读取聚合好的数据。

3. 执行如下命令检验查询语句是否匹配到物化视图。

```
explain select page_id, count(*) from visit_records group by page_id;
```

返回结果如下。

```
job0 is root job
In Job job0:
root Tasks: M1
In Task M1:
    Data source: doc_test_dev.count_mv
    TS: doc_test_dev.count_mv
        FS: output: Screen
            schema:
              page_id (string)
              _c1 (bigint)
OK
```

从返回结果中的Data source可查看到当前查询读取的表是count_mv，说明物化视图有效，查询改写成功。

计费规则

物化视图费用包含如下两部分：

- 存储费用

  物化视图会占用物理存储空间，会产生存储费用。

  - 当MaxCompute项目的规格类型为包年包月标准版或按量计费标准版时，存储费用按量计费。更多计费信息，请参见存储费用。
  - 当MaxCompute项目的规格类型为包年包月套餐版时，超出存储限制部分按量计费。更多计费信息，请参见套餐计费。

- 计算费用

  创建、更新、查询物化视图及查询改写（物化视图有效）过程中涉及到查询数据，会消耗计算资源产生计算费用。

  - 当MaxCompute项目的规格类型为包年包月标准版或包年包月套餐版时，不单独收费。
  - 当MaxCompute项目的规格类型为按量计费标准版时，按照SQL复杂度及输入数据量计算费用。更多计费信息，请参见SQL作业按量计费。您需要注意如下信息：
    - 更新物化视图执行的SQL与创建物化视图执行的SQL相同，费用取决于执行SQL时输入的数据量和复杂度。
    - 当物化视图处于生效状态时查询改写会从物化视图中读取数据，查询语句的输入数据量（从物化视图读取部分）与物化视图相关，与物化视图源表无关。当物化视图处于失效状态时不支持查询改写，查询语句会直接查询源表，查询语句的输入数据量与源表相关。更多查询物化视图状态信息，请参见查询物化视图状态。
    - 由于多表关联生成物化视图会产生数据膨胀等原因，从物化视图读取的数据量不一定绝对小于源表，MaxCompute不能保证读取物化视图一定比读取源表节省费用。

3.5. DML操作

3.5.1. 插入或覆写数据（INSERT INTO | INSERT OVERWRITE）

MaxCompute支持通过 `insert into` 或 `insert overwrite` 操作向目标表或静态分区中插入、更新数据。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

前提条件

执行 `insert into` 和 `insert overwrite` 操作前需要具备目标表的修改权限（Alter）及源表的元信息读取权限（Describe）。授权操作请参见权限列表。

功能介绍

在使用MaxCompute SQL处理数据时， `insert into` 或 `insert overwrite` 操作可以将 `select` 查询的结果保存至目标表中。二者的区别是：

- `insert into` ：直接向表或静态分区中插入数据。您可以在 `insert` 语句中直接指定分区值，将数据插入指定的分区。如果您需要插入少量测试数据，可以配合VALUES使用。
- `insert overwrite` ：先清空表中的原有数据，再向表或静态分区中插入数据。

> 说明
> - MaxCompute的 insert 语法与通常使用的MySQL或Oracle的 insert 语法有差别。在 insert overwrite 后需要加 table 关键字，非直接使用 table_name 。 insert into 可以省略 table 关键字。
> - 在反复对同一个分区执行 insert overwrite 操作时，您通过 desc 命令查看到的数据分区Size会不同。这是因为从同一个表的同一个分区 select 出来再 insert overwrite 回相同分区时，文件切分逻辑发生变化，从而导致数据的Size发生变化。数据的总长度在 insert overwrite 前后是不变的，您不必担心存储计费会产生问题。
> - 并发写入场景，MaxCompute会根据ACID保障并发写入操作。关于ACID的具体语义，请参见ACID语义。

向动态分区插入数据的操作请参见插入或覆写动态分区数据（DYNAMIC PARTITION）。

使用限制

执行 insert into 和 insert overwrite 操作更新表或静态分区数据的使用限制如下：

- insert into ：不支持向聚簇表中追加数据。
- insert overwrite ：不支持指定插入列，只能使用 insert into 。例如 create table t(a string, b string); insert into t(a) values ('1'); ，a列插入1，b列为NULL或默认值。
- MaxCompute对正在操作的表没有锁机制，不要同时对一个表执行 insert into 或 insert overwrite 操作。

命令格式

```
insert {into|overwrite} table <table_name> [partition (<pt_spec>)] [(<col_name> [,<col_name> ...)]
<select_statement>
from <from_statement>
[zorder by <zcol_name> [, <zcol_name> ...]];
```

- table_name：必填。需要插入数据的目标表名称。
- pt_spec：可选。需要插入数据的分区信息，不允许使用函数等表达式，只能是常量。格式为 (partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...) 。
- col_name：可选。需要插入数据的目标表的列名称。 insert overwrite 不支持指定 [(<col_name> [,<col_name> ...)] 。
- select_statement：必填。 select 子句，从源表中查询需要插入目标表的数据。更多 select 信息，请参见SELECT语法。

> 说明
> - 源表与目标表的对应关系依赖于 select 子句中列的顺序，而不是表与表之间列名的对应关系。
> - 如果目标表是静态分区，向某个分区插入数据时，分区列不允许出现在 select 子句中。

- from_statement：必填。 from 子句，表示数据来源。例如，源表名称。
- zorder by <zcol_name> [, <zcol_name> ...]：可选。向表或分区写入数据时，支持根据指定的一列或多列（select_statement对应表中的列），把排序列数据相近的行排列在一起，提升查询时的过滤性能，在一定程度上降低存储成本。需要注意的是， order by x, y 会严格地按照先x后y的顺序对数据进行排序， zorder by x, y 会把相近的<x, y>尽量排列在一起。当SQL查询语句的过滤条件中包含排序列时， order by 后的数据仅对包含x的表达式有较好的过滤效果， zorder by 后的数据对包含x或同时包含x、y的表达式均有较好的过滤效果，列压缩比例更高。

  zorder by 的**使用限制**如下：
  - 目标表为聚簇表时，不支持 zorder by 子句。
  - zorder by 可以与 distribute by 一起使用，不能与 order by 、 cluster by 或 sort by 一起使用。

> 说明 使用 zorder by 子句写入数据时，会占用较多资源，比不排序花费时间更多。

使用示例

- 示例1：执行 insert into 命令向分区表 sale_detail 中追加数据。命令示例如下：

```
--创建一张分区表sale_detail。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china');
--向源表追加数据。其中：insert into table table_name可以简写为insert into table_name，但insert overwrite table table_name不可以省略table关键字。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3);
--开启全表扫描，仅此Session有效。执行select语句查看表sale_detail中的数据。
set odps.sql.allow.fullscan=true;
select * from sale_detail;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- 示例2：执行 `insert overwrite` 命令更新表 `sale_detail_insert` 中的数据。命令示例如下：

```
--创建目标表sale_detail_insert，与sale_detail有相同的结构。
create table sale_detail_insert like sale_detail;
--给目标表增加分区。
alter table sale_detail_insert add partition (sale_date='2013', region='china');
--从源表sale_detail中取出数据插入目标表sale_detail_insert。注意不需要声明目标表字段，也不支持重排目标表字段顺序。
--对于静态分区目标表，分区字段赋值已经在partition()部分声明，不需要在select_statement中包含，只要按照目标表普通列顺序查出对应字段，按顺序映射到目标表即可。动态分区表则需要在select中包含分区字段，详情请参见插入或覆写动态分区数据（DYNAMIC PARTITION）。
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china')
  select
  shop_name,
  customer_id,
  total_price
  from sale_detail
  zorder by customer_id, total_price;
--开启全表扫描，仅此Session有效。执行select语句查看表sale_detail_insert中的数据。
set odps.sql.allow.fullscan=true;
select * from sale_detail_insert;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- 示例3：执行 `insert overwrite` 命令更新表 `sale_detail_insert` 中的数据，调整 `select` 子句中列的顺序。源表与目标表的对应关系依赖于 `select` 子句中列的顺序，而不是表与表之间列名的对应关系。命令示例如下：

```
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china')
    select customer_id, shop_name, total_price from sale_detail;
select * from sale_detail_insert;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| c1         | s1          | 100.1       | 2013       | china      |
| c2         | s2          | 100.2       | 2013       | china      |
| c3         | s3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

在创建 `sale_detail_insert` 表时，列的顺序为 `shop_name string、customer_id string、total_price bigint` ，而从 `sale_detail` 向 `sale_detail_insert` 插入数据的顺序为 `customer_id、shop_name、total_price` 。此时，会将 `sale_detail.customer_id` 的数据插入 `sale_detail_insert.shop_name` ，将 `sale_detail.shop_name` 的数据插入 `sale_detail_insert.customer_id` 。

- 示例4：向某个分区插入数据时，分区列不允许出现在 `select` 子句中。如下语句会返回报错， `sale_date` 和 `region` 为分区列，不允许出现在静态分区的 `select` 子句中。错误命令示例如下：

```
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china')
   select shop_name, customer_id, total_price, sale_date, region from sale_detail;
```

- 示例5： `partition` 的值只能是常量，不可以为表达式。错误命令示例如下：

```
insert overwrite table sale_detail_insert partition (sale_date=datepart('2016-09-18 01:10:00', 'yyyy') , region='china')
   select shop_name, customer_id, total_price from sale_detail;
```

3.5.2. 插入或覆写动态分区数据（DYNAMIC PARTITION）

MaxCompute支持通过 `insert into` 或 `insert overwrite` 操作向动态分区中插入数据。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

前提条件

执行 `insert into` 和 `insert overwrite` 操作前需要具备目标表的修改权限（Alter）及源表的元信息读取权限（Describe）。授权操作请参见权限列表。

功能介绍

在使用MaxCompute SQL处理数据时， `insert into` 或 `insert overwrite` 语句中不直接指定分区值，只指定分区列名（分区字段）。分区列的值在 `select` 子句中提供，系统自动根据分区列的值将数据插入到相应分区。

向静态分区插入数据的操作请参见插入或覆写数据（INSERT INTO | INSERT OVERWRITE）。

使用限制

通过 `insert into` 和 `insert overwrite` 操作向动态分区中插入数据的使用限制如下：

- `insert into` 最多可以生成10000个动态分区， `insert overwrite` 最多可以生成60000个动态分区。
- 分布式环境下，在使用动态分区功能的SQL语句中，单个进程最多只能输出512个动态分区，否则会运行异常。
- 动态生成的分区值不允许为NULL，也不支持特殊字符和中文，否则会报错 `FAILED: ODPS-0123031:Partition exception - invalid dynamic partition value: province=xxx` 。
- 聚簇表不支持动态分区。

注意事项

如果您需要更新表数据到动态分区，需要注意：

- `insert into partition` 时，如果分区不存在，会自动创建分区。
- 多个 `insert into partition` 作业并发时，如果分区不存在，优先执行成功的作业会自动创建分区，但只会成功创建一个分区。
- 如果不能控制 `insert into partition` 作业并发，建议您通过 `alter table` 命令提前创建分区，详情请参见分区和列操作。
- 如果目标表有多级分区，在执行 `insert` 操作时，允许指定部分分区为静态分区，但是静态分区必须是高级分区。
- 向动态分区插入数据时，动态分区列必须在 `select` 列表中，否则会执行失败。

命令格式

```
insert {into|overwrite} table <table_name> partition (<ptcol_name>[, <ptcol_name> ...])
<select_statement> from <from_statement>;
```

- table_name：必填。需要插入数据的目标表名。
- ptcol_name：必填。目标表分区列的名称。
- select_statement：必填。 `select` 子句，从源表中查询需要插入目标表的数据。

  如果目标表只有一级动态分区，则 `select` 子句的最后一个字段值即为目标表的动态分区值。源表 `select` 的值和输出分区的值的关系是由字段顺序决定，并不是由列名称决定的。当源表的字段与目标表字段顺序不一致时，建议您按照目标表顺序在select_statement语句中指定字段。
- from_statement：必填。 `from` 子句，表示数据来源。例如，源表名称。

使用示例

- 示例1：将源表中的数据插入到目标表中。在运行SQL语句之前，您无法得知会产生哪些分区。只有在语句运行结束后，才能通过region字段产生的值确定产生的分区。命令示例如下：

```
--创建目标表total_revenues。
create table total_revenues (revenue double) partitioned by (region string);
--将源表sale_detail中的数据插入到目标表total_revenues。源表信息请参见插入或覆写数据（INSERT INTO | INSERT OVERWRITE）。
insert overwrite table total_revenues partition(region)
select total_price as revenue, region from sale_detail;
--执行show partitions语句查看表total_revenues的分区。
show partitions total_revenues;
--返回结果。
region=china
--开启全表扫描，仅此Session有效。执行select语句查看表total_revenues中的数据。
set odps.sql.allow.fullscan=true;
select * from total_revenues;
--返回结果。
+------------+------------+
| revenue    | region     |
+------------+------------+
| 100.1      | china      |
| 100.2      | china      |
| 100.3      | china      |
+------------+------------+
```

- 示例2：将源表中的数据插入到目标表中。多级分区，指定一级分区sale_date。命令示例如下：

```
--创建目标表sale_detail_dypart。
create table sale_detail_dypart like sale_detail;
--指定一级分区，将数据插入目标表。
insert overwrite table sale_detail_dypart partition (sale_date='2013', region)
select shop_name,customer_id,total_price,region from sale_detail;
--开启全表扫描，仅此Session有效。执行select语句查看表sale_detail_dypart中的数据。
set odps.sql.allow.fullscan=true;
select * from sale_detail_dypart;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- 示例3：动态分区中，select_statement字段和目标表动态分区的对应关系是由字段顺序决定，并不是由列名称决定的。命令示例如下：

```
--将源表sale_detail中的数据插入到目标表sale_detail_dypart。
insert overwrite table sale_detail_dypart partition (sale_date, region)
select shop_name,customer_id,total_price,sale_date,region from sale_detail;
--开启全表扫描，仅此Session有效。执行select语句查看表sale_detail_dypart中的数据。
set odps.sql.allow.fullscan=true;
select * from sale_detail_dypart;
--返回结果。决定目标表sale_detail_dypart动态分区的字段sale_date为源表sale_detail的字段sale_date；决定目标表sale_detail_dypart动态分区的字段region为源表sale_detail的字段region。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
--将源表sale_detail中的数据插入到目标表sale_detail_dypart，调整select字段顺序。
insert overwrite table sale_detail_dypart partition (sale_date, region)
select shop_name,customer_id,total_price,region,sale_date from sale_detail;
--开启全表扫描，仅此Session有效。执行select语句查看表sale_detail_dypart中的数据。
set odps.sql.allow.fullscan=true;
select * from sale_detail_dypart;
--返回结果。决定目标表sale_detail_dypart动态分区的字段sale_date为源表sale_detail的字段region；决定目标表sale_detail_dypart动态分区的字段region为源表sale_detail的字段sale_date。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | china      | 2013       |
| s2         | c2          | 100.2       | china      | 2013       |
| s3         | c3          | 100.3       | china      | 2013       |
+------------+-------------+-------------+------------+------------+
```

- 示例4：向动态分区插入数据时，动态分区列必须在 `select` 列表中，否则会执行失败。错误命令示例如下：

```
insert overwrite table sale_detail_dypart partition (sale_date='2013', region)
select shop_name,customer_id,total_price from sale_detail;
```

- 示例5：向动态分区插入数据时，不能仅指定低级子分区，而动态插入高级分区，否则会执行失败。错误命令示例如下：

```
insert overwrite table sale_detail_dypart partition (region='china', sale_date)
select shop_name,customer_id,total_price,sale_date from sale_detail_dypart;
```

- 示例6：MaxCompute在向动态分区插入数据时，如果分区列的类型与对应 `select` 中列的类型不严格一致，会隐式转换，命令示例如下：

```
--创建源表src。
create table src (c int, d string) partitioned by (e int);
--向源表src增加分区。
alter table src add if not exists partition (e=201312);
--向源表src追加数据。
insert into src partition (e=201312) values (1,100.1),(2,100.2),(3,100.3);
--创建目标表parttable。
create table parttable(a int, b double) partitioned by (p string);
--将源表src数据插入目标表parttable。
insert into parttable partition (p) select c, d, current_timestamp() from src;
--查询目标表parttable。
select * from parttable;
--返回结果如下。
+------------+------------+------------+
| a          | b          | p          |
+------------+------------+------------+
| 1          | 100.1      | 2020-11-25 15:13:28.686 |
| 2          | 100.2      | 2020-11-25 15:13:28.686 |
| 3          | 100.3      | 2020-11-25 15:13:28.686 |
+------------+------------+------------+
```

> **说明** 如果您的数据是有序的，动态分区插入会把数据随机打散，导致压缩率较低。推荐您使用Tunnel命令上传数据到动态分区，以获取较好的压缩率。使用该命令的详细示例请参见RDS迁移至MaxCompute实现动态分区。

3.5.3. 更新或删除数据（UPDATE | DELETE）

MaxCompute支持通过 `delete` 、 `update` 操作，在行级别删除或更新Transactional表中的数据。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

前提条件

执行 `delete` 、 `update` 操作前需要具备目标Transactional表的读取表数据权限（Select）及更新表数据权限（Update）。授权操作请参见权限列表。

功能介绍

MaxCompute的 `delete` 、 `update` 功能具备与传统数据库用法类似的删除或更新表中指定行的能力。

实际使用 `delete` 、 `update` 功能时，系统会针对每一次删除或更新操作自动生成用户不可见的Delta文件，用于记录删除或更新的数据信息。具体实现原理如下：

- `delete` ：Delta文件中使用 `txnid(bigint)` 和 `rowid(bigint)` 字段标识Transactional表的Base文件（表在系统中的存储形式）中的记录在哪次删除操作中被删除。

  例如，表t1的Base文件为f1，且内容为 `a, b, c, a, b` ，当执行 `delete from t1 where c1='a';` 命令后，系统会生成一个单独的 `f1.delta` 文件。假设 `txnid` 是 `t0` ，则 `f1.delta` 的内容是 `((0, t0), (3, t0))` ，标识行0和行3，在txnt0中被删除了。如果再执行一次 `delete` 操作，系统会又生成一个 `f2.delta` 文件，该文件仍然是根据Base文件f1编号，读取文件时，基于Base文件f1和当前所有Delta文件的共同表示结果，读取没有被删除的数据。

- `update` ： `update` 操作会转换为 `delete` + `insert into` 的实现逻辑。

`delete` 、 `update` 功能具备的优势如下：

- 写数据量下降

  此前，MaxCompute通过 `insert into` 或 `insert overwrite` 操作方式删除或更新表数据，更多信息，请参见插入或覆写数据（INSERT INTO | INSERT OVERWRITE）。当用户需要更新表或分区中的少量数据时，如果通过 `insert` 操作实现，需要先读取表的全量数据，然后通过 `select` 操作更新数据，最后通过 `insert` 操作将全量数据写回表中，效率较低。使用 `delete` 、 `update` 功能后，系统无需写回全部数据，写数据量会显著下降。

> 说明
> - 对于按量计费场景，delete、update和insert overwrite任务的写数据部分不收费，但是delete、update任务需要按分区过滤读取需要变更的数据，用于标注删除的记录或写回更新的记录，而读取数据部分依然遵照SQL作业按量计费模型收费，所以delete、update任务相比insert overwrite任务，费用并不能因为写数据量减少而减少。
> - 对于包年包月场景，delete、update减少了写数据资源消耗，与insert overwrite相比，相同资源可以运行更多的任务。

- 可直接读取最新状态的表

  此前，MaxCompute在批量更新表数据场景使用的是拉链表，该方式需要在表中增加start_date和end_date辅助列，标识某一行记录的生命周期。当查询表的最新状态时，系统需要从大量数据中根据时间戳获取表的最新状态，使用不够直观。使用delete、update功能后，系统可以基于表的Base文件和Delta文件的共同表示结果，直接读取最新状态的表。

> 注意 多次delete、update操作会使Transactional表的底层存储增大，会提高存储和后续查询费用，且影响后续查询效率，建议定期合并（Compact）后台数据。更多合并操作信息，请参见合并Transactional表文件。

当作业并发运行且操作的目标表相同时，可能会出现作业冲突问题，更多信息，请参见ACID语义。

应用场景

delete、update功能适用于随机、低频删除或更新表或分区中的少量数据。例如，按照T+1周期性地批量对表或分区中5%以下的行删除或更新数据。

delete、update功能不适用于高频更新、删除数据或实时写入目标表场景。

使用限制

delete、update功能及对应Transactional表的使用限制如下：

- 仅支持Transactional表。更多创建Transactional表信息，请参见表操作。
- 在创建表时，不支持将聚簇表、外部表设置为Transactional表。
- 不支持MaxCompute内部表、外部表、聚簇表与Transactional表互转。
- 不支持其他系统的作业（例如MaxCompute Spark、PAI、Graph）访问Transactional表。
- 不支持clone table、merge partition操作。
- 不支持通过备份与恢复功能备份数据，因此在对Transactional表的重要数据执行update、delete或insert overwrite操作前需要手动通过select + insert操作将数据备份至其他表中。

注意事项

通过delete、update操作删除或更新表或分区内的数据时，注意事项如下：

- 如果需要对表中较少数据进行删除或更新操作，且操作和后续读数据的频率也不频繁，建议使用delete、update操作，并且在多次执行删除或更新操作之后，请合并表的Base文件和Delta文件，降低表的实际存储。更多信息，请参见合并Transactional表文件。
- 如果删除或更新行数较多（超过5%）并且操作不频繁，但后续对该表的读操作比较频繁，建议使用insert overwrite或insert into操作。更多信息，请参见插入或覆写数据（INSERT INTO | INSERT OVERWRITE）。

  例如，某业务场景为每次删除或更新10%的数据，一天更新10次。建议根据实际情况评估delete、update操作产生的费用及后续对读性能的消耗是否小于每次使用insert overwrite或insert into操作产生的费用及后续对读性能的消耗，比较两种方式在具体场景中的效率，选择更优方案。
- 删除数据会生成Delta文件，所以删除数据不一定能降低存储，如果您希望通过delete操作删除数据来降低存储，请合并表的Base文件和Delta文件，降低表的实际存储。更多信息，请参见合并Transactional表文件。
- MaxCompute会按照批处理方式执行delete、update作业，每一条语句都会使用资源并产生费用，建议您使用批量方式删除或更新数据。例如您通过Python脚本生成并提交了大量行级别更新作业，且每条语句只操作一行或者少量行数据，则每条语句都会产生与SQL扫描输入数据量对应的费用，并使用相应的计算资源，多条语句累加时将明显增加费用成本，降低系统效率。命令示例如下。

```
--推荐方案。
update table1 set col1= (select value1 from table2 where table1.id = table2.id and table1.region = table2.region);
--不推荐方案。
update table1 set col1=1 where id='2021063001'and region='beijing';
update table1 set col1=2 where id='2021063002'and region='beijing';
```

删除数据（DELETE）

delete操作用于删除Transactional分区表或非分区表中满足指定条件的单行或多行数据。

- 命令格式

```
delete from <table_name> [where <where_condition>];
```

- 参数说明
  - table_name：必填。待执行delete操作的Transactional表名称。

- where_condition：可选。WHERE子句，用于筛选满足条件的数据。更多WHERE子句信息，请参见WHERE子句（where_condition）。如果不带WHERE子句，会删除表中的所有数据。

- 使用示例
  - 示例1：创建非分区表acid_delete，并导入数据，执行 `delete` 操作删除满足指定条件的行数据。命令示例如下：

```
--创建Transactional表acid_delete。
create table if not exists acid_delete(id bigint) tblproperties ("transactional"="true");
--插入数据。
insert overwrite table acid_delete values(1),(2),(3),(2);
--查看插入结果。
select * from acid_delete;
+------------+
| id         |
+------------+
| 1          |
| 2          |
| 3          |
| 2          |
+------------+
--删除id为2的行，如果在MaxCompute客户端（odpscmd）执行，需要输入yes|no确认。
delete from acid_delete where id = 2;
--查看结果表中数据只有1、3。
select * from acid_delete;
+------------+
| id         |
+------------+
| 1          |
| 3          |
+------------+
```

  - 示例2：创建分区表acid_delete_pt，并导入数据，执行 `delete` 操作删除满足指定条件的行。命令示例如下：

```
--创建Transactional表acid_delete_pt。
create table if not exists acid_delete_pt(id bigint) partitioned by(ds string) tblproperties ("transactional"="true");
--添加分区。
alter table acid_delete_pt add if not exists partition (ds= '2019');
alter table acid_delete_pt add if not exists partition (ds= '2018');
--插入数据。
insert overwrite table acid_delete_pt partition (ds='2019') values(1),(2),(3);
insert overwrite table acid_delete_pt partition (ds='2018') values(1),(2),(3);
--查看插入结果。
select * from acid_delete_pt;
+------------+------------+
| id         | ds         |
+------------+------------+
| 1          | 2018       |
| 2          | 2018       |
| 3          | 2018       |
| 1          | 2019       |
| 2          | 2019       |
| 3          | 2019       |
+------------+------------+
--删除分区为2019且id为2的数据，如果在MaxCompute客户端（odpscmd）执行，需要输入yes|no确认。
delete from acid_delete_pt where ds='2019' and id = 2;
--查看结果表中已删除分区为2019且id为2的数据。
select * from acid_delete_pt;
+------------+------------+
| id         | ds         |
+------------+------------+
| 1          | 2018       |
| 2          | 2018       |
| 3          | 2018       |
| 1          | 2019       |
| 3          | 2019       |
+------------+------------+
```

- 示例3：创建目标表acid_delete_t和关联表acid_delete_s，通过关联操作删除满足指定条件的行。命令示例如下：

```
--创建目标Transactional表acid_delete_t和关联表acid_delete_s。
create table if not exists acid_delete_t(id int,value1 int,value2 int) tblproperties ("transactional"="true");
create table if not exists acid_delete_s(id int,value1 int,value2 int);
--插入数据。
insert overwrite table acid_delete_t values(2,20,21),(3,30,31),(4,40,41);
insert overwrite table acid_delete_s values(1,100,101),(2,200,201),(3,300,301);
--删除acid_delete_t表中id与acid_delete_s表中id不匹配的行。如果在MaxCompute客户端（odpscmd）执行，需要输入yes|no确认。
delete from acid_delete_t where not exists (select * from acid_delete_s where acid_delete_t.id=acid_delete_s.id);
--查看结果表中只有id为2、3的数据。
select * from acid_delete_t;
+------------+------------+------------+
| id         | value1     | value2     |
+------------+------------+------------+
| 2          | 20         | 21         |
| 3          | 30         | 31         |
+------------+------------+------------+
```

更新数据（UPDATE）

`update` 操作用于将Transactional分区表或非分区表中行对应的单列或多列数据更新为新值。

- 命令格式

```
update <table_name> set <col1_name> = <value1> [, <col2_name> = <value2> ...] [where <where_condition>];
update <table_name> set (<col1_name> [, <col2_name> ...]) = (<value1> [, <value2> ...])[where <where_condition>];
```

- 参数说明
  - table_name：必填。待执行 `update` 操作的Transactional表名称。
  - col1_name、col2_name：至少更新一个。待修改行对应的列名称。
  - value1、value2：至少更新一个列值。修改后的新值。
  - where_condition：可选。WHERE子句，用于筛选满足条件的数据。更多WHERE子句信息，请参见WHERE子句（where_condition）。如果不带WHERE子句，会更新表中的所有数据。
- 使用示例
  - 示例1：创建非分区表acid_update，并导入数据，执行 `update` 操作更新满足指定条件的行对应的列数据。命令示例如下：

```
--创建Transactional表acid_update。
create table if not exists acid_update(id bigint) tblproperties ("transactional"="true");
--插入数据。
insert overwrite table acid_update values(1),(2),(3),(2);
--查看插入结果。
select * from acid_update;
+------------+
| id         |
+------------+
| 1          |
| 2          |
| 3          |
| 2          |
+------------+
--将所有id为2的行，id值更新为4。
update acid_update set id = 4 where id = 2;
--查看更新结果，2被更新为4。
select * from acid_update;
+------------+
| id         |
+------------+
| 1          |
| 3          |
| 4          |
| 4          |
+------------+
```

- 示例2：创建分区表acid_update，并导入数据，执行 `update` 操作更新满足指定条件的行对应的列数据。命令示例如下：

```
--创建Transactional表acid_update_pt。
create table if not exists acid_update_pt(id bigint) partitioned by(ds string) tblproperties ("transactional"="true");
--添加分区。
alter table acid_update_pt add if not exists partition (ds= '2019');
--插入数据。
insert overwrite table acid_update_pt partition (ds='2019') values(1),(2),(3);
--查看插入结果
select * from acid_update_pt where ds = '2019';
+------------+------------+
| id         | ds         |
+------------+------------+
| 1          | 2019       |
| 2          | 2019       |
| 3          | 2019       |
+------------+------------+
--更新指定行的一列数据，将分区为2019的所有id=2的行，id值更新为4。
update acid_update_pt set id = 4 where ds = '2019' and id = 2;
--查看更新结果，2被更新为4。
select * from acid_update_pt where ds = '2019';
+------------+------------+
| id         | ds         |
+------------+------------+
| 4          | 2019       |
| 1          | 2019       |
| 3          | 2019       |
+------------+------------+
```

- 示例3：创建目标表acid_update_t和关联表acid_update_s，实现同时更新多列值。命令示例如下：

```
--创建待更新目标Transactional表acid_update_t和关联表acid_update_s。
create table if not exists acid_update_t(id int,value1 int,value2 int) tblproperties ("transactional"="true");
create table if not exists acid_update_s(id int,value1 int,value2 int);
--插入数据。
insert overwrite table acid_update_t values(2,20,21),(3,30,31),(4,40,41);
insert overwrite table acid_update_s values(1,100,101),(2,200,201),(3,300,301);
--方式一：用常量更新。
update acid_update_t set (value1, value2) = (60,61);
--查询方式一目标表结果数据。
select * from acid_update_t;
+------------+------------+------------+
| id         | value1     | value2     |
+------------+------------+------------+
| 2          | 60         | 61         |
| 3          | 60         | 61         |
| 4          | 60         | 61         |
+------------+------------+------------+
--方式二：关联更新，规则为acid_update_t表左关联acid_update_s表。
update acid_update_t set (value1, value2) = (select value1, value2 from acid_update_s where acid_update_t.id = acid_update_s
.id);
--查询方式二目标表结果数据。
select * from acid_update_t;
+------------+------------+------------+
| id         | value1     | value2     |
+------------+------------+------------+
| 2          | 200        | 201        |
| 3          | 300        | 301        |
| 4          | NULL       | NULL       |
+------------+------------+------------+
--方式三：关联更新，规则为增加过滤条件，只更新交集。
update acid_update_t set (value1, value2) = (select value1, value2 from acid_update_s where acid_update_t.id = acid_update_s
.id) where acid_update_t.id in (select id from acid_update_s);
--查询方式三目标表结果数据。
select * from acid_update_t;
+------------+------------+------------+
| id         | value1     | value2     |
+------------+------------+------------+
| 4          | 40         | 41         |
| 2          | 200        | 201        |
| 3          | 300        | 301        |
+------------+------------+------------+
--方式四：用汇总结果关联更新。
update acid_update_t set (id, value1, value2) = (select id, max(value1),max(value2) from acid_update_s where acid_update_t.i
d = acid_update_s.id group by acid_update_s.id) where acid_update_t.id in (select id from acid_update_s);
--查询方式四目标表结果数据。
select * from acid_update_t;
+------------+------------+------------+
| id         | value1     | value2     |
+------------+------------+------------+
| 4          | 40         | 41         |
| 2          | 200        | 201        |
| 3          | 300        | 301        |
+------------+------------+------------+
```

合并Transactional表文件

Transactional表底层物理存储为不支持直接读取的Base文件和Delta文件。对Transactional表执行 `update` 或 `delete` 操作，不会修改Base文件，只会追加Delta文件，所以会出现更新或删除次数越多，表实际占用存储越大的情况，多次累积的Delta文件会产生较高的存储和后续查询费用。

对同一表或分区，执行多次 `update` 或 `delete` 操作，会生成较多Delta文件。系统读数据时，需要加载这些Delta文件来确定哪些行被更新或删除，较多的Delta文件会影响数据读取效率。此时您可以将Base文件和Delta合并，减少存储以便提升数据读取效率。

- 命令格式

```
alter table <table_name> [partition (<partition_key> = '<partition_value>' [, ...])] compact {minor|major};
```

- 参数说明
  - table_name：必填。待合并文件的Transactional表名称。
  - partition_key：可选。当Transactional表为分区表时，指定分区列名。
  - partition_value：可选。当Transactional表为分区表时，指定分区列名对应的列值。

- major|minor：至少选择其中一个。二者的区别是：
  - `minor`：只将Base文件及其下所有的Delta文件合并，消除Delta文件。
  - `major`：不仅将Base文件及其下所有的Delta文件合并，消除Delta文件，还会把表对应的Base文件中的小文件进行合并。当Base文件较小（小于32 MB）或有Delta文件的情况下，等价于重新对表执行`insert overwrite`操作，但当Base文件足够大（大于等于32 MB），且不存在Delta文件的情况下，不会重写。

- 使用示例
  - 示例1：基于Transactional表acid_delete，合并表文件。命令示例如下：

    ```
    alter table acid_delete compact minor;
    ```

    返回结果如下：

    ```
    Summary:
    Nothing found to merge, set odps.merge.cross.paths=true if cross path merge is permitted.
    OK
    ```

  - 示例2：基于Transactional表acid_update_pt，合并表文件。命令示例如下：

    ```
    alter table acid_update_pt partition (ds = '2019') compact major;
    ```

    返回结果如下：

    ```
    Summary:
    table name: acid_update_pt /ds=2019  instance count: 2  run time: 6
      before merge, file count:        8  file size: 2613  file physical size: 7839
       after merge, file count:        2  file size: 679  file physical size: 2037
    OK
    ```

常见问题

- 问题一：
  - 问题现象：执行`update`操作时，报错`ODPS-0010000:System internal error - fuxi job failed, caused by: Data Set should contain exactly one row`。
  - 问题原因：待更新的行数据与子查询结果中的数据无法一一对应，系统无法判断对哪一行数据进行更新。命令示例如下：

    ```
    update store set (s_county, s_manager) = (select d_country, d_manager from store_delta sd where sd.s_store_sk = store.s_store_sk) where s_store_sk in (select s_store_sk from store_delta);
    ```

    通过子查询`select d_country, d_manager from store_delta sd where sd.s_store_sk = store.s_store_sk`与store_delta关联，并用store_delta的数据更新store。假设store的s_store_sk中有`[1, 2, 3]`三行数据，如果store_delta的s_store_sk有`[1, 1]`两行数据，数据无法一一对应，执行报错。
  - 解决措施：确保待更新的行数据与子查询结果中的数据一一对应。

- 问题二：
  - 问题现象：在DataWorks DataStudio中使用`compact`命令时，报错`ODPS-0130161:[1,39] Parse exception - invalid token 'minor', expect one of 'StringLiteral','DoubleQuoteStringLiteral'`。
  - 问题原因：DataWorks独享资源组中的MaxCompute客户端版本不支持`compact`命令。
  - 解决措施：请提工单联系DataWorks技术支持团队升级独享资源组中的MaxCompute客户端版本。

3.5.4. MERGE INTO

当您需要对Transactional表执行`insert`、`update`、`delete`操作时，可以通过`merge into`功能将这些操作合并为一条SQL语句，根据与源表关联的结果，对目标Transactional表执行插入、更新或删除操作，提升执行效率。

执行`merge into`操作前需要具备目标Transactional表的读取表数据权限（Select）及更新表数据权限（Update）。授权操作请参见权限列表。

功能介绍

MaxCompute支持了`delete`、`update`功能，但当您需要使用多个`insert`、`update`、`delete`对目标表进行批量操作时，需要编写多条SQL语句，然后进行多次全表扫描才能完成操作。MaxCompute提供的`merge into`功能，只需要进行一次全表扫描操作，就可以完成全部操作，执行效率要高于`insert`+`update`+`delete`。

`merge into`操作具备原子性，作业中的`insert`、`update`、`delete`操作都执行成功时，作业才算执行成功；任一内部逻辑处理失败，则整体作业执行失败。

同时，`merge into`可以为您避免分别执行`insert`、`update`、`delete`操作时，可能导致部分操作执行成功，部分操作执行失败，其中成功部分无法回退的问题。

使用限制

`merge into`功能的使用限制如下：

- 仅支持Transactional表。更多创建Transactional表信息，请参见表操作。

- 不允许在同一条 `merge into` 语句中对相同的行执行多次 `insert` 或 `update` 操作。

命令格式

```
merge into <target_table> as <alias_name_t> using <source expression|table_name> as <alias_name_s>
--从on开始对源表和目标表的数据进行关联判断。
on <boolean expression1>
--when matched…then指定on的结果为True的行为。多个when matched…then之间的数据无交集。
when matched [and <boolean expression2>] then update set <set_clause_list>
when matched [and <boolean expression3>] then delete
--when not matched…then指定on的结果为False的行为。
when not matched [and <boolean expression4>] then insert values <value_list>
```

- target_table：必填。目标表名称，必须是实际存在的表。
- alias_name_t：必填。目标表的别名。
- source expression|table_name：必填。关联的源表名称、视图或子查询。
- alias_name_s：必填。关联的源表、视图或子查询的别名。
- boolean expression1：必填。BOOLEAN类型判断条件，判断结果必须为True或False。
- boolean expression2、boolean expression3、boolean expression4：可选。`update`、`delete`、`insert`操作相应的BOOLEAN类型判断条件。需要注意的是：
  - 当出现三个WHEN子句时，`update`、`delete`、`insert`都只能出现一次。
  - 如果`update`和`delete`同时出现，出现在前的操作必须包括`[and <boolean expression>]`。
  - `when not matched`只能出现在最后一个WHEN子句中，并且只支持`insert`操作。
- set_clause_list：当出现`update`操作时必填。待更新数据信息。更多`update`信息，请参见更新数据（UPDATE）。
- value_list：当出现`insert`操作时必填。待插入数据信息。更多`values`信息，请参见VALUES。

使用示例

创建目标表acid_address_book_base1及源表exists tmp_table1，并插入数据。执行`merge into`操作，对符合`on`条件的数据用源表的数据对目标表进行更新操作，对不符合`on`条件并且源表中满足event_type为I的数据插入目标表。命令示例如下：

```
--创建目标表acid_address_book_base1。
create table if not exists acid_address_book_base1
(id bigint,first_name string,last_name string,phone string)
partitioned by(year string, month string, day string, hour string)
tblproperties ("transactional"="true");
--创建源表exists tmp_table1。
create table if not exists tmp_table1
(id bigint, first_name string, last_name string, phone string, _event_type_ string);
--向目标表acid_address_book_base1插入测试数据。
insert overwrite table acid_address_book_base1
partition(year='2020', month='08', day='20', hour='16')
values (4, 'nihaho', 'li', '222'), (5, 'tahao', 'ha', '333'),
(7, 'djh', 'hahh', '555');
--向源表exists tmp_table1插入测试数据。
insert overwrite table tmp_table1 values
(1, 'hh', 'liu', '999', 'I'), (2, 'cc', 'zhang', '888', 'I'),
(3, 'cy', 'zhang', '666', 'I'),(4, 'hh', 'liu', '999', 'U'),
(5, 'cc', 'zhang', '888', 'U'),(6, 'cy', 'zhang', '666', 'U');
--执行merge into操作。
merge into acid_address_book_base1 as t using tmp_table1 as s
on s.id = t.id and t.year='2020' and t.month='08' and t.day='20' and t.hour='16'
when matched then update set t.first_name = s.first_name, t.last_name = s.last_name, t.phone = s.phone
when not matched and (s._event_type_='I') then insert values(s.id, s.first_name, s.last_name,s.phone,'2020','08','20','16');
--查询目标表的数据确认merge into操作结果。
select * from acid_address_book_base1;
+------------+------------+------------+------------+------------+------------+------------+------------+
| id         | first_name | last_name  | phone      | year       | month      | day        | hour       |
+------------+------------+------------+------------+------------+------------+------------+------------+
| 4          | hh         | liu        | 999        | 2020       | 08         | 20         | 16         |
| 5          | cc         | zhang      | 888        | 2020       | 08         | 20         | 16         |
| 7          | djh        | hahh       | 555        | 2020       | 08         | 20         | 16         |
| 1          | hh         | liu        | 999        | 2020       | 08         | 20         | 16         |
| 2          | cc         | zhang      | 888        | 2020       | 08         | 20         | 16         |
| 3          | cy         | zhang      | 666        | 2020       | 08         | 20         | 16         |
+------------+------------+------------+------------+------------+------------+------------+------------+
```

3.5.5. 多路输出（MULTI INSERT）

MaxCompute SQL支持您在一条SQL语句中通过 `insert into` 或 `insert overwrite` 操作将数据插入不同的目标表或者分区中，实现多路输出。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

前提条件

执行操作前需要具备目标表的修改权限（Alter）及源表的元信息读取权限（Describe）。授权操作请参见权限列表。

功能介绍

在使用MaxCompute SQL处理数据时， `multi insert` 操作可以将数据插入不同的目标表或分区中，实现多路输出。

使用限制

`multi insert` 操作的使用限制如下：

- 单条 `multi insert` 语句中最多可以写255路输出。超过255路，会上报语法错误。
- 单条 `multi insert` 语句中，对于分区表，同一个目标分区不允许出现多次。
- 单条 `multi insert` 语句中，对于非分区表，该表不能出现多次。

命令格式

```
from <from_statement>
insert overwrite | into table <table_name1> [partition (<pt_spec1>)]
<select_statement1>
insert overwrite | into table <table_name2> [partition (<pt_spec2>)]
<select_statement2>
...;
```

- from_statement：必填。 `from` 子句，代表数据来源。例如，源表名称。
- table_name：必填。需要插入数据的目标表名称。
- pt_spec：可选。需要插入数据的目标分区信息，此参数不允许使用函数等表达式，只能是常量。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partiton_col_value2, ...)` 。插入多个分区时，例如pt_spec1和pt_spec2，目标分区不允许出现多次，即pt_spec1和pt_spec2的分区信息不相同。
- select_statement：必填。 `select` 子句，从源表中查询需要插入的数据。

使用示例

- 示例1：将表*sale_detail*的数据插入到表*sale_detail_mult*的2010年及2011年中国的销售记录中。命令示例如下：

```
--创建表sale_detail_multi。
create table sale_detail_multi like sale_detail;
--开启全表扫描，仅此Session有效。将表sale_detail中的数据插入到表sale_detail_multi。
set odps.sql.allow.fullscan=true;
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2011', region='china' )
select shop_name, customer_id, total_price;
--开启全表扫描，仅此Session有效。执行select语句查看表sale_detail_multi中的数据。
set odps.sql.allow.fullscan=true;
select * from sale_detail_multi;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2010       | china      |
| s2         | c2          | 100.2       | 2010       | china      |
| s3         | c3          | 100.3       | 2010       | china      |
| s1         | c1          | 100.1       | 2011       | china      |
| s2         | c2          | 100.2       | 2011       | china      |
| s3         | c3          | 100.3       | 2011       | china      |
+------------+-------------+-------------+------------+------------+
```

- 示例2：同一目标分区不允许出现多次，否则会返回报错。**错误命令示例**如下：

```
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price;
```

3.5.6. VALUES

如果需要向表中插入少量数据，您可以通过 `insert … values` 或 `values table` 操作向数据量小的表中插入数据。

执行 `insert into` 操作前需要具备目标表的修改权限（Alter）及源表的元信息读取权限（Describe）。授权操作请参见权限列表。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

功能介绍

MaxCompute支持您通过 `insert … values` 或 `values table` 操作向表中插入少量数据。

功能	说明
`insert … values`	在业务测试阶段，您可以通过 `insert … values` 操作向表中插入数据执行简单测试： • 如果插入几条或十几条数据，您可以通过 `insert … values` 语句快速向测试表中写入数据。 • 如果插入几十条数据，您可以通过Tunnel上传一个*TXT*或*CSV*格式的数据文件导入数据，详情请参见导入数据。您还可以通过DataWorks的导入功能快速**导入**一个数据文件，详情请参见数据开发功能索引。
`values table`	如果您需要对插入的数据进行简单的运算，推荐使用MaxCompute的 `values table` 。 `values table` 可以在 `insert` 语句和任何DML语句中使用。功能如下： • 在没有任何物理表时，您可以模拟一个有任意数据的、多行的表，并进行任意运算。 • 取代 `select * from` 与 `union all` 组合的方式，构造常量表。 • `values table` 支持特殊形式。您可以不通过 `from` 子句，直接执行 `select` ， `select` 表达式列表中不可以出现其它表中的数据。其底层实现为从一个1行0列的匿名VALUES表中进行选取操作。在测试UDF或其它函数时，您可以通过该方式免去手工创建DUAL表的过程。

使用限制

通过 `insert … values` 或 `values table` 操作向表中插入数据时，不支持通过 `insert overwrite` 操作指定插入列，只能通过 `insert into` 操作指定插入列。

命令格式

```
--insert … values
insert into table <table_name>
[partition (<pt_spec>)][(<col1_name> ,<col2_name>,...)]
values (<col1_value>,<col2_value>,...),(<col1_value>,<col2_value>,...),...
--values table
values (<col1_value>,<col2_value>,...),(<col1_value>,<col2_value>,...),<table_name> (<col1_name> ,<col2_name>,...)...
```

- table_name：必填。待插入数据的表名称。该表为已经存在的表。
- pt_spec：可选。需要插入数据的目标分区信息，格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。如果需要更新的表为分区表，您需要指定该参数。
- col_name：可选。需要插入数据的目标列名称。
- col_value：可选。目标表中列对应的列值。多个列值之间用英文逗号（,）分隔。该列值支持常量，同时还支持非常量表达式，例如自定义函数或内建函数表达式。未指定列值时，默认值为NULL。

> 说明
> - 复杂数据类型无法构造对应的常量，例如ARRAY，您可以在 `values` 中使用ARRAY类型，请参见示例3。
> - 通过 `values` 写入DATETIME或TIMESTAMP数据类型时，需要在 `values` 中指定类型名称，请参见示例4。

使用示例

- 示例1：通过 `insert … values` 操作向特定分区内插入数据。命令示例如下：

```
--创建分区表srcp。
create table if not exists srcp (key string,value bigint) partitioned by (p string);
--向分区表srcp添加分区。
alter table srcp add if not exists partition (p='abc');
--向表srcp的指定分区abc中插入数据。
insert into table srcp partition (p='abc') values ('a',1),('b',2),('c',3);
--查询表srcp。
select * from srcp where p='abc';
--返回结果。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| a          | 1          | abc        |
| b          | 2          | abc        |
| c          | 3          | abc        |
+------------+------------+------------+
```

- 示例2：通过 `insert … values` 操作向非特定分区内插入数据。命令示例如下：

```
--创建分区表srcp。
create table if not exists srcp (key string,value bigint) partitioned by (p string);
--向表srcp中插入数据，不指定分区。
insert into table srcp partition (p)(key,p) values ('d','20170101'),('e','20170101'),('f','20170101');
--查询表srcp。
select * from srcp where p='20170101';
--返回结果。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| d          | NULL       | 20170101   |
| e          | NULL       | 20170101   |
| f          | NULL       | 20170101   |
+------------+------------+------------+
```

- 示例3：使用复杂数据类型构造常量，通过 `insert` 操作导入数据。命令示例如下：

```
--创建分区表srcp。
create table if not exists srcp (key string,value array<int>) partitioned by (p string);
--向分区表srcp添加分区。
alter table srcp add if not exists partition (p='abc');
--向表srcp的指定分区abc中插入数据。
insert into table srcp partition (p='abc') select 'a', array(1, 2, 3);
--查询表srcp。
select * from srcp where p='abc';
--返回结果。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| a          | [1,2,3]    | abc        |
+------------+------------+------------+
```

- 示例4：通过 `insert … values` 操作写入DATETIME或TIMESTAMP数据类型，需要在 `values` 中指定类型名称。命令示例如下：

```
--创建分区表srcp。
create table if not exists srcp (key string, value timestamp) partitioned by (p string);
--向分区表srcp添加分区。
alter table srcp add if not exists partition (p='abc');
--向表srcp的指定分区abc中插入数据。
insert into table srcp partition (p='abc') values (datetime'2017-11-11 00:00:00',timestamp'2017-11-11 00:00:00.123456789');
--查询表srcp。
select * from srcp where p='abc';
--返回结果。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| 2017-11-11 00:00:00 | 2017-11-11 00:00:00.123 | abc        |
+------------+------------+------------+
```

- 示例5：通过 `values table` 操作插入数据。命令示例如下：

```
--创建分区表srcp。
create table if not exists srcp (key string,value bigint) partitioned by (p string);
--向表srcp中插入数据。
insert into table srcp partition (p) select concat(a,b), length(a)+length(b),'20170102' from values ('d',4),('e',5),('f',6) t(
a,b);
--查询表srcp。
select * from srcp where p='20170102';
--返回结果。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| d4         | 2          | 20170102   |
| e5         | 2          | 20170102   |
| f6         | 2          | 20170102   |
+------------+------------+------------+
```

`values (…), (…) t(a, b)` 相当于定义了一个名为 `t`，列为 `a` 和 `b`，数据类型分别为STRING和BIGINT的表。列的类型需要从 `values` 列表中推导。

- 示例6：取代 `select * from` 与 `union all` 组合的方式，构造常量表。命令示例如下：

```
select 1 c union all select 2 c;
--等价于如下语句。
select * from values (1), (2) t(c);
--返回结果。
+------------+
| c          |
+------------+
| 1          |
| 2          |
+------------+
```

- 示例7：通过 `values table` 的特殊形式插入数据，不带 `from` 子句。命令示例如下：

```
--创建分区表srcp。
create table if not exists srcp (key string,value bigint) partitioned by (p string);
--向表srcp中插入数据。
insert into table srcp partition (p) select abs(-1), length('abc'), getdate();
--查询表srcp。
select * from srcp;
--返回结果。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| 1          | 3          | 2020-11-25 18:39:48 |
+------------+------------+------------+
```

- 示例8：使用非常量表达式，命令示例如下：

```
select * from values ('a'),(to_date('20190101', 'yyyyMMdd')),(getdate()) t(d);
```

返回结果如下：

```
+-------------+
| d           |
+-------------+
| 2021-02-01 18:01:38 |
| 2019-01-01 00:00:00 |
| a           |
+-------------+
```

3.6. DQL操作（SELECT）

3.6.1. SELECT语法

MaxCompute支持通过 `select` 语句查询数据。本文为您介绍 `select` 命令格式及如何实现嵌套查询、分组查询、排序等操作。

执行 `select` 操作前需要具备目标表的读取数据权限（Select）。授权操作请参见权限列表。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

功能介绍

`select` 语句用于从表中选取满足指定条件的数据。您可以根据实际场景结合以下功能完成多样化的查询操作。

类型	功能
子查询（SUBQUERY）	在某个查询的执行结果基础上进一步执行查询操作时，可以通过子查询操作实现。
交集、并集和补集	对查询结果数据集执行取交集、并集或补集操作。
JOIN	通过 `join` 操作连接表并返回符合连接条件和查询条件的数据信息。
SEMI JOIN（半连接）	通过右表过滤左表的数据，右表的数据不出现在结果集中。
MAPJOIN HINT	对一个大表和一个或多个小表执行 `join` 操作时，可以在 `select` 语句中显式指定 `mapjoin` Hint提示以提升查询性能。
DISTRIBUTED MAPJOIN	`distributed mapjoin` 是 `mapjoin` 的升级版，适用于小表 `join` 大表的场景。
SKEWJOIN HINT	当两张表Join存在热点，导致出现长尾问题时，您可以通过取出热点key，将数据分为热点数据和非热点数据两部分处理，最后合并的方式，提高Join效率。
Lateral View	通过Lateral View与UDTF（表生成函数）结合，将单行数据拆成多行数据。
GROUPING SETS	对数据进行多维度的聚合分析。
SELECT TRANSFORM	`select transform` 语法允许您启动一个指定的子进程，将输入数据按照一定的格式通过标准输入至子进程，并且通过解析子进程的标准输出获取输出数据。
Split Size Hint	通过修改Split Size来控制并发度数量。

使用限制

- 当使用 `select` 语句时，屏显最多只能显示10000行结果。当 `select` 语句作为子句时则无此限制， `select` 子句会将全部结果返回给上层查询。
- `select` 语句查询分区表时默认禁止全表扫描。

  自2018年1月10日20:00:00后，在新创建的项目上执行SQL语句时，默认情况下，针对该项目里的分区表不允许执行全表扫描操作。在查询分区表数据时必须指定分区，由此减少SQL的不必要I/O，从而减少计算资源的浪费以及按量计费模式下不必要的计算费用。

  如果您需要对分区表进行全表扫描，可以在全表扫描的SQL语句前加上命令 `set odps.sql.allow.fullscan=true;` ，并和SQL语句一起提交执行。假设 `sale_detail` 表为分区表，需要同时执行如下语句进行全表查询：

  ```
  set odps.sql.allow.fullscan=true;
  select * from sale_detail;
  ```

  如果整个项目都需要开启全表扫描，项目空间Owner执行如下命令打开开关：

  ```
  setproject odps.sql.allow.fullscan=true;
  ```

命令格式

```
[with <cte>[, ...] ]
select [all | distinct] <select_expr>[, <except_expr>)][, <replace_expr>] ...
       from <table_reference>
       [where <where_condition>]
       [group by {<col_list>|rollup(<col_list>)}]
           [having <having_condition>]
       [order by <order_condition>]
       [distribute by <distribute_condition> [sort by <sort_condition>]|[ cluster by <cluster_condition>] ]
       [limit <number>]
       [window <window_clause>]
```

命令中各字段的执行语序请参见SELECT语序。

示例数据

为便于理解使用方法，本文为您提供源数据，基于源数据提供相关示例。创建表sale_detail，并添加数据，命令示例如下。

```
--创建一张分区表sale_detail。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3)
;
```

WITH子句（cte）

可选。WITH子句包含一个或多个常用的表达式CTE。CTE充当当前运行环境中的临时表，您可以在之后的查询中引用该表。CTE使用规则如下：

- 在同一WITH子句中的CTE必须具有唯一的名字。
- 在WITH子句中定义的CTE仅对在其后定义的同一WITH子句中的其他CTE可以使用。

  假设A是子句中的第一个CTE，B是子句中的第二个CTE：

  - A引用A：无效。**错误**命令示例如下。

    ```
    with
    A as (select 1 from A)
    select * from A;
    ```

  - A引用B：无效。**错误**命令示例如下。

    ```
    with
    A as (select * from B )
    B as (select 1 as C)
    select * from B;
    ```

  - A引用B，B引用A：无效，不允许循环引用。**错误**命令示例如下

    ```
    with
    A as (select * from B ),
    B as (select * from A )
    select * from B;
    ```

正确命令示例如下。

```
with
A as (select 1 as C),
B as (select * from A)
select * from B;
```

返回结果如下。

```
+---+
| c |
+---+
| 1 |
+---+
```

列表达式（select_expr）

必填。`select_expr`格式为`col1_name, col2_name, 列表达式, ...`，表示待查询的普通列、分区列或正则表达式。列表达式使用规则如下：

- 用列名指定要读取的列。

  读取表`sale_detail`的列`shop_name`。命令示例如下。

  ```
  select shop_name from sale_detail;
  ```

  返回结果如下。

```
+------------+
| shop_name  |
+------------+
| s1         |
| s2         |
| s3         |
+------------+
```

- 用星号（ * ）代表查询所有的列。可配合 where 子句指定过滤条件。
  - 读取表 sale_detail 中所有的列。命令示例如下。

    ```
    --开启全表扫描，仅此Session有效。
    set odps.sql.allow.fullscan=true;
    select * from sale_detail;
    ```

    返回结果如下。

    ```
    +------------+-------------+-------------+------------+------------+
    | shop_name  | customer_id | total_price | sale_date  | region     |
    +------------+-------------+-------------+------------+------------+
    | s1         | c1          | 100.1       | 2013       | china      |
    | s2         | c2          | 100.2       | 2013       | china      |
    | s3         | c3          | 100.3       | 2013       | china      |
    +------------+-------------+-------------+------------+------------+
    ```

  - 在 where 子句中指定过滤条件。命令示例如下。

    ```
    select * from sale_detail where shop_name='s1';
    ```

    返回结果如下。

    ```
    +------------+-------------+-------------+------------+------------+
    | shop_name  | customer_id | total_price | sale_date  | region     |
    +------------+-------------+-------------+------------+------------+
    | s1         | c1          | 100.1       | 2013       | china      |
    +------------+-------------+-------------+------------+------------+
    ```

- 可以使用正则表达式。
  - 选出 sale_detail 表中所有列名以 sh 开头的列。命令示例如下。

    ```
    select `sh.*` from sale_detail;
    ```

    返回结果如下。

    ```
    +------------+
    | shop_name  |
    +------------+
    | s1         |
    | s2         |
    | s3         |
    +------------+
    ```

  - 选出 sale_detail 表中列名不为 shop_name 的所有列。命令示例如下。

    ```
    select `(shop_name)?+.+` from sale_detail;
    ```

    返回结果如下。

    ```
    +-------------+-------------+------------+------------+
    | customer_id | total_price | sale_date  | region     |
    +-------------+-------------+------------+------------+
    | c1          | 100.1       | 2013       | china      |
    | c2          | 100.2       | 2013       | china      |
    | c3          | 100.3       | 2013       | china      |
    +-------------+-------------+------------+------------+
    ```

- 选出 `sale_detail` 表中排除 `shop_name` 和 `customer_id` 两列的其它列。命令示例如下。

```
select `(shop_name|customer_id)?+.+` from sale_detail;
```

返回结果如下。

```
+-------------+------------+------------+
| total_price | sale_date  | region     |
+-------------+------------+------------+
| 100.1       | 2013       | china      |
| 100.2       | 2013       | china      |
| 100.3       | 2013       | china      |
+-------------+------------+------------+
```

- 选出 `sale_detail` 表中排除列名以 `t` 开头的其它列。命令示例如下。

```
select `(t.*)?+.+` from sale_detail;
```

返回结果如下。

```
+------------+-------------+------------+------------+
| shop_name  | customer_id | sale_date  | region     |
+------------+-------------+------------+------------+
| s1         | c1          | 2013       | china      |
| s2         | c2          | 2013       | china      |
| s3         | c3          | 2013       | china      |
+------------+-------------+------------+------------+
```

> **说明**
>
> 在排除多个列时，如果col2是col1的前缀，则需保证col1写在col2的前面（较长的col写在前面）。例如，一个表有2个分区无需被查询，一个分区名为 `ds` ，另一个分区名为 `dshh` ，由于前者是后者的前缀，正确表达式为 `select `(dshh|ds)?+.+` from t;` ；错误表达式为 `select `(ds|dshh)?+.+` from t;` 。

- 在选取的列名前可以使用 `distinct` 去掉重复字段，只返回去重后的值。使用 `all` 会返回字段中所有重复的值。不指定此选项时，默认值为 `all` 。
  - 查询表sale_detail中region列数据，如果有重复值时仅显示一条。命令示例如下。

```
select distinct region from sale_detail;
```

返回结果如下。

```
+------------+
| region     |
+------------+
| china      |
+------------+
```

  - 去重多列时， `distinct` 的作用域是 `select` 的列集合，不是单个列。命令示例如下。

```
select distinct region, sale_date from sale_detail;
```

返回结果如下。

```
+------------+------------+
| region     | sale_date  |
+------------+------------+
| china      | 2013       |
+------------+------------+
```

排除列（except_expr）

可选。 `except_expr` 格式为 `except(col1_name, col2_name, ...)` 。当您希望读取表内大多数列的数据，同时要排除表中少数列的数据时，可以通过 `select * except(col1_name, col2_name, ...) from ...;` 语句实现，表示读取表数据时会排除指定列（col1、col2）的数据。

命令示例如下。

```
--读取sale_detail表的数据，并排除region列的数据。
select * except(region) from sale_detail;
```

返回结果如下。

```
+-----------+-------------+-------------+-----------+
| shop_name | customer_id | total_price | sale_date |
+-----------+-------------+-------------+-----------+
| s1        | c1          | 100.1       | 2013      |
| s2        | c2          | 100.2       | 2013      |
| s3        | c3          | 100.3       | 2013      |
+-----------+-------------+-------------+-----------+
```

修改列（replace_expr）

可选。`replace_expr` 格式为 `replace(exp1 [as] col1_name, exp2 [as] col2_name, ...)` 。当您希望读取表内大多数列的数据，同时要对表中少数列的数据进行修改时，可以通过 `select * replace(exp1 as col1_name, exp2 as col2_name, ...) from ...;` 实现，表示读取表数据时会将col1的数据修改为exp1，将col2的数据修改为exp2。

命令示例如下。

```
--读取sale_detail表的数据，并修改total_price、region两列的数据。
select * replace(total_price+100 as total_price, 'shanghai' as region) from sale_detail;
```

返回结果如下。

```
+-----------+-------------+-------------+-----------+--------+
| shop_name | customer_id | total_price | sale_date | region |
+-----------+-------------+-------------+-----------+--------+
| s1        | c1          | 200.1       | 2013      | shanghai |
| s2        | c2          | 200.2       | 2013      | shanghai |
| s3        | c3          | 200.3       | 2013      | shanghai |
+-----------+-------------+-------------+-----------+--------+
```

目标表信息（table_reference）

必填。`table_reference` 表示查询的目标表信息。目标表使用规则如下：

- 直接指定目标表名。命令示例如下。

  ```
  select customer_id from sale_detail;
  ```

  返回结果如下。

  ```
  +-------------+
  | customer_id |
  +-------------+
  | c1          |
  | c2          |
  | c3          |
  +-------------+
  ```

- 嵌套子查询。命令示例如下。

  ```
  select * from (select region,sale_date from sale_detail) t where region = 'china';
  ```

  返回结果如下。

  ```
  +------------+------------+
  | region     | sale_date  |
  +------------+------------+
  | china      | 2013       |
  | china      | 2013       |
  | china      | 2013       |
  +------------+------------+
  ```

WHERE子句（where_condition）

可选。`where` 子句为过滤条件。如果表是分区表，可以实现列裁剪。使用规则如下：

- 配合关系运算符，筛选满足指定条件的数据。关系运算符包含：
  - `>` 、`<` 、`=` 、`>=` 、`<=` 、`<>`
  - `like` 、`rlike`
  - `in` 、`not in`
  - `between…and`

  详情请参见关系运算符。

  在 `where` 子句中，您可以指定分区范围，只扫描表的指定部分，避免全表扫描。命令示例如下。

```
select *
from sale_detail
where sale_date >= '2008' and sale_date <= '2014';
--等价于如下语句。
select *
from sale_detail
where sale_date between '2008' and '2014';
```

返回结果如下。

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

> **说明** 您可以通过EXPLAIN语句查看分区裁剪是否生效。普通的UDF或JOIN的分区条件写法都有可能导致分区裁剪不生效，详情请参见分区剪裁合理性评估。

- 通过UDF实现分区裁剪，将UDF语句先当作一个小作业执行，再将执行的结果替换到原来UDF出现的位置。
  - 实现方式
    - 在编写UDF的时候时，UDF类上加入Annotation。

      ```
      @com.aliyun.odps.udf.annotation.UdfProperty(isDeterministic=true)
      ```

      > **说明** `com.aliyun.odps.udf.annotation.UdfProperty` 定义在*odps-sdk-udf.jar*文件中。您需要把引用的*odps-sdk-udf*版本提高到0.30.x或以上。

    - 在SQL语句前增加 `set odps.sql.udf.ppr.deterministic = true;` 语句，此时SQL中所有的UDF均被视为 `deterministic` 。该操作执行的原理是进行执行结果回填，但是结果回填最多回填1000个分区。因此，如果UDF类加入Annotation，则可能会导致出现超过1000个回填结果的报错。此时您如果需要忽视此错误，可以通过设置 `set odps.sql.udf.ppr.to.subquery = false;` 全局关闭此功能。关闭后，UDF分区裁剪也会失效。
  - 注意事项

    使用UDF实现分区裁剪时，UDF必须在查询表的 `where` 条件里才能生效。
    - 用UDF实现分区裁剪正确示例如下。

      ```
      --UDF必须放在查询的源表的where条件中：
      select key, value from srcp where udf(ds) = 'xx';
      ```

    - 用UDF实现分区裁剪错误示例如下。

      ```
      --放在join on后面分区裁剪不会生效
      select A.c1, A.c2 from srcp1 A  join srcp2  B on A.c1 = B.c1 and udf(A.ds) ='xx';
      ```

- 在列表达式（select_expr）中，如果被重命名的列字段（赋予了列别名）使用了函数，则不能在 `where` 子句中引用列别名。**错误命令示例如下。**

```
select  task_name
        ,inst_id
        ,settings
        ,GET_JSON_OBJECT(settings, '$.SKYNET_ID') as skynet_id
        ,GET_JSON_OBJECT(settings, '$.SKYNET_NODENAME') as user_agent
from    Information_Schema.TASKS_HISTORY
where   ds = '20211215' and skynet_id is not null
limit 10;
```

GROUP BY分组查询（col_list）

可选。通常， `group by` 和聚合函数配合使用，根据指定的普通列、分区列或正则表达式进行分组。 `group by` 使用规则如下：

- `group by` 操作优先级高于 `select` 操作，因此 `group by` 的取值是 `select` 输入表的列名或由输入表的列构成的表达式。需要注意的是：
  - `group by` 取值为正则表达式时，必须使用列的完整表达式。
  - `select` 语句中没有使用聚合函数的列必须出现在 `group by` 中。

使用示例：

- 直接使用输入表列名region作为 `group by` 的列，即以region值分组。命令示例如下。

  ```
  select region from sale_detail group by region;
  ```

  返回结果如下。

  ```
  +------------+
  | region     |
  +------------+
  | china      |
  +------------+
  ```

- 以region值分组，返回每一组的销售额总量。命令示例如下。

  ```
  select sum(total_price) from sale_detail group by region;
  ```

  返回结果如下。

  ```
  +------------+
  | _c0        |
  +------------+
  | 300.6      |
  +------------+
  ```

- 以region值分组，返回每一组的region值（组内唯一）及销售额总量。命令示例如下。

  ```
  select region, sum (total_price) from sale_detail group by region;
  ```

  返回结果如下。

  ```
  +------------+------------+
  | region     | _c1        |
  +------------+------------+
  | china      | 300.6      |
  +------------+------------+
  ```

- 以 `select` 列的别名分组，命令示例如下。

  ```
  select region as r from sale_detail group by r;
  --等效于如下语句。
  select region as r from sale_detail group by region;
  ```

  返回结果如下。

  ```
  +------------+
  | r          |
  +------------+
  | china      |
  +------------+
  ```

- 以列表达式分组，必须使用列的完整表达式。命令示例如下。

  ```
  select 2 + total_price as r from sale_detail group by 2 + total_price;
  ```

  返回结果如下。

  ```
  +------------+
  | r          |
  +------------+
  | 102.1      |
  | 102.2      |
  | 102.3      |
  +------------+
  ```

- `select` 的所有列中没有使用聚合函数的列，必须出现在 `group by` 中，否则返回报错。**错误命令示例**如下。

  ```
  select region, total_price from sale_detail group by region;
  ```

  正确命令示例如下。

  ```
  select region, total_price from sale_detail group by region, total_price;
  ```

  返回结果如下。

  ```
  +------------+-------------+
  | region     | total_price |
  +------------+-------------+
  | china      | 100.1       |
  | china      | 100.2       |
  | china      | 100.3       |
  +------------+-------------+
  ```

- 当SQL语句设置了属性，即 `set odps.sql.groupby.position.alias=true;` ，`group by` 中的整型常量会被当做 `select` 的列序号处理。命令示例如下。

  ```
  --与下一条SQL语句一起执行。
  set odps.sql.groupby.position.alias=true;
  --1代表select的列中第一列即region，以region值分组，返回每一组的region值（组内唯一）及销售额总量。
  select region, sum(total_price) from sale_detail group by 1;
  ```

  返回结果如下。

  ```
  +------------+------------+
  | region     | _c1        |
  +------------+------------+
  | china      | 300.6      |
  +------------+------------+
  ```

HAVING子句（having_condition）

可选。通常 `having` 子句与聚合函数一起使用，实现过滤。命令示例如下。

```
--为直观展示数据呈现效果，向sale_detail表中追加数据。
insert into sale_detail partition (sale_date='2014', region='shanghai') values ('null','c5',null),('s6','c6',100.4),('s7','c7',10
0.5);
--使用having子句配合聚合函数实现过滤。
select region,sum(total_price) from sale_detail
group by region
having sum(total_price)<305;
```

返回结果如下。

```
+------------+------------+
| region     | _c1        |
+------------+------------+
| china      | 300.6      |
| shanghai   | 200.9      |
+------------+------------+
```

ORDER BY全局排序（order_condition）

可选。 `order by` 用于对所有数据按照指定普通列、分区列或指定常量进行全局排序。 `order by` 使用规则如下：

- 默认对数据进行升序，如果降序排序，需要使用 `desc` 关键字。
- `order by` 默认要求带 `limit` 数据行数限制，没有 `limit` 会返回报错。如您需要解除 `order by` 必须带 `limit` 的限制，详情请参见LIMIT NUMBER限制输出行数>解除ORDER BY必须带LIMIT的限制。
  - 查询表sale_detail的信息，并按照total_price升序排列前2条。命令示例如下。

    ```
    select * from sale_detail order by total_price limit 2;
    ```

    返回结果如下。

    ```
    +------------+-------------+-------------+------------+------------+
    | shop_name  | customer_id | total_price | sale_date  | region     |
    +------------+-------------+-------------+------------+------------+
    | s1         | c1          | 100.1       | 2013       | china      |
    | s2         | c2          | 100.2       | 2013       | china      |
    +------------+-------------+-------------+------------+------------+
    ```

- 查询表sale_detail的信息，并按照total_price降序排列前2条。命令示例如下。

```
select * from sale_detail order by total_price desc limit 2;
```

返回结果如下。

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s3         | c3          | 100.3       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- 在使用 `order by` 排序时，NULL会被认为比任何值都小，这个行为与MySQL一致，但是与Oracle不一致。

查询表sale_detail的信息，并按照total_price升序排列前2条。命令示例如下。

```
select * from sale_detail order by total_price limit 2;
```

返回结果如下。

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- `order by` 后面需要加上 `select` 列的别名。当 `select` 某列时，如果没有指定列的别名，则列名会被作为列的别名。

`order by` 加列的别名。命令示例如下。

```
select total_price as t from sale_detail order by total_price limit 3;
--等效于如下语句。
select total_price as t from sale_detail order by t limit 3;
```

返回结果如下。

```
+------------+
| t          |
+------------+
| 100.1      |
| 100.2      |
| 100.3      |
+------------+
```

- 当SQL语句设置了属性，即 `set odps.sql.orderby.position.alias=true;` ， `order by` 中的整型常量会被当做 `select` 的列序号处理。命令示例如下。

```
--与下一条SQL语句一起执行。
set odps.sql.orderby.position.alias=true;
select * from sale_detail order by 3 limit 3;
```

返回结果如下。

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- `offset` 可以和 `order by...limit` 语句配合使用，用于指定跳过的行数，格式为 `order by...limit m offset n` ，也可以简写为 `order by...limit n, m` 。其中： `limit m` 控制输出m行数据， `offset n` 表示在开始返回数据之前跳过的行数。 `offset 0` 与省略offset子句效果相同。

将表sale_detail按照total_price升序排序后，输出从第3行开始的3行数据。命令示例如下。

```
select customer_id,total_price from sale_detail order by total_price limit 3 offset 2;
--等效于如下语句。
select customer_id,total_price from sale_detail order by total_price limit 2, 3;
```

返回结果如下。

```
+-------------+-------------+
| customer_id | total_price |
+-------------+-------------+
| c3          | 100.3       |
+-------------+-------------+
```

由于查询到的数据从第3行开始仅剩1行数据，不足3行，所以返回结果只有1行。

DISTRIBUTE BY哈希分片（distribute_condition）

可选。`distribute by`用于对数据按照某几列的值做Hash分片。

`distribute by`控制Map（读数据）的输出在Reducer中是如何划分的，如果不希望Reducer的内容存在重叠，或需要对同一分组的数据一起处理，您可以使用`distribute by`来保证同组数据分发到同一个Reducer中。

必须使用`select`的输出列别名，当`select`某列时，如果没有指定列的别名，则列名会被作为列的别名。命令示例如下：

```
--查询表sale_detail中的列region值并按照region值进行哈希分片。
select region from sale_detail distribute by region;
--等价于如下语句。
select region as r from sale_detail distribute by region;
select region as r from sale_detail distribute by r;
```

SORT BY局部排序（sort_condition）

可选。通常，配合`distribute by`使用。`sort by`使用规则如下：

- `sort by`默认对数据进行升序，如果降序排序，需要使用`desc`关键字。
- 如果`sort by`语句前有`distribute by`，`sort by`会对`distribute by`的结果按照指定的列进行排序。
  - 查询表sale_detail中的列region和total_price的值并按照region值进行哈希分片，然后按照total_price对哈希分片结果进行局部升序排序。命令示例如下。

    ```
    --为直观展示数据呈现效果，向sale_detail表中追加数据。
    insert into sale_detail partition (sale_date='2014', region='shanghai') values ('null','c5',null),('s6','c6',100.4),('s7','c7',100.5);
    select region,total_price from sale_detail distribute by region sort by total_price;
    ```

    返回结果如下。

    ```
    +------------+-------------+
    | region     | total_price |
    +------------+-------------+
    | shanghai   | NULL        |
    | china      | 100.1       |
    | china      | 100.2       |
    | china      | 100.3       |
    | shanghai   | 100.4       |
    | shanghai   | 100.5       |
    +------------+-------------+
    ```

  - 查询表sale_detail中的列region和total_price的值并按照region值进行哈希分片，然后按照total_price对哈希分片结果进行局部降序排序。命令示例如下。

    ```
    select region,total_price from sale_detail distribute by region sort by total_price desc;
    ```

    返回结果如下。

    ```
    +------------+-------------+
    | region     | total_price |
    +------------+-------------+
    | shanghai   | 100.5       |
    | shanghai   | 100.4       |
    | china      | 100.3       |
    | china      | 100.2       |
    | china      | 100.1       |
    | shanghai   | NULL        |
    +------------+-------------+
    ```

- 如果`sort by`语句前没有`distribute by`，`sort by`会对每个Reduce中的数据进行局部排序。

  保证每个Reduce的输出数据都是有序的，从而增加存储压缩率，同时读取时如果有过滤，能够减少真正从磁盘读取的数据量，提高后续全局排序的效率。命令示例如下。

  ```
  select region,total_price from sale_detail sort by total_price desc;
  ```

  返回结果如下。

```
+------------+-------------+
| region     | total_price |
+------------+-------------+
| china      | 100.3       |
| china      | 100.2       |
| china      | 100.1       |
| shanghai   | 100.5       |
| shanghai   | 100.4       |
| shanghai   | NULL        |
+------------+-------------+
```

> 说明
> - `order by|distribute by|sort by` 的取值必须是 `select` 语句的输出列，即列的别名。列的别名可以为中文。
> - 在MaxCompute SQL解析中，`order by|distribute by|sort by` 执行顺序在 `select` 操作之后，因此它们的取值只能为 `select` 语句的输出列。
> - `order by` 不和 `distribute by` 、 `sort by` 同时使用， `group by` 也不和 `distribute by` 、 `sort by` 同时使用。

LIMIT限制输出行数（number）

可选。 `limit <number>` 中的 `number` 是常数，用于限制输出行数。

> 说明 `limit` 基于分布式系统对数据进行扫描后过滤，您无法通过 `limit` 减少返回数据量进而减少计算费用。

当您涉及到如下场景时，可参考对应解决方案处理：

- 解除 `order by` 必须带 `limit` 的限制

  因为 `order by` 需要对单个执行节点做全局排序，所以默认带 `limit` 限制，避免误用导致单点处理大量数据。如果您的使用场景确实需要 `order by` 放开 `limit` 限制，可以通过如下两种方式实现：

  - Project级别：设置 `setproject odps.sql.validate.orderby.limit=false;` 关闭 `order by` 必须带 `limit` 的限制。
  - Session级别：设置 `set odps.sql.validate.orderby.limit=false;` 关闭 `order by` 必须带 `limit` 的限制，需要与SQL语句一起提交。

    > 说明 如果关闭 `order by` 必须带 `limit` 的限制，在单个执行节点有大量数据排序的情况下，资源消耗或处理时长等性能表现会受到影响。

- 解除屏显限制

  当使用无 `limit` 的 `select` 语句或 `limit` 的 `number` 数量超过设置的屏显上限时，如果您直接从屏显窗口查看结果，最多只能输出屏显上限设置的行数。

  每个项目空间的屏显上限可能不同，您可以参考如下方法控制：

  - 如果关闭了项目空间数据保护，修改odpscmd_config.ini文件。

    设置odpscmd_config.ini文件中的 `use_instance_tunnel=true` ，如果不配置 `instance_tunnel_max_record` 参数，则屏显行数不受限制；否则，屏显行数受 `instance_tunnel_max_record` 参数值限制。 `instance_tunnel_max_record` 参数值上限为10000行。Instance Tunnel详情请参见使用说明。
  - 如果开启了项目空间数据保护，屏显行数受 `READ_TABLE_MAX_ROW` 参数值限制，配置上限为10000行。

    > 说明 您可以执行 `show SecurityConfiguration;` 命令查看 `ProjectProtection` 属性配置。如果 `ProjectProtection=true` ，根据项目空间数据保护需求判断是否关闭数据保护机制。如果可以关闭，通过 `set ProjectProtection=false;` 命令关闭。 `ProjectProtection` 属性默认不开启。项目空间数据保护机制详情请参见数据保护机制。

窗口子句（window_clause）

详细窗口子句信息，请参见窗口函数语法。

Split Size Hint

可选。您可通过调整Split Size来控制并发度，调整计算性能。Split Size功能可以作用到表级别。指定的值单位为MB，默认值为256MB。

- 使用场景。
  - 当您发现作业有很多子任务都在等待资源，没有办法申请到更多资源的情况，可以通过调高Split Size值，减少并发度，可以节省起停子任务的时间。
  - 当并发度比较低，例如并发度只有几百，而且当前执行的子任务运行很久都没有出结果，但是资源池中还有很多资源时，可以调低值来提高并发度，降低整个作业的运行时间。
- 注意事项。
  - 此Hint对于Cluster表，如果优化中会使用Cluster的分桶属性，那么Split Size Hint会失效。

- Split Size的值按照256MB的倍数进行调整，例如128MB、512MB等。
- 一个SQL中有重复读同张表，那么Hint会被合并成为指定的最小值。
  - SQL中有两个读表src的地方，一个Hint为1MB一个为10MB，那么最后会按照1MB来切分。
  - SQL中有两个读表src的地方，一个Hint为1MB一个没有，那么最后会按照1MB来切分。

- 使用示例。

```
--设置split size大小为1MB，此hint会在读表src时，按照1M的大小来切分task
select a.key from src a /*+split_size(1)*/ join src2 b on a.key=b.key;
```

3.6.2. SELECT语序

对于按照 `select` 语法格式书写的 `select` 语句，它的逻辑执行顺序与标准的书写语序并不相同。本文为您介绍 `select` 语句中操作的执行语序并提供使用示例。

执行语序

在SELECT语法中，涉及的操作主要包括：

- `select`
- `from`
- `where`
- `group by`
- `having`
- `order by`
- `distribute by`
- `sort by`
- `limit`

基于 `order by` 不和 `distribute by` 、 `sort by` 同时使用， `group by` 也不和 `distribute by` 、 `sort by` 同时使用的限制，常见 `select` 语句的执行顺序如下：

- 场景1： `from` -> `where` -> `group by` -> `having` -> `select` -> `order by` -> `limit`
- 场景2： `from` -> `where` -> `select` -> `distribute by` -> `sort by`

为避免混淆，MaxCompute支持以执行顺序书写查询语句，语法结构可改为如下形式：

```
from <table_reference>
[where <where_condition>]
[group by <col_list>]
[having <having_condition>]
select [all | distinct] <select_expr>, <select_expr>, ...
[order by <order_condition>]
[distribute by <distribute_condition> [sort by <sort_condition>] ]
[limit <number>]
```

示例数据

为便于理解，本文为您提供源数据，基于源数据提供相关示例。创建表sale_detail，并添加数据，命令示例如下：

```
--创建一张分区表sale_detail。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china') partition (sale_date='2014', region='shanghai');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3)
;
insert into sale_detail partition (sale_date='2014', region='shanghai') values ('null','c5',null),('s6','c6',100.4),('s7','c7',10
0.5);
```

使用示例

- 示例1，符合场景1的命令示例如下：

```
--按照select语法书写。
select region,max(total_price)
from sale_detail
where total_price > 100
group by region
having sum(total_price)>305
order by region
limit 5;
--按照执行顺序书写。与上一写法等效。
from sale_detail
where total_price > 100
group by region
having sum(total_price)>305
select region,max(total_price)
order by region
limit 5;
```

返回结果如下：

```
+------------+------------+
| region     | _c1        |
+------------+------------+
| china      | 100.3      |
+------------+------------+
```

该命令的执行逻辑如下：

i. 从sale_detail表（`from sale_detail`）中取出满足`where total_price > 100`条件的数据。

ii. 对于i中得到的结果按照region进行分组（`group by region`）。

iii. 对于ii中得到的结果筛选分组中满足total_price之和大于305的数据（`having sum(total_price)>305`）。

iv. 对于iii中得到的结果`select region,max(total_price)`。

v. 对于iv中得到的结果按照region进行排序（`order by region`）。

vi. 对于v中得到的结果仅显示前5条数据（`limit 5`）。

- 示例2，符合场景2的命令示例如下：

```
--按照select语法书写。
select shop_name
       ,total_price
       ,region
from   sale_detail
where  total_price > 100.2
distribute by region
sort by total_price;
--按照执行顺序书写。与上一写法等效。
from   sale_detail
where  total_price > 100.2
select shop_name
       ,total_price
       ,region
distribute by region
sort by total_price;
```

返回结果如下：

```
+------------+-------------+------------+
| shop_name  | total_price | region     |
+------------+-------------+------------+
| s3         | 100.3       | china      |
| s6         | 100.4       | shanghai   |
| s7         | 100.5       | shanghai   |
+------------+-------------+------------+
```

该命令的执行逻辑如下：

i. 从sale_detail表（`from sale_detail`）中取出满足`where total_price > 100.2`条件的数据。

ii. 对于i中得到的结果`select shop_name, total_price, region`。

iii. 对于ii中得到的结果按照region进行哈希分片（`distribute by region`）。

iv. 对于iii中得到的结果按照total_price进行升序排列（`sort by total_price`）。

3.6.3. 子查询（SUBQUERY）

当您需要在某个查询的执行结果基础上进一步执行查询操作时，可以通过子查询操作实现。本文为您介绍MaxCompute支持的子查询的定义与使用方法。

功能介绍

子查询指在一个完整的查询语句之中，嵌套若干个不同功能的小查询，从而一起完成复杂查询的一种编写形式。MaxCompute支持的子查询包含如下几种：

- 基础子查询
- IN SUBQUERY
- NOT IN SUBQUERY
- EXISTS SUBQUERY
- NOT EXISTS SUBQUERY
- SCALAR SUBQUERY

示例数据

为便于理解，本文为您提供源数据，基于源数据提供相关示例。创建表sale_detail，并添加数据，命令示例如下：

```
--创建一张分区表sale_detail。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china') partition (sale_date='2014', region='shanghai');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3)
;
insert into sale_detail partition (sale_date='2014', region='shanghai') values ('null','c5',null),('s6','c6',100.4),('s7','c7',10
0.5);
```

基础子查询

普通查询操作的对象是目标表，但是查询的对象也可以是另一个 `select` 语句，这种查询为子查询。在 `from` 子句中，子查询可以被当作一张表，与其他表或子查询进行 `join` 操作。 `join` 详情请参见JOIN。

- 命令格式
  - 格式1

    ```
    select <select_expr> from (<select_statement>) [<sq_alias_name>];
    ```

  - 格式2

    ```
    select (<select_statement>) from <table_name>;
    ```

- 参数说明
  - select_expr：必填。格式为 `col1_name, col2_name, 正则表达式, ...` ，表示待查询的普通列、分区列或正则表达式。
  - select_statement：必填。子查询语句。如果子查询语句为格式2，子查询结果必须只有一行。格式请参见SELECT语法。
  - sq_alias_name：可选。子查询的别名。
  - table_name：必填。目标表名称。
- 使用示例
  - 示例1：使用格式1子查询语法。命令示例如下。

    ```
    select * from (select shop_name from sale_detail) a;
    ```

    返回结果如下：

    ```
    +------------+
    | shop_name  |
    +------------+
    | s1         |
    | s2         |
    | s3         |
    | null       |
    | s6         |
    | s7         |
    +------------+
    ```

- 示例2：使用格式2子查询语法。命令示例如下。

```
select (select * from sale_detail where shop_name='s1') from sale_detail;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s1         | c1          | 100.1       | 2013       | china      |
| s1         | c1          | 100.1       | 2013       | china      |
| s1         | c1          | 100.1       | 2013       | china      |
| s1         | c1          | 100.1       | 2013       | china      |
| s1         | c1          | 100.1       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- 示例3：使用格式1子查询语法，在 `from` 子句中，子查询可以被当做一张表，与其他的表或子查询进行 `join` 操作。命令示例如下。

```
--先新建一张表，再执行join操作。
create table shop as select shop_name,customer_id,total_price from sale_detail;
select a.shop_name, a.customer_id, a.total_price from
(select * from shop) a join sale_detail on a.shop_name = sale_detail.shop_name;
```

返回结果如下：

```
+------------+-------------+-------------+
| shop_name  | customer_id | total_price |
+------------+-------------+-------------+
| null       | c5          | NULL        |
| s6         | c6          | 100.4       |
| s7         | c7          | 100.5       |
| s1         | c1          | 100.1       |
| s2         | c2          | 100.2       |
| s3         | c3          | 100.3       |
+------------+-------------+-------------+
```

IN SUBQUERY

`in subquery` 与 `left semi join` 用法类似。

- 命令格式
  - 格式1

```
select <select_expr1> from <table_name1> where <select_expr2> in (select <select_expr2> from <table_name2>);
--等效于left semi join如下语句。
select <select_expr1> from <table_name1> <alias_name1> left semi join <table_name2> <alias_name2> on <alias_name1>.<select_e
xpr1> = <alias_name2>.<select_expr2>;
```

> **说明** 如果 `select_expr2` 为分区列，则 `select <select_expr2> from <table_name2>` 会单独启动作业执行子查询，而不会转化为 `semi join` 。执行后的结果会依次与 `select_expr2` 比较， `table_name1` 中 `select_expr2` 值不在返回结果中的分区将不会被读取，保证分区裁剪仍然有效。

  - 格式2

MaxCompute不仅支持 `in subquery` ，还支持Correlated条件。子查询中的 `where <table_name2_colname> = <table_name1>.<colname>` 即是一个Correlated条件。MaxCompute 1.0版本不支持这种既引用了子查询中源表，又引用了外层查询源表的表达式。MaxCompute 2.0已支持这种用法，这种过滤条件构成了 `semi join` 中 `on` 条件的一部分。

```
select <select_expr1> from <table_name1> where <select_expr2> in (select <select_expr2> from <table_name2> where <table_name
2_colname> = <table_name1>.<colname>);
```

> **说明** MaxCompute支持 `in subquery` 不作为 `join` 条件，例如出现在非 `where` 语句中，或虽然在 `where` 语句中，但无法转换为 `join` 条件场景。此时无法转换为 `semi join` ，必须启动一个单独的作业运行子查询，不支持Correlated条件。

  - 格式3

在上述能力及限制的基础上，兼容PostgreSQL支持多列的需求，相较于拆分为多个Subquery的实现方式，会减少一次JOIN过程并节省计算资源。支持的多列用法如下：

  - `in` 后的表达式可以为简单的SELECT多列语句。
  - `in` 后的表达式中可以使用聚合函数。更多聚合函数信息，请参见聚合函数。
  - `in` 后的表达式可以为常量。

- 参数说明
  - select_expr1：必填。格式为 `col1_name, col2_name, 正则表达式,...` ，表示待查询的普通列、分区列或正则表达式。
  - table_name1、table_name2：必填。表的名称。
  - select_expr2：必填。表示table_name1和table_name2互相映射的列名。
  - col_name：必填。表的列名。
- 使用示例
  - 示例1：使用格式1子查询语法。命令示例如下。

    ```
    select * from sale_detail where shop_name in (select shop_name from shop);
    ```

    返回结果如下：

    ```
    +------------+-------------+-------------+------------+------------+
    | shop_name  | customer_id | total_price | sale_date  | region     |
    +------------+-------------+-------------+------------+------------+
    | null       | c5          | NULL        | 2014       | shanghai   |
    | s6         | c6          | 100.4       | 2014       | shanghai   |
    | s7         | c7          | 100.5       | 2014       | shanghai   |
    | s1         | c1          | 100.1       | 2013       | china      |
    | s2         | c2          | 100.2       | 2013       | china      |
    | s3         | c3          | 100.3       | 2013       | china      |
    +------------+-------------+-------------+------------+------------+
    ```

  - 示例2：使用格式2子查询语法。命令示例如下。

    ```
    select * from sale_detail where shop_name in (select shop_name from shop where customer_id = shop.customer_id);
    ```

    返回结果如下：

    ```
    +-----------+-------------+-------------+-----------+--------+
    | shop_name | customer_id | total_price | sale_date | region |
    +-----------+-------------+-------------+-----------+--------+
    | null      | c5          | NULL        | 2014      | shanghai |
    | s6        | c6          | 100.4       | 2014      | shanghai |
    | s7        | c7          | 100.5       | 2014      | shanghai |
    | s1        | c1          | 100.1       | 2013      | china  |
    | s2        | c2          | 100.2       | 2013      | china  |
    | s3        | c3          | 100.3       | 2013      | china  |
    +-----------+-------------+-------------+-----------+--------+
    ```

  - 示例3： `in subquery` 不作为 `join` 条件。命令示例如下。

    ```
    select * from sale_detail where shop_name in (select shop_name from shop) and total_price > 100.3;
    ```

    因为 `where` 中包含了 `and` ，所以无法转换为 `semi join` ，会单独启动作业执行子查询。

    返回结果如下：

    ```
    +------------+-------------+-------------+------------+------------+
    | shop_name  | customer_id | total_price | sale_date  | region     |
    +------------+-------------+-------------+------------+------------+
    | s6         | c6          | 100.4       | 2014       | shanghai   |
    | s7         | c7          | 100.5       | 2014       | shanghai   |
    +------------+-------------+-------------+------------+------------+
    ```

- 示例4：SELECT多列场景。命令示例如下。

```
--为方便理解，此处重新构造示例数据。
create table if not exists t1(a bigint,b bigint,c bigint,d bigint,e bigint);
create table if not exists t2(a bigint,b bigint,c bigint,d bigint,e bigint);
insert into table t1 values (1,3,2,1,1),(2,2,1,3,1),(3,1,1,1,1),(2,1,1,0,1),(1,1,1,0,1);
insert into table t2 values (1,3,5,0,1),(2,2,3,1,1),(3,1,1,0,1),(2,1,1,0,1),(1,1,1,0,1);
--场景一：in后的表达式为简单的SELECT多列语句。
select a, b from t1 where (c, d) in (select a, b from t2 where e = t1.e);
--返回结果如下。
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 3          |
| 2          | 2          |
| 3          | 1          |
+------------+------------+
--场景二：in后的表达式使用聚合函数。
select a, b from t1 where (c, d) in (select max(a), b from t2 where e = t1.e group by b having max(a) > 0);
--返回结果如下。
+------------+------------+
| a          | b          |
+------------+------------+
| 2          | 2          |
+------------+------------+
--场景三：in后的表达式为常量。
select a, b from t1 where (c, d) in ((1, 3), (1, 1));
--返回结果如下。
+------------+------------+
| a          | b          |
+------------+------------+
| 2          | 2          |
| 3          | 1          |
+------------+------------+
```

NOT IN SUBQUERY

`not in subquery` 与 `left anti join` 用法类似，但并不完全相同。如果查询目标表的指定列名中有任意一行为NULL，则 `not in` 表达式值为NULL，导致 `where` 条件不成立，无数据返回，这点与 `left anti join` 不同。

- 命令格式
  - 格式1

```
select <select_expr1> from <table_name1> where <select_expr2> not in (select <select_expr2> from <table_name2>);
--等效于left anti join如下语句。
select <select_expr1> from <table_name1> <alias_name1> left anti join <table_name2> <alias_name2> on <alias_name1>.<select_e
xpr1> = <alias_name2>.<select_expr2>;
```

> **说明** 如果 `select_expr2` 为分区列，则 `select <select_expr2> from <table_name2>` 会单独启动作业执行子查询，而不会转化为 `anti join` 。执行后的结果会依次与 `select_expr2` 比较， `table_name1` 中 `select_expr2` 值不在返回结果中的分区将不会被读取，保证分区裁剪仍然有效。

  - 格式2

MaxCompute不仅支持 `not in subquery` ，还支持Correlated条件。子查询中的 `where <table_name2_colname> = <table_name1>.<colname>` 即是一个Correlated条件。MaxCompute 1.0版本不支持这种既引用了子查询中源表，又引用了外层查询源表的表达式。MaxCompute 2.0已支持这种用法，这种过滤条件构成了 `anti join` 中 `on` 条件的一部分。

```
select <select_expr1> from <table_name1> where <select_expr2> not in (select <select_expr2> from <table_name2> where <table_
name2_colname> = <table_name1>.<colname>);
```

> **说明** MaxCompute支持 `not in subquery` 不作为 `join` 条件，例如出现在非 `where` 语句中，或虽然在 `where` 语句中，但无法转换为 `join` 条件场景。此时无法转换为 `anti join` ，必须启动一个单独的作业运行子查询，不支持Correlated条件。

  - 格式3

在上述能力的基础上，兼容PostgreSQL支持多列的需求，相较于拆分为多个Subquery的实现方式，会减少一次JOIN过程并节省计算资源。支持的多列场景如下：

    - `not in` 后的表达式可以为简单的SELECT多列语句。
    - `not in` 后的表达式中可以使用聚合函数。更多聚合函数信息，请参见聚合函数。
    - `not in` 后的表达式可以为常量。

- 参数说明
  - select_expr1：必填。格式为 `col1_name, col2_name, 正则表达式,...` ，表示待查询的普通列、分区列或正则表达式。
  - table_name1、table_name2：必填。表的名称。
  - select_expr2：必填。表示table_name1和table_name2互相映射的列名。
  - col_name：必填。表的列名。
- 使用示例
  - 示例1：使用格式1子查询语法。命令示例如下。

```
--创建一张新表shop1并追加数据。
create table shop1 as select shop_name,customer_id,total_price from sale_detail;
insert into shop1 values ('s8','c1',100.1);
select * from shop1 where shop_name not in (select shop_name from sale_detail);
```

返回结果如下：

```
+------------+-------------+-------------+
| shop_name  | customer_id | total_price |
+------------+-------------+-------------+
| s8         | c1          | 100.1       |
+------------+-------------+-------------+
```

  - 示例2：使用格式2子查询语法。命令示例如下。

```
select * from shop1 where shop_name not in (select shop_name from sale_detail where customer_id = shop1.customer_id);
```

返回结果如下：

```
+------------+-------------+-------------+
| shop_name  | customer_id | total_price |
+------------+-------------+-------------+
| s8         | c1          | 100.1       |
+------------+-------------+-------------+
```

  - 示例3： `not in subquery` 不作为 `join` 条件。命令示例如下。

```
select * from shop1 where shop_name not in (select shop_name from sale_detail) and total_price < 100.3;
```

因为 `where` 中包含了 `and` ，所以无法转换为 `anti join` ，会单独启动作业执行子查询。

返回结果如下：

```
+------------+-------------+-------------+
| shop_name  | customer_id | total_price |
+------------+-------------+-------------+
| s8         | c1          | 100.1       |
+------------+-------------+-------------+
```

  - 示例4：假设查询表中有任意一行为NULL，则无数据返回。命令示例如下。

```
--创建一张新表sale并追加数据。
create table if not exists sale
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
alter table sale add partition (sale_date='2013', region='china');
insert into sale partition (sale_date='2013', region='china') values ('null','null',null),('s2','c2',100.2),('s3','c3',100.3
);
select * from sale where shop_name not in (select shop_name from sale_detail);
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
+------------+-------------+-------------+------------+------------+
```

- 示例5：SELECT多列场景。命令示例如下。

```
--为方便理解，此处重新构造示例数据。与IN SUBQUERY中的示例数据相同。
create table if not exists t1(a bigint,b bigint,c bigint,d bigint,e bigint);
create table if not exists t2(a bigint,b bigint,c bigint,d bigint,e bigint);
insert into table t1 values (1,3,2,1,1),(2,2,1,3,1),(3,1,1,1,1),(2,1,1,0,1),(1,1,1,0,1);
insert into table t2 values (1,3,5,0,1),(2,2,3,1,1),(3,1,1,0,1),(2,1,1,0,1),(1,1,1,0,1);
--场景一：not in后的表达式为简单的SELECT多列语句。
select a, b from t1 where (c, d) not in (select a, b from t2 where e = t1.e);
--返回结果如下。
+------------+------------+
| a          | b          |
+------------+------------+
| 2          | 1          |
| 1          | 1          |
+------------+------------+
--场景二：not in后的表达式使用聚合函数。
select a, b from t1 where (c, d) not in (select max(a), b from t2 where e = t1.e group by b having max(a) > 0);
--返回结果如下。
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 3          |
| 3          | 1          |
| 2          | 1          |
| 1          | 1          |
+------------+------------+
--场景三：not in后的表达式为常量。
select a, b from t1 where (c, d) not in ((1, 3), (1, 1));
--返回结果如下。
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 3          |
| 2          | 1          |
| 1          | 1          |
+------------+------------+
```

EXISTS SUBQUERY

使用 `exists subquery` 时，当子查询中有至少一行数据时，返回True，否则返回False。

MaxCompute只支持含有Correlated条件的 `where` 子查询。 `exists subquery` 实现的方式是转换为 `left semi join` 。

- 命令格式

```
select <select_expr> from <table_name1> where exists (select <select_expr> from <table_name2> where <table_name2_colname> = <t
able_name1>.<colname>);
```

- 参数说明
  - select_expr：必填。格式为 `col1_name, col2_name, 正则表达式,...` ，表示待查询的普通列、分区列或正则表达式。
  - table_name1、table_name2：必填。表的名称。
  - col_name：必填。表的列名。
- 使用示例

```
select * from sale_detail where exists (select * from shop where customer_id = sale_detail.customer_id);
--等效于以下语句。
select * from sale_detail a left semi join shop b on a.customer_id = b.customer_id;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

NOT EXISTS SUBQUERY

使用 `not exists subquery` 时，当子查询中无数据时，返回True，否则返回False。

MaxCompute只支持含有Correlated条件的 `where` 子查询。 `not exists subquery` 实现的方式是转换为 `left anti join` 。

- 命令格式

```
select <select_expr> from <table_name1> where not exists (select <select_expr> from <table_name2> where <table_name2_colname>
= <table_name1>.<colname>);
```

- 参数说明
  - select_expr：必填。格式为 `col1_name, col2_name, 正则表达式, ...` ，表示待查询的普通列、分区列或正则表达式。
  - table_name1、table_name2：必填。表的名称。
  - col_name：必填。表的列名。
- 使用示例

```
select * from sale_detail where not exists (select * from shop where shop_name = sale_detail.shop_name);
--等效于以下语句。
select * from sale_detail a left anti join shop b on a.shop_name = b.shop_name;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
+------------+-------------+-------------+------------+------------+
```

SCALAR SUBQUERY

当子查询的输出结果为单行单列时，可以做为标量使用，即可以参与标量运算。

所有的满足一行一列输出值的子查询，都可以按照如下命令格式重写。如果查询的结果只有一行，在外面嵌套一层 `max` 或 `min` 操作，其结果不变。

- 命令格式

```
select <select_expr> from <table_name1> where (<select count(*) from <table_name2> where <table_name2_colname> = <table_name1>
.<colname>) <标量运算符> <scalar_value>;
--等效于以下语句。
select <table_name1>.<select_expr> from <table_name1> left semi join (select <colname>, count(*) from <table_name2> group by <
colname> having count(*) <标量运算符> <scalar_value>) <table_name2> on <table_name1>.<colname> = <table_name2>.<colname>;
```

> 说明
>
> - `select count(*) from <table_name2> where <table_name2_colname> = <table_name1>.<colname>` 的输出结果是一个Row Set，可以判断出这条语句的输出有且仅有一行一列。因此它可以作为标量。但在实现过程中，会尽可能地转成 `join` 来处理。
> - 可以作为标量的子查询必须是在编译阶段就能够确认其返回结果只有一行一列的查询。如果一个子查询只能在实际运行过程中才能判断出它只产生一行数据（而无法在编译过程中判断），则编译器会报错。目前编译器能够接受的语句需满足两个特征：
>   - 子查询的 `select` 列表里面用了聚合函数，且不在表值函数的参数列表中。
>   - 子查询中包含聚合函数的这一层查询没有 `group by` 语句。

在上述能力及限制的基础上，SCALAR SUBQUERY还支持多列用法如下：

- SELECT列为包含多列的SCALAR SUBQUERY表达式，只支持等值表达式。
- SELECT列可以为BOOLEAN表达式，只支持等值比较。
- `where` 支持多列比较，只支持等值比较。

- 参数说明
  - select_expr：必填。格式为 `col1_name, col2_name, 正则表达式, ...` ，表示待查询的普通列、分区列或正则表达式。
  - table_name1、table_name2：必填。表的名称。
  - col_name：必填。表的列名。
  - 标量运算符：必填。例如大于（>）、小于（<）、等于（=）、大于等于（>=）或小于等于（<=）等。
  - scalar_value：必填。标量值
- 使用限制
  - `scalar subquery` 支持引用外层查询的列，当嵌套多层 `scalar subquery` 时，只支持引用直接外层的列。

```
--允许的操作。
select * from t1 where (select count(*) from t2 where t1.a = t2.a) = 3;
--不允许的操作，不能在子查询的select中引用外部查询的列。
select * from t1 where (select count(*) from t2 where (select count(*) from t3 where t3.a = t1.a) = 2) = 3;
```

- `scalar subquery`只能在`where`子句中使用。

```
--不能在子查询的select中引用。
select * from t1 where (select t1.b + count(*) from t2) = 3;
--select返回列不允许引用外层的列。
select (select count(t1.a) from t2 where t2.a = t1.a) from t1;
select (select t1.a from t2 where t2.a = t1.a) from t1;
```

- 使用示例
  - 示例1：常见用法，命令示例如下。

```
select * from shop where (select count(*) from sale_detail where sale_detail.shop_name = shop.shop_name) >= 1;
```

返回结果如下：

```
+------------+-------------+-------------+
| shop_name  | customer_id | total_price |
+------------+-------------+-------------+
| s1         | c1          | 100.1       |
| s2         | c2          | 100.2       |
| s3         | c3          | 100.3       |
| null       | c5          | NULL        |
| s6         | c6          | 100.4       |
| s7         | c7          | 100.5       |
+------------+-------------+-------------+
```

  - 示例2：SELECT多列场景。命令示例如下。

```
--为方便理解，此处重新构造示例数据。
create table if not exists ts(a bigint,b bigint,c double);
create table if not exists t(a bigint,b bigint,c double);
insert into table ts values (1,3,4.0),(1,3,3.0);
insert into table t values (1,3,4.0),(1,3,5.0);
--场景一：SELECT列为包含多列的SCALAR SUBQUERY表达式，只支持等值表达式。错误用法：select (select a, b from t where c > ts.c) as (a, b
), a from ts;
select (select a, b from t where c = ts.c) as (a, b), a from ts;
--返回结果如下。
+------------+------------+------------+
| a          | b          | a2         |
+------------+------------+------------+
| 1          | 3          | 1          |
| NULL       | NULL       | 1          |
+------------+------------+------------+
--场景二：SELECT列为BOOLEAN表达式，只支持等值比较。错误用法：select (a,b) > (select a,b from ts where c = t.c) from t;
select (a,b) = (select a,b from ts where c = t.c) from t;
--返回结果如下。
+------+
| _c0  |
+------+
| true |
| false |
+------+
--场景三：where支持多列比较，只支持等值比较。错误用法：select * from t where (a,b) > (select a,b from ts where c = t.c);
select * from t where c > 3.0 and (a,b) = (select a,b from ts where c = t.c);
--返回结果如下。
+------------+------------+------------+
| a          | b          | c          |
+------------+------------+------------+
| 1          | 3          | 4.0        |
+------------+------------+------------+
select * from t where c > 3.0 or (a,b) = (select a,b from ts where c = t.c);
--返回结果如下。
+------------+------------+------------+
| a          | b          | c          |
+------------+------------+------------+
| 1          | 3          | 4.0        |
| 1          | 3          | 5.0        |
+------------+------------+------------+
```

3.6.4. 交集、并集和补集

您可以通过MaxCompute对查询结果数据集执行取交集、并集或补集操作。本文为您介绍交集（`intersect`、`intersect all`、`intersect distinct`）、并集（`union`、`union all`、`union distinct`）和补集（`except`、`except all`、`except distinct`、`minus`、`minus all`、`minus distinct`）的使用方法。

功能介绍

MaxCompute支持如下三种操作：

- 交集：求两个数据集的交集，即输出两个数据集均包含的记录。
- 并集：求两个数据集的并集，即将两个数据集合并成一个数据集。
- 补集：求第二个数据集在第一个数据集中的补集，即输出第一个数据集包含而第二个数据集不包含的记录。

使用限制

对数据集取交集、并集或补集的使用限制如下：

- MaxCompute最多允许同时对256个数据集进行操作，超出256个将报错。
- 左右两个数据集的列数必须保持一致。

注意事项

对数据集取交集、并集或补集的注意事项如下：

- 对数据集进行操作的结果不一定会按序排列。
- 如果数据集的数据类型不一致，系统会进行隐式转换。由于兼容性原因，STRING类型和非STRING类型数据在集合操作中的隐式转换已被禁用。

交集

- 命令格式

```
--取交集不去重。
<select_statement1> intersect all <select_statement2>;
--取交集并去重。intersect效果等同于intersect distinct。
<select_statement1> intersect [distinct] <select_statement2>;
```

- 参数说明
  - select_statement1、select_statement2：必填。`select`语句，格式请参见SELECT语法。
  - distinct：可选。对两个数据集取交集的结果去重。
- 使用示例
  - 示例1：对两个数据集取交集，不去重。命令示例如下：

```
select * from values (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
intersect all
select * from values (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);
```

返回结果如下：

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 1          | 2          |
| 3          | 4          |
+------------+------------+
```

  - 示例2：对两个查询结果取交集并去重。命令示例如下：

```
select * from values (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
intersect distinct
select * from values (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);
--等效于如下语句。
select distinct * from
(select * from values (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
intersect all
select * from values (1, 2), (1, 2), (3, 4), (5, 7) t(a, b)) t;
```

返回结果如下：

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 3          | 4          |
+------------+------------+
```

并集

- 命令格式

```
--取并集不去重。
<select_statement1> union all <select_statement2>;
--取并集并去重。
<select_statement1> union [distinct] <select_statement2>;
```

- 注意事项
  - 存在多个 `union all` 时，支持通过括号指定 `union all` 的优先级。
  - `union` 后如果有 `cluster by` 、 `distribute by` 、 `sort by` 、 `order by` 或 `limit` 子句时，如果设置 `set odps.sql.type.system.odps2=false;` ，其作用于 `union` 的最后一个 `select_statement` ；如果设置 `set odps.sql.type.system.odps2=true;` 时，作用于前面所有 `union` 的结果。
- 参数说明
  - select_statement1、select_statement2：必填。 `select` 语句，格式请参见SELECT语法。
  - distinct：可选。对两个数据集取并集的结果去重。
- 使用示例
  - 示例1：对两个数据集取并集，不去重。命令示例如下：

```
select * from values (1, 2), (1, 2), (3, 4) t(a, b)
union all
select * from values (1, 2), (1, 4) t(a, b);
```

返回结果如下：

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 1          | 2          |
| 3          | 4          |
| 1          | 2          |
| 1          | 4          |
+------------+------------+
```

  - 示例2：对两个数据集取并集并去重。命令示例如下：

```
select * from values (1, 2), (1, 2), (3, 4) t(a, b)
union distinct
select * from values (1, 2), (1, 4) t(a, b);
--等效于如下语句。
select distinct * from (
select * from values (1, 2), (1, 2), (3, 4) t(a, b)
union all
select * from values (1, 2), (1, 4) t(a, b));
```

返回结果如下：

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 1          | 4          |
| 3          | 4          |
+------------+------------+
```

- 示例3：通过括号指定 `union all` 的优先级。命令示例如下：

```
select * from values (1, 2), (1, 2), (5, 6) t(a, b)
union all
(select * from values (1, 2), (1, 2), (3, 4) t(a, b)
union all
select * from values (1, 2), (1, 4) t(a, b));
```

返回结果如下：

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 1          | 2          |
| 5          | 6          |
| 1          | 2          |
| 1          | 2          |
| 3          | 4          |
| 1          | 2          |
| 1          | 4          |
+------------+------------+
```

- 示例4：`union` 后有 `cluster by`、`distribute by`、`sort by`、`order by` 或 `limit` 子句，设置 `set odps.sql.type.system.odps2=true;` 属性。命令示例如下：

```
set odps.sql.type.system.odps2=true;
select explode(array(3, 1)) as (a) union all select explode(array(0, 4, 2)) as (a) order by a limit 3;
```

返回结果如下：

```
+------------+
| a          |
+------------+
| 0          |
| 1          |
| 2          |
+------------+
```

- 示例5：`union` 后有 `cluster by`、`distribute by`、`sort by`、`order by` 或 `limit` 子句，设置 `set odps.sql.type.system.odps2=false;` 属性。命令示例如下：

```
set odps.sql.type.system.odps2=false;
select explode(array(3, 1)) as (a) union all select explode(array(0, 4, 2)) as (a) order by a limit 3;
```

返回结果如下：

```
+------------+
| a          |
+------------+
| 3          |
| 1          |
| 0          |
| 2          |
| 4          |
+------------+
```

补集

- 命令格式

```
--取补集不去重。
<select_statement1> except all <select_statement2>;
<select_statement1> minus all <select_statement2>;
--取补集并去重。
<select_statement1> except [distinct] <select_statement2>;
<select_statement1> minus [distinct] <select_statement2>;
```

> 说明 `except` 和 `minus` 等效。

- 参数说明
  - select_statement1、select_statement2：必填。`select` 语句，格式请参见SELECT语法。
  - distinct：可选。对取补集的结果去重。
- 使用示例

- 示例1：求数据集的补集，不去重。命令示例如下：

```
select * from values (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
except all
select * from values (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
--等效于如下语句。
select * from values (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
minus all
select * from values (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
```

返回结果如下。

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 1          | 2          |
| 3          | 4          |
| 7          | 8          |
+------------+------------+
```

- 示例2：求数据集的补集并去重。命令示例如下：

```
select * from values (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
except distinct
select * from values (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
--等效于如下语句。
select * from values (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
minus distinct
select * from values (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
--等效于如下语句。
select distinct * from values (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b) except all select * from values (3, 4),
(5, 6), (5, 6), (9, 10) t(a, b);
```

返回结果如下：

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 7          | 8          |
+------------+------------+
```

3.6.5. JOIN

MaxCompute支持通过 `join` 操作连接表并返回符合连接条件和查询条件的数据。本文为您介绍左连接、右连接、全连接、内连接、自然连接、隐式连接和多路连接的使用方法。

功能介绍

MaxCompute支持如下 `join` 操作：

- 左连接（ `left outer join` ）

  可简写为 `left join` 。返回左表中的所有记录，即使右表中没有与之匹配的记录。

  > **说明** 通常， `join` 操作的左边为大表，右表为小表，如果右表值不唯一，建议不要连续使用过多 `left join` ，以免在 `join` 过程中产生数据膨胀，导致作业停滞。

- 右连接（ `right outer join` ）

  可简写为 `right join` 。返回右表中的所有记录，即使左表中没有与之匹配的记录。

- 全连接（ `full outer join` ）

  可简写为 `full join` 。返回左右表中的所有记录。

- 内连接（ `inner join` ）

  关键字 `inner` 可以省略。左右表中至少存在一个匹配行时， `inner join` 返回数据行。

- 自然连接（ `natural join` ）

  参与 `join` 的两张表根据字段名称自动决定连接字段。支持 `outer natural join` ，支持使用 `using` 子句执行 `join` ，输出字段中公共字段只出现一次。

- 隐式连接

即不指定 `join` 关键字执行连接。

- 多路连接

  多路 `join` 连接。支持通过括号指定 `join` 的优先级，括号内的 `join` 优先级较高。

> 说明 如果SQL语句中包含 `where` 过滤条件，且 `join` 在 `where` 条件之前，先进行 `join` 操作，然后对 `join` 的结果执行 `where` 条件过滤，获取的结果是两个表的交集，而不是全表。

使用限制

`join` 操作的使用限制如下：

- MaxCompute不支持 `cross join` ，即无 `on` 条件的连接。
- 只允许出现 `and` 连接的等值条件。您可以通过 `mapjoin` 操作使用不等值连接或 `or` 连接多个条件，详情请参见MAPJOIN。

命令格式

```
<table_reference> join <table_factor> [<join_condition>]
| <table_reference> {left outer|right outer|full outer|inner|natural} join <table_reference> <join_condition>
```

- table_reference：必填。待执行 `join` 操作的左表查询语句。格式为 `table_name [alias] | table_query [alias] |...` 。
- table_factor：必填。待执行 `join` 操作的右表或表查询语句。格式为 `table_name [alias] | table_subquery [alias] |...` 。
- join_condition：可选。 `join` 连接条件，是一个或多个等式表达式组合。格式为 `on equality_expression [and equality_expression]...` ， `equality_expression` 为等式表达式。

> 说明 如果分区裁剪条件置于 `where` 子句中，分区裁剪会生效；如果置于 `on` 子句中，从表的分区裁剪会生效，主表的分区剪裁不会生效即会全表扫描。详情请参见分区剪裁合理性评估。

示例数据

为便于理解，本文为您提供源数据，基于源数据提供相关示例。创建表sale_detail和sale_detail_jt，并添加数据，命令示例如下：

```
--创建分区表sale_detail和sale_detail_jt。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
create table if not exists sale_detail_jt
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china') partition (sale_date='2014', region='shanghai');
alter table sale_detail_jt add partition (sale_date='2013', region='china');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3)
;
insert into sale_detail partition (sale_date='2014', region='shanghai') values ('null','c5',null),('s6','c6',100.4),('s7','c7',10
0.5);
insert into sale_detail_jt partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s5','c2',100
.2);
--创建做关联的表。
create table shop as select shop_name, customer_id, total_price from sale_detail;
```

使用示例

- 示例1：左连接。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--由于表sale_detail_jt及sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区分。
select a.shop_name as ashop, b.shop_name as bshop from sale_detail_jt a
       left outer join sale_detail b on a.shop_name=b.shop_name;
```

返回结果如下：

```
+------------+------------+
| ashop      | bshop      |
+------------+------------+
| s2         | s2         |
| s1         | s1         |
| s5         | NULL       |
+------------+------------+
```

- 示例2：右连接。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--由于表sale_detail_jt及sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区分。
select a.shop_name as ashop, b.shop_name as bshop from sale_detail_jt a
       right outer join sale_detail b on a.shop_name=b.shop_name;
```

返回结果如下：

```
+------------+------------+
| ashop      | bshop      |
+------------+------------+
| NULL       | s3         |
| NULL       | s6         |
| NULL       | null       |
| s2         | s2         |
| NULL       | s7         |
| s1         | s1         |
+------------+------------+
```

- 示例3：全连接。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--由于表sale_detail_jt及sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区分。
select a.shop_name as ashop, b.shop_name as bshop from sale_detail_jt a
       full outer join sale_detail b on a.shop_name=b.shop_name;
```

返回结果如下：

```
+------------+------------+
| ashop      | bshop      |
+------------+------------+
| NULL       | s3         |
| NULL       | s6         |
| s2         | s2         |
| NULL       | null       |
| NULL       | s7         |
| s1         | s1         |
| s5         | NULL       |
+------------+------------+
```

- 示例4：内连接。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--由于表sale_detail_jt及sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区分。
select a.shop_name as ashop, b.shop_name as bshop from sale_detail_jt a
       inner join sale_detail b on a.shop_name=b.shop_name;
```

返回结果如下：

```
+------------+------------+
| ashop      | bshop      |
+------------+------------+
| s2         | s2         |
| s1         | s1         |
+------------+------------+
```

- 示例5：自然连接。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--自然连接。
select * from sale_detail_jt natural join sale_detail;
--等效于如下语句。
select sale_detail_jt.shop_name as shop_name, sale_detail_jt.customer_id as customer_id, sale_detail_jt.total_price as total_p
rice, sale_detail_jt.sale_date as sale_date, sale_detail_jt.region as region from sale_detail_jt
inner join sale_detail
on sale_detail_jt.shop_name=sale_detail.shop_name and sale_detail_jt.customer_id=sale_detail.customer_id and sale_detail_jt.to
tal_price=sale_detail.total_price and sale_detail_jt.sale_date=sale_detail.sale_date and sale_detail_jt.region=sale_detail.reg
ion;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- 示例6：隐式连接。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--隐式连接。
select * from sale_detail_jt, sale_detail where sale_detail_jt.shop_name = sale_detail.shop_name;
--等效于如下语句。
select * from sale_detail_jt join sale_detail on sale_detail_jt.shop_name = sale_detail.shop_name;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+------------+-------------+-------------+------------+--
----------+
| shop_name  | customer_id | total_price | sale_date  | region     | shop_name2 | customer_id2 | total_price2 | sale_date2 | r
egion2    |
+------------+-------------+-------------+------------+------------+------------+-------------+-------------+------------+--
----------+
| s2         | c2          | 100.2       | 2013       | china      | s2         | c2           | 100.2        | 2013       | c
hina      |
| s1         | c1          | 100.1       | 2013       | china      | s1         | c1           | 100.1        | 2013       | c
hina      |
+------------+-------------+-------------+------------+------------+------------+-------------+-------------+------------+--
----------+
```

- 示例7：多路连接，不指定优先级。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--由于表sale_detail_jt及sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区分。
select a.* from sale_detail_jt a full outer join sale_detail b on a.shop_name=b.shop_name
        full outer join sale_detail c on a.shop_name=c.shop_name;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s5         | c2          | 100.2       | 2013       | china      |
| NULL       | NULL        | NULL        | NULL       | NULL       |
| NULL       | NULL        | NULL        | NULL       | NULL       |
| NULL       | NULL        | NULL        | NULL       | NULL       |
| NULL       | NULL        | NULL        | NULL       | NULL       |
| NULL       | NULL        | NULL        | NULL       | NULL       |
| NULL       | NULL        | NULL        | NULL       | NULL       |
| s1         | c1          | 100.1       | 2013       | china      |
| NULL       | NULL        | NULL        | NULL       | NULL       |
| s2         | c2          | 100.2       | 2013       | china      |
| NULL       | NULL        | NULL        | NULL       | NULL       |
+------------+-------------+-------------+------------+------------+
```

- 示例7：多路连接，通过括号指定优先级。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--多路连接，通过括号指定优先级。
select * from shop join (sale_detail_jt join sale_detail on sale_detail_jt.shop_name = sale_detail.shop_name) on shop.shop_nam
e=sale_detail_jt.shop_name;
```

返回结果如下：

```
+------------+-------------+-------------+------------+------------+------------+-------------+-------------+------------+--
----------+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     | shop_name2 | customer_id2 | total_price2 | sale_date2 | r
egion2     | shop_name3 | customer_id3 | total_price3 | sale_date3 | region3    |
+------------+-------------+-------------+------------+------------+------------+-------------+-------------+------------+--
----------+------------+-------------+-------------+------------+------------+
| s2         | c2          | 100.2       | 2013       | china      | s2         | c2          | 100.2        | 2013       | c
hina       | s2         | c2           | 100.2        | 2013       | china      |
| s1         | c1          | 100.1       | 2013       | china      | s1         | c1          | 100.1        | 2013       | c
hina       | s1         | c1           | 100.1        | 2013       | china      |
+------------+-------------+-------------+------------+------------+------------+-------------+-------------+------------+--
----------+------------+-------------+-------------+------------+------------+
```

- 示例8：`join` 与 `where` 相结合，查询两表中region为china且shop_name一致的记录数，保留sale_detail表的全部记录。命令示例如下：

```
--分区表需要开启全表扫描功能，否则join操作会执行失败。
set odps.sql.allow.fullscan=true;
--执行SQL语句。
select a.shop_name
       ,a.customer_id
       ,a.total_price
       ,b.total_price
from    (select * from sale_detail where region = "china") a
left join  (select * from sale_detail_jt where region = "china") b
on      a.shop_name = b.shop_name;
```

返回结果如下：

```
+------------+-------------+-------------+--------------+
| shop_name  | customer_id | total_price | total_price2 |
+------------+-------------+-------------+--------------+
| s1         | c1          | 100.1       | 100.1        |
| s2         | c2          | 100.2       | 100.2        |
| s3         | c3          | 100.3       | NULL         |
+------------+-------------+-------------+--------------+
```

错误命令示例如下：

```
select a.shop_name
       ,a.customer_id
       ,a.total_price
       ,b.total_price
from    sale_detail a
left join  sale_detail_jt b
on      a.shop_name = b.shop_name
where   a.region = "china" and b.region = "china";
```

返回结果如下：

```
+------------+-------------+-------------+--------------+
| shop_name  | customer_id | total_price | total_price2 |
+------------+-------------+-------------+--------------+
| s1         | c1          | 100.1       | 100.1        |
| s2         | c2          | 100.2       | 100.2        |
+------------+-------------+-------------+--------------+
```

从返回结果可看到，获取的结果是两个表的交集，非sale_detail表的全部记录。

3.6.6. SEMI JOIN（半连接）

MaxCompute支持半连接操作，通过右表过滤左表的数据，右表的数据不出现在结果集中。本文为您介绍半连接中 `left semi join` 和 `left anti join` 两种语法的使用方法。

功能介绍

MaxCompute支持如下 `semi join` 操作：

- `left semi join`

当 `join` 条件成立时，返回左表中的数据。如果左表中满足指定条件的某行数据在右表中出现过，则此行保留在结果集中。

在MaxCompute中，与 `left semi join` 类似的操作为 `in subquery` ，请参见IN SUBQUERY。您可以自行选择其中一种方式。

- `left anti join`

  当 `join` 条件不成立时，返回左表中的数据。如果左表中满足指定条件的某行数据没有在右表中出现过，则此行保留在结果集中。

  在MaxCompute中，与 `left anti join` 类似的操作为 `not in subquery` ，但并不完全相同，请参见NOT IN SUBQUERY。

`semi join` 支持 `mapjoin hint` ，可以提高 `left semi join` 和 `left anti join` 的性能。更多 `mapjoin hint` 信息，请参见MAPJOIN HINT。

示例数据

为便于理解，本文为您提供源数据，基于源数据提供相关示例。创建表sale_detail和sale_detail_sj，并添加数据，命令示例如下：

```
--创建一张分区表sale_detail。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
create table if not exists sale_detail_sj
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china');
alter table sale_detail_sj add partition (sale_date='2013', region='china');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3)
;
insert into sale_detail_sj partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s5','c2',100
.2),('s2','c2',100.2);
```

使用示例

- 示例1：查询sale_detail表中，total_price出现在sale_detail_sj表中的数据集。命令示例如下：

  ```
  select * from sale_detail a left semi join sale_detail_sj b on a.total_price=b.total_price;
  ```

  返回结果如下：

  ```
  +------------+-------------+-------------+------------+------------+
  | shop_name  | customer_id | total_price | sale_date  | region     |
  +------------+-------------+-------------+------------+------------+
  | s2         | c2          | 100.2       | 2013       | china      |
  | s1         | c1          | 100.1       | 2013       | china      |
  +------------+-------------+-------------+------------+------------+
  ```

  只会返回sale_detail中的数据，只要sale_detail的total_price在sale_detail_sj的total_price中**出现过**。

- 示例2：查询sale_detail表中，total_price没有出现在sale_detail_sj表中的数据集。命令示例如下：

  ```
  select * from sale_detail a left anti join sale_detail_sj b on a.total_price=b.total_price;
  ```

  返回结果如下：

  ```
  +------------+-------------+-------------+------------+------------+
  | shop_name  | customer_id | total_price | sale_date  | region     |
  +------------+-------------+-------------+------------+------------+
  | s3         | c3          | 100.3       | 2013       | china      |
  +------------+-------------+-------------+------------+------------+
  ```

  只会返回sale_detail中的数据，只要sale_detail的total_price在sale_detail_sj的total_price中**没有出现过**。

3.6.7. MAPJOIN HINT

当您对一个大表和一个或多个小表执行 `join` 操作时，可以在 `select` 语句中显式指定 `mapjoin` Hint提示以提升查询性能。本文为您介绍如何通过 `mapjoin hint` 连接表。

功能介绍

整个JOIN过程包含Map、Shuffle和Reduce三个阶段。通常情况下，`join`操作在Reduce阶段执行表连接。

`mapjoin`在Map阶段执行表连接，而非等到Reduce阶段才执行表连接，可以缩短大量数据传输时间，提升系统资源利用率，从而起到优化作业的作用。

在对大表和一个或多个小表执行`join`操作时，`mapjoin`会将您指定的小表全部加载到执行`join`操作的程序的内存中，在Map阶段完成表连接从而加快`join`的执行速度。

此外，MaxCompute SQL不支持在普通`join`的`on`条件中使用不等值表达式、`or`等逻辑复杂的`join`条件，但是在`mapjoin`中可以进行上述操作。

使用限制

`mapjoin`操作的使用限制如下：

- `mapjoin`在Map阶段会将指定表的数据全部加载在内存中，因此指定的表仅能为小表，且表被加载到内存后占用的总内存不得超过512 MB。由于MaxCompute是压缩存储，因此小表在被加载到内存后，数据大小会急剧膨胀。此处的512 MB是指加载到内存后的空间大小。
- `mapjoin`中`join`操作的限制如下：
  - `left outer join`的左表必须是大表。
  - `right outer join`的右表必须是大表。
  - 不支持`full outer join`。
  - `inner join`的左表或右表均可以是大表。
- `mapjoin`最多支持指定128张小表，否则报语法错误。

使用方法

您需要在`select`语句中使用Hint提示`/*+ mapjoin(<table_name>) */`才会执行`mapjoin`。需要注意的是：

- 引用小表或子查询时，需要引用别名。
- `mapjoin`支持小表为子查询。
- 在`mapjoin`中，可以使用不等值连接或`or`连接多个条件。您可以通过不写`on`语句而通过`mapjoin on 1 = 1`的形式，实现笛卡尔乘积的计算。例如`select /*+ mapjoin(a) */ a.id from shop a join table_name b on 1=1;`，但此操作可能带来数据量膨胀问题。
- `mapjoin`中多个小表用英文逗号（,）分隔，例如`/*+ mapjoin(a,b,c)*/`。

示例数据

为便于理解，本文为您提供源数据，基于源数据提供相关示例。创建表sale_detail和sale_detail_sj，并添加数据，命令示例如下：

```
--创建一张分区表sale_detail。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
create table if not exists sale_detail_sj
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china');
alter table sale_detail_sj add partition (sale_date='2013', region='china');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3)
;
insert into sale_detail_sj partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s5','c2',100
.2),('s2','c2',100.2);
```

使用示例

对表sale_detail和sale_detail_sj执行`join`操作，满足sale_detail_sj中的total_price小于sale_detail中的total_price或sale_detail_sj中的total_price与sale_detail中的total_price之和小于500的条件，命令示例如下：

```
select /*+ mapjoin(a) */
       a.shop_name,
       a.total_price,
       b.total_price
from sale_detail_sj a join sale_detail b
on a.total_price < b.total_price or a.total_price + b.total_price < 500;
```

返回结果如下：

```
+-----------+-------------+--------------+
| shop_name | total_price | total_price2 |
+-----------+-------------+--------------+
| s1        | 100.1       | 100.1        |
| s2        | 100.2       | 100.1        |
| s5        | 100.2       | 100.1        |
| s2        | 100.2       | 100.1        |
| s1        | 100.1       | 100.2        |
| s2        | 100.2       | 100.2        |
| s5        | 100.2       | 100.2        |
| s2        | 100.2       | 100.2        |
| s1        | 100.1       | 100.3        |
| s2        | 100.2       | 100.3        |
| s5        | 100.2       | 100.3        |
| s2        | 100.2       | 100.3        |
+-----------+-------------+--------------+
```

`mapjoin` 多个小表示例，请参见如何用MAPJOIN缓存多张小表？。

3.6.8. DISTRIBUTED MAPJOIN

Distributed MapJoin是MapJoin的升级版，适用于小表Join大表的场景，二者的核心目的都是为了减少大表侧的Shuffle和排序。

注意事项

- Join两侧的表数据量要求不同，大表侧数据在10 TB以上，小表侧数据在[1 GB, 100 GB]范围内。
- 小表侧的数据需要均匀分布，没有明显的长尾，否则单个分片会产生过多的数据，导致OOM（Out Of Memory）及RPC（Remote Procedure Call）超时问题。
- SQL任务运行时间在20分钟以上，建议使用Distributed MapJoin进行优化。
- 由于在执行任务时，需要占用较多的资源，请避免在较小的Quota组运行。

> **说明** 您可以在配额页面，修改配额组，详情请参见设置配额组。

使用方法

您需要在 `select` 语句中使用Hint提示 `/*+distmapjoin(<table_name>(shard_count=<n>,replica_count=<m>))*/` 才会执行 `distmapjoin` 。shard_count和replica_count共同决定任务运行的并发度，即 `并发度=shard_count * replica_count` 。

- 参数说明
  - table_name：目标表名。
  - shard_count=<n>：设置小表数据的分片数，小表数据分片会分布至各个计算节点处理。n即为分片数，一般按奇数设置。

    > **说明**
    > - shard_count值建议手动指定，shard_count值可以根据小表数据量来大致估算，预估一个分片节点处理的数据量范围是[200 MB, 500 MB]。
    > - shard_count设置过大，性能和稳定性会受影响；shard_count设置过小，会因内存使用过多而报错。

  - replica_count=<m>：设置小表数据的副本数。m即为副本数，默认为1。

    > **说明** 为了减少访问压力以及避免单个节点失效导致整个任务失败，同一个分片的数据，可以有多个副本。当并发过多，或者环境不稳定导致运行节点频繁重启，可以适当提高replica_count，一般建议为2或3。

- 语法示例

```
-- 推荐，指定shard_count（replica_count默认为1）
/*+distmapjoin(a(shard_count=5))*/
-- 推荐，指定shard_count和replica_count
/*+distmapjoin(a(shard_count=5,replica_count=2))*/
-- distmapjoin多个小表
/*+distmapjoin(a(shard_count=5,replica_count=2),b(shard_count=5,replica_count=2)) */
-- distmapjoin和mapjoin混用
/*+distmapjoin(a(shard_count=5,replica_count=2)),mapjoin(b)*/
```

使用示例

为了便于理解，本文以向分区表tmall_dump_lasttable插入数据为例，为您演示Distributed MapJoin的用法。

- 常规写法。

```
insert OVERWRITE table tmall_dump_lasttable partition(ds='20211130')
select t1.*
from
(
    select nid, doc,type
    from search_ods.dump_lasttable where ds='20211203'
)t1
join
(
    select distinct item_id
    from tbcdm.dim_tb_itm
    where ds='20211130'
    and bc_type='B'
    and is_online='Y'
)t2
on t1.nid=t2.item_id;
```

- 优化后写法。

```
insert OVERWRITE table tmall_dump_lasttable partition (ds='20211130')
select /*+ distmapjoin(t2(shard_count=35)) */ t1.*
from
(
    select nid, doc, type
    from search_ods.dump_lasttable where ds='20211203'
)t1
join
(
    select distinct item_id
    from tbcdm.dim_tb_itm
    where ds='20211130'
    and bc_type='B'
    and is_online='Y'
)t2
on t1.nid=t2.item_id;
```

3.6.9. SKEWJOIN HINT

当两张表Join存在热点，导致出现长尾问题时，您可以通过取出热点key，将数据分为热点数据和非热点数据两部分处理，最后合并的方式，提高Join效率。SkewJoin Hint可以通过自动或手动方式获取两张表的热点key，分别计算热点数据和非热点数据的Join结果并合并，加快Join的执行速度。

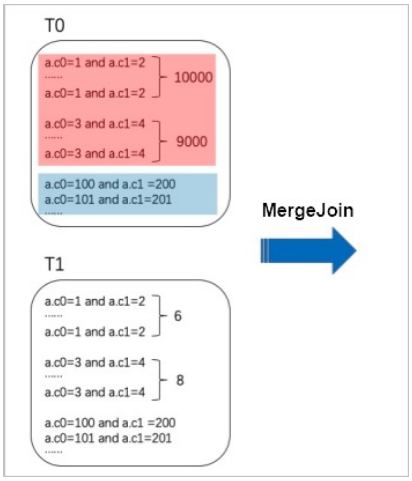
使用方法

您需要在 `select` 语句中使用Hint提示 `/*+ skewJoin(<table_name>[(<column1_name>[,<column2_name>,...])][((<value11>,<value12>)[,(<value21>,<value22>)...])]*/` 才会执行MapJoin。table_name为倾斜表名，column_name为倾斜列名，value为倾斜key值。

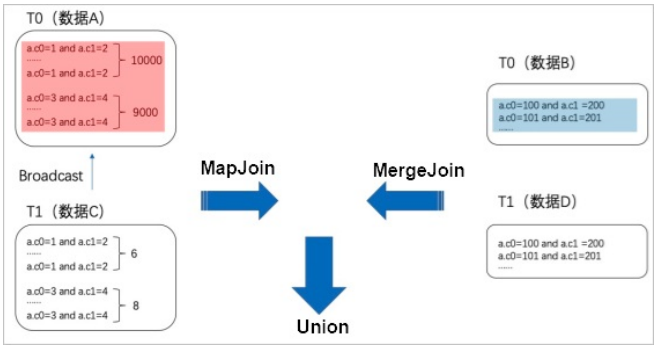
> 说明

实现原理

热值Key指出现次数很多的key值。例如下图中红色部分，`a.c0=1 and a.c1=2`有10000行，`a.c0=3 and a.c1 = 4`有9000行。



在不加SkewJoin Hint的情况下，将表T0和表T1进行Join，由于T0和T1的数量都很大，只能进行MergeJoin，因此相同的热值都会Shuffle到一个节点，导致数据倾斜。加SkewJoin Hint后，优化器会运行一个Aggregate动态获取重复行数前20的热值，并将表T0中属于热值的值（数据A）、T0中不属于热值的值（数据B）拆分；将表T1中能与T0中热值Join的值（数据C）、表T1中不能与T0中热值Join的值（数据D）进行拆分。然后将数据A与数据C进行MapJoin（由于数据C量很少，可以进行MapJoin），将数据B和数据D进行MergeJoin。最后将MapJoin和MergeJoin的结果Union，生成最后的结果，如下图所示。



注意事项

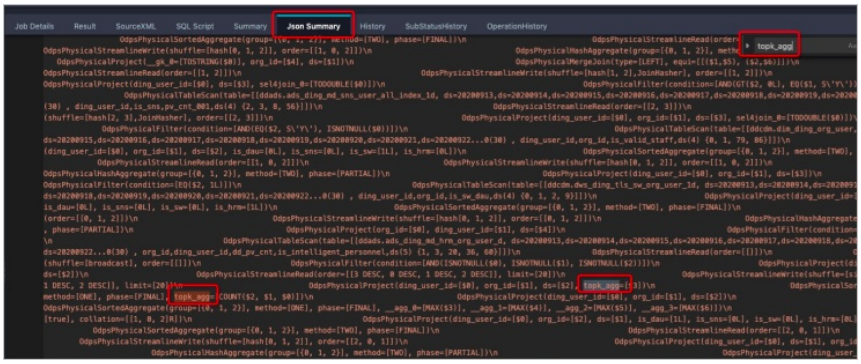
- SkewJoin Hint支持的Join类型：
  - Inner Join可以对Join两侧表中的任意一侧进行Hint。
  - Left Join、Semi Join和Anti Join只可以Hint左侧表。
  - Right Join只可以Hint右侧表。
  - Full Join不支持Skew Join Hint。
- 建议只对一定会出现数据倾斜的Join添加Hint，因为Hint会运行一个Aggregate，存在一定代价。
- 被Hint的Join的Left Side Join Key的类型需要与Right Side Join Key的类型一致，否则SkewJoin Hint不生效。例如上例中的a.c0与b.c0的类型需要一致，a.c1与b.c1的类型需要一致。您可以通过在子查询中将Join key进行Cast从而保持一致。示例如下：

```
create table T0(c0 int, c1 int, c2 int, c3 int);
create table T1(c0 string, c1 int, c2 int);
--方法1：
select /*+ skewjoin(a) */ * from T0 a join T1 b on cast(a.c0 as string) = cast(b.c0 as string) and a.c1 = b.c1;
--方法2：
select /*+ skewjoin(b) */ * from (select cast(a.c0 as string) as c00 from T0 a) b join T1 c on b.c00 = c.c0;
```

- 加SkewJoin Hint后，优化器会运行一个Aggregate获取前20的热值。20是默认值，您可以通过`set odps.optimizer.skew.join.topk.num = xx;`进行设置。
- SkewJoin Hint只支持对Join其中一侧进行Hint。
- 被Hint的Join一定要有`left key = right key`，不支持笛卡尔积Join。
- 与其它Hint一起使用的方法如下，但需要注意，被MapJoin Hint的Join不能再添加SkewJoin Hint。

```
select /*+ mapjoin(c), skewjoin(a) */ * from T0 a join T1 b on a.c0 = b.c3 join T2 c on a.c0 = c.c7;
```

- 您可以在Logview的**Json Summary**中搜索是否出现topk_agg字段判断SkewJoin Hint是否生效，示例如下：



3.6.10. 动态过滤器（Dynamic Filter）

JOIN是分布式系统中常见的操作，同时也是一个耗时、耗资源的操作，因为其涉及到的Shuffle操作尤其在海量数据场景下，会耗费较多的资源和时间。针对Shuffle操作，MaxCompute可以利用JOIN本身的等值连接属性进行优化。

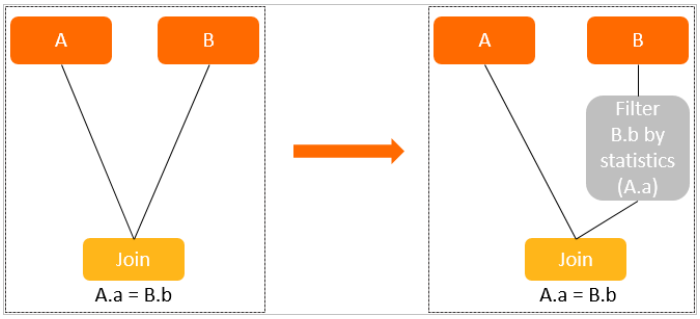
优化思路

一个典型的包含JOIN的SQL语句如下：

```
select * from (table1) A join (table2) B on A.a= B.b;
```

基于JOIN等值连接的特性，MaxCompute可以通过表A的数据生成一个过滤器，在Shuffle或JOIN之前提前过滤表B的数据，甚至可以将过滤器下推到底层存储，在源头过滤数据。这种在作业运行时动态生成过滤器的功能称为动态过滤器DF（Dynamic Filter）。

上述SQL语句在打开动态过滤器前后的执行计划示意图如下。



应用场景

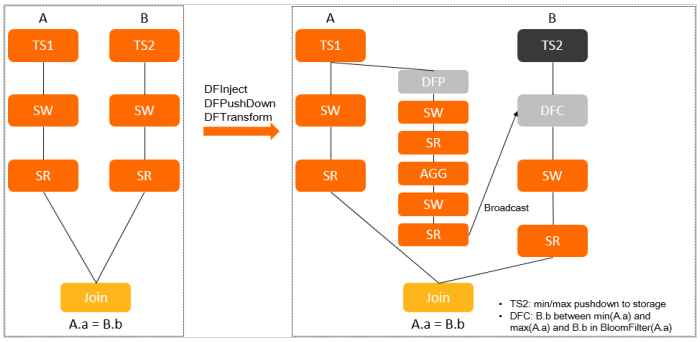
动态过滤器功能是利用等值JOIN的特性，基于运行时动态生成过滤器，以便在Shuffle或JOIN之前提前过滤数据，实现加速查询运行。该功能适用于维度表和事实表执行JOIN的场景。

动态范围过滤器或布隆过滤器（Dynamic Range|Bloom Filter）

从上图可知，在原始的执行计划中不存在过滤器，过滤器是由系统根据JOIN的特性自动产生的，它的作用就是判断B表中的元素是否存在于A表生成的集合中，如不存在，则过滤掉。

从空间和时间效率上看，Bloom Filter正是上述功能的有效选择。但在实现中，动态过滤器不仅包含Bloom Filter，还可以包含Range Filter，即利用 `[min, max]` 来过滤数据。

动态过滤器是一个典型的生产者和消费者模式，示意图如下。



- DFP（Dynamic Filter Producer）算子：动态过滤器的生产者（Producer），利用小表侧的数据生成Bloom Filter及获取JOIN Key对应的 `min` 、 `max` 值（Range Filter），然后发送至DFC。

- DFC（Dynamic Filter Consumer）算子：动态过滤器的消费者（Consumer），利用Bloom Filter和Range Filter过滤大表侧的数据。Range Filter会尽可能将过滤条件下推到底层存储，以便从源头过滤数据。

对于不同类型的JOIN语义，JOIN对象可担任的角色不同：

- A JOIN B：A或B都能作为生产者、消费者。
- A LEFT JOIN B：A只能作为生产者，B只能作为消费者。
- A RIGHT JOIN B：A只能作为消费者，B只能作为生产者。
- A FULL OUTER JOIN B：无法使用动态过滤器功能。

动态过滤器的使用方法请参见动态过滤器的使用方法。

动态分区裁剪（Dynamic Partition Pruning）

上述Bloom Filter或Range Filter的例子是基于非分区表的优化，即JOIN Key是非分区列。当JOIN Key为分区列时，动态范围过滤器或布隆过滤器（Dynamic Range|Bloom Filter）仍然可用，但MaxCompute会读取完整个分区的数据后再过滤数据，读取分区数据的过程可以进一步优化。即在读取数据前，将无用的分区裁剪掉，即动态分区裁剪DPP（Dynamic Partition Pruning）功能。

例如包含JOIN的SQL语句如下：

```
--A为非分区表，表中a列的值为20200701。
--B为分区表，表中ds列的值包含3个分区20200701、20200702、20200703。
select * from (table1) A join (table2) B on A.a= B.ds;
```

打开动态分区裁剪功能后，优化器会根据表是否是分区表来决定是否采用动态分区裁剪功能。动态分区裁剪功能生效后，MaxCompute会采集小表侧数据生成Bloom Filter，然后过滤大表侧的分区列表，再把需要读取的分区列表聚合，裁剪掉不需要扫描的分区。如果一个运行进程所有待读的分区都被裁剪了，则该进程不被调度。

在上述示例中，由于A表中的a列值只有20200701，因而打开动态分区裁剪功能后，B表中的20200702和20200703分区会被裁剪掉，既节省了资源，也降低了作业运行时长。

动态分区裁剪功能的使用方法请参见动态分区裁剪的使用方法。

动态过滤器的使用方法

MaxComoputе提供了如下打开动态过滤器的方式：

- 方式一：在Session级别通过开关强制打开动态过滤器，与SQL语句一起提交执行。

  ```
  set odps.optimizer.force.dynamic.filter=true;
  ```

  > **说明** 该属性也支持在Project级别设置，但推荐您在Session级别设置。由于Project级别开启后，对所有JOIN作业都会启用动态过滤器，当JOIN无法过滤数据时，处理效率反而更低。

  使用该方式时，对所有能打开动态过滤器的JOIN，都插入动态过滤器。

- 方式二：在Session级别通过开关智能打开动态过滤器。

  ```
  set odps.optimizer.enable.dynamic.filter=true;
  ```

  使用该方式时，优化器会智能地估计插入动态过滤器是否有足够的资源或时间获益，如果有收益则插入动态过滤器，否则不会插入。

  > **说明** 该方式依赖元数据统计，例如ndv，更多元数据统计信息，请参见优化器信息收集。因为元数据统计是优化器的估算结果，可能不准确，因此会存在无法如预期地插入动态过滤器。

- 方式三：在SQL语句中通过HINT方式打开动态过滤器。

  HINT格式为 `/*+dynamicfilter(Producer, Consumer1[, Consumer2,...])*/` ，允许一个生产者过滤多个消费者。命令示例如下：

  ```
  select /*+dynamicfilter(A, B)*/ * from (table1) A join (table2) B on A.a= B.b;
  ```

动态分区裁剪的使用方法

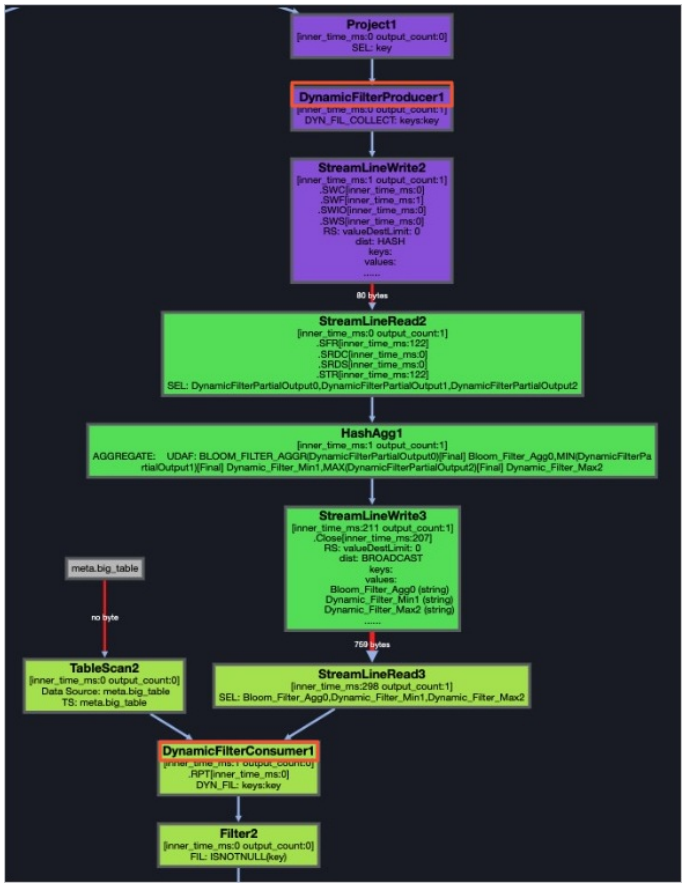
在Session级别通过开关打开动态分区裁剪功能，与SQL语句一起提交执行。

```
set odps.optimizer.dynamic.filter.dpp.enable=true;
```

> **说明** 该属性也支持在Project级别设置，但推荐您在Session级别设置。由于Project级别开启后，对所有JOIN作业中涉及到的分区表都会启用动态分区裁剪功能，当JOIN无法过滤数据时，处理效率反而更低。

验证方法

如果您已按照动态过滤器的使用方法或动态分区裁剪的使用方法完成配置，运行SQL作业后，查看作业的Logview信息。如果Logview中出现类似DynamicFilterConsumer1的算子，说明动态过滤器已生效。



3.6.11. Lateral View

MaxCompute支持通过Lateral View与UDTF（表生成函数）结合，将单行数据拆成多行数据。本文为您介绍如何使用Lateral View拆分行数据，并执行聚合操作。

功能介绍

直接在 `select` 语句中使用UDTF会存在限制，为解决此问题，您可以通过MaxCompute的Lateral View与UDTF结合使用，将一行数据拆成多行数据，并对拆分后的数据进行聚合。

当您定义的UDTF不输出任何一行时，对应的输入行在Lateral View结果中依然保留，且所有UDTF输出列为NULL。

命令格式

```
lateralView: lateral view [outer] <udtf_name>(<expression>) <table_alias> as <columnAlias> (',' <columnAlias>)
fromClause: from <baseTable> (lateralView) [(lateralView) ...]
```

- udtf_name：必填。将一行数据拆成多行数据的UDTF，请参见其他函数。
- expression：必填。待拆分行数据所属列名。
- table_alias：必填。UDTF结果集的别名。
- columnAlias：必填。拆分后得到的列的别名。
- baseTable：必填。数据源表。

> 说明
>
> `from` 后可以有多个Lateral View语句，后面的Lateral View语句能够引用它前面的所有表和列名，实现对不同列的行数据进行拆分。

示例数据

假设已有一张表pageAds，它有三列数据，第一列是pageid string，第二列是col1 array<int>，第三列是col2 array<string>，详细数据如下。

pageid	col1	col2
front_page	[1, 2, 3]	[“a” , “b” , “c”]

pageid	col1	col2
contact_page	[3, 4, 5]	[“d” , “e” , “f”]

使用示例

- 单个Lateral View语句
  - 示例1：拆分col1。命令示例如下：

```
select pageid, col1_new, col2 from pageAds lateral view explode(col1) adTable as col1_new;
```

返回结果如下：

```
+------------+------------+------------+
| pageid     | col1_new   | col2       |
+------------+------------+------------+
| front_page | 1          | ["a","b","c"] |
| front_page | 2          | ["a","b","c"] |
| front_page | 3          | ["a","b","c"] |
| contact_page | 3          | ["d","e","f"] |
| contact_page | 4          | ["d","e","f"] |
| contact_page | 5          | ["d","e","f"] |
+------------+------------+------------+
```

  - 示例2：拆分col1并执行聚合统计。命令示例如下：

```
select col1_new, count(1) as count from pageAds lateral view explode(col1) adTable as col1_new group by col1_new;
```

返回结果如下：

```
+------------+------------+
| col1_new   | count      |
+------------+------------+
| 1          | 1          |
| 2          | 1          |
| 3          | 2          |
| 4          | 1          |
| 5          | 1          |
+------------+------------+
```

- 多个Lateral View语句

拆分col1和col2。命令示例如下：

```
select pageid,mycol1, mycol2 from pageAds
    lateral view explode(col1) myTable1 as mycol1
    lateral view explode(col2) myTable2 as mycol2;
```

返回结果如下：

```
+------------+------------+------------+
| pageid     | mycol1     | mycol2     |
+------------+------------+------------+
| front_page | 1          | a          |
| front_page | 1          | b          |
| front_page | 1          | c          |
| front_page | 2          | a          |
| front_page | 2          | b          |
| front_page | 2          | c          |
| front_page | 3          | a          |
| front_page | 3          | b          |
| front_page | 3          | c          |
| contact_page | 3          | d          |
| contact_page | 3          | e          |
| contact_page | 3          | f          |
| contact_page | 4          | d          |
| contact_page | 4          | e          |
| contact_page | 4          | f          |
| contact_page | 5          | d          |
| contact_page | 5          | e          |
| contact_page | 5          | f          |
+------------+------------+------------+
```

相关参考

在实际业务开发过程中，如果您遇到行转列或列转行需求，除了可以借鉴上述Lateral View方法外，还可以参见行转列及列转行最佳实践。

3.6.12. HAVING子句

MaxCompute SQL的WHERE关键字无法与聚合函数一起使用，此时您可以使用HAVING子句来实现。

命令格式如下。

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

例如有一张订单表 `Orders` ，包括客户名称（ `Customer` ）、订单金额（ `OrderPrice` ）、订单日期（ `Order_date` ）和订单号（ `Order_id` ）四个字段。现在需要查找订单总额少于2000的客户，SQL语句如下。

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
```

3.6.13. GROUPING SETS

对于经常需要对数据进行多维度的聚合分析的场景，您既需要对A列做聚合，也要对B列做聚合，同时要对A、B两列做聚合，因此需要多次使用 `union all` 。您可以使用 `grouping sets` 快速解决此类问题。本文为您介绍如何使用 `grouping sets` 进行多维聚合。

功能介绍

`grouping sets` 是对 `select` 语句中 `group by` 子句的扩展，允许您采用多种方式对结果分组，而不必使用多个 `select` 语句再 `union all` 来实现。这样能够使MaxCompute的引擎给出更有效的执行计划，从而提高执行性能。

与 `grouping sets` 相关联的语法如下。

类型	说明

类型	说明
cube	特殊的 grouping sets，将指定列的所有可能组合作为 grouping sets，也可以与 grouping sets 组合使用。 group by cube (a, b, c) --等效于以下语句。 grouping sets ((a,b,c),(a,b),(a,c),(b,c),(a),(b),(c),()) group by cube ((a, b), (c, d)) --等效于以下语句。 grouping sets ((a, b, c, d), (a, b), (c, d), ( )) group by a, cube (b, c), grouping sets ((d), (e)) --等效于以下语句。 group by grouping sets ((a, b, c, d), (a, b, c, e), (a, b, d), (a, b, e), (a, c, d), (a, c, e), (a, d), (a, e))
rollup	特殊的 grouping sets，以按层级聚合的方式产生 grouping sets，也可以与 grouping sets 组合使用。 group by rollup (a, b, c) --等效价于以下语句。 grouping sets ((a,b,c),(a,b),(a), ()) group by rollup (a, (b, c), d) --等效于以下语句。 grouping sets ((a, b, c, d), (a, b, c), (a), ( )) group by grouping sets((b), (c), rollup(a,b,c)) --等效于以下语句。 group by grouping sets ((b), (c), (a,b,c), (a,b), (a), ())
grouping	grouping sets 结果中使用NULL充当占位符，导致您会无法区分占位符NULL与数据中真正的NULL。因此，MaxCompute为您提供了 grouping 。 grouping 接受一个列名作为参数，如果结果对应行使用了参数列做聚合，返回0，此时意味着NULL来自输入数据。否则返回1，此时意味着NULL是 grouping sets 的占位符。
grouping_id	接受一个或多个列名作为参数。结果是将参数列的 grouping 结果按照Bitmap的方式组成整数。

类型	说明
`grouping__id`	`grouping__id` 不带参数，用于兼容Hive查询。此表达方式在MaxCompute中等价于 `grouping_id(group by参数列表)` ，参数与 `group by` 的顺序一致。 **说明** MaxCompute和Hive 2.3.0及以上版本兼容该函数，在Hive 2.3.0以下版本中该函数输出不一致，因此并不推荐您使用此函数。

GROUPING SETS使用示例

`grouping sets` 使用示例如下：

1. 准备数据。

```
create table requests lifecycle 20 as
select * from values
    (1, 'windows', 'PC', 'Beijing'),
    (2, 'windows', 'PC', 'Shijiazhuang'),
    (3, 'linux', 'Phone', 'Beijing'),
    (4, 'windows', 'PC', 'Beijing'),
    (5, 'ios', 'Phone', 'Shijiazhuang'),
    (6, 'linux', 'PC', 'Beijing'),
    (7, 'windows', 'Phone', 'Shijiazhuang')
as t(id, os, device, city);
```

2. 对数据进行分组。您可以通过如下两种方式进行分组：
   - 使用多个 `select` 语句进行分组。

     ```
     select NULL, NULL, NULL, count(*)
     from requests
     union all
     select os, device, NULL, count(*)
     from requests group by os, device
     union all
     select null, null, city, count(*)
     from requests group by city;
     ```

   - 使用 `grouping sets` 进行分组。

     ```
     select os,device, city ,count(*)
     from requests
     group by grouping sets((os, device), (city), ());
     ```

     返回结果如下：

     ```
     +------------+------------+------------+------------+
     | os         | device     | city       | _c3        |
     +------------+------------+------------+------------+
     | NULL       | NULL       | NULL       | 7          |
     | NULL       | NULL       | Beijing    | 4          |
     | NULL       | NULL       | Shijiazhuang | 3          |
     | ios        | Phone      | NULL       | 1          |
     | linux      | PC         | NULL       | 1          |
     | linux      | Phone      | NULL       | 1          |
     | windows    | PC         | NULL       | 3          |
     | windows    | Phone      | NULL       | 1          |
     +------------+------------+------------+------------+
     ```

   说明 分组集中不使用的表达式，会使用NULL充当占位符，使得这些结果集可以做操作。例如结果第4~8行的city列。

CUBE | ROLLUP使用示例

基于 `grouping sets` 示例表， `cube` 及 `rollup` 使用示例如下：

- 示例1：通过 `cube` 枚举os、device、city的所有可能列为 `grouping sets` 。命令示例如下：

```
select os,device, city, count(*)
from requests
group by cube (os, device, city);
--等效于如下语句。
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(os, city),(device,city),(os),(device),(city),());
```

返回结果如下：

```
+------------+------------+------------+------------+
| os         | device     | city       | _c3        |
+------------+------------+------------+------------+
| NULL       | NULL       | NULL       | 7          |
| NULL       | NULL       | Beijing    | 4          |
| NULL       | NULL       | Shijiazhuang | 3          |
| NULL       | PC         | NULL       | 4          |
| NULL       | PC         | Beijing    | 3          |
| NULL       | PC         | Shijiazhuang | 1          |
| NULL       | Phone      | NULL       | 3          |
| NULL       | Phone      | Beijing    | 1          |
| NULL       | Phone      | Shijiazhuang | 2          |
| ios        | NULL       | NULL       | 1          |
| ios        | NULL       | Shijiazhuang | 1          |
| ios        | Phone      | NULL       | 1          |
| ios        | Phone      | Shijiazhuang | 1          |
| linux      | NULL       | NULL       | 2          |
| linux      | NULL       | Beijing    | 2          |
| linux      | PC         | NULL       | 1          |
| linux      | PC         | Beijing    | 1          |
| linux      | Phone      | NULL       | 1          |
| linux      | Phone      | Beijing    | 1          |
| windows    | NULL       | NULL       | 4          |
| windows    | NULL       | Beijing    | 2          |
| windows    | NULL       | Shijiazhuang | 2          |
| windows    | PC         | NULL       | 3          |
| windows    | PC         | Beijing    | 2          |
| windows    | PC         | Shijiazhuang | 1          |
| windows    | Phone      | NULL       | 1          |
| windows    | Phone      | Shijiazhuang | 1          |
+------------+------------+------------+------------+
```

- 示例2：通过 `cube` 枚举 `(os, device),(device, city)` 所有可能列为 `grouping sets` 。命令示例如下：

```
select os,device, city, count(*)
from requests
group by cube ((os, device), (device, city));
--等效于如下语句。
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(device,city),());
```

返回结果如下：

```
+------------+------------+------------+------------+
| os         | device     | city       | _c3        |
+------------+------------+------------+------------+
| NULL       | NULL       | NULL       | 7          |
| NULL       | PC         | Beijing    | 3          |
| NULL       | PC         | Shijiazhuang | 1          |
| NULL       | Phone      | Beijing    | 1          |
| NULL       | Phone      | Shijiazhuang | 2          |
| ios        | Phone      | NULL       | 1          |
| ios        | Phone      | Shijiazhuang | 1          |
| linux      | PC         | NULL       | 1          |
| linux      | PC         | Beijing    | 1          |
| linux      | Phone      | NULL       | 1          |
| linux      | Phone      | Beijing    | 1          |
| windows    | PC         | NULL       | 3          |
| windows    | PC         | Beijing    | 2          |
| windows    | PC         | Shijiazhuang | 1          |
| windows    | Phone      | NULL       | 1          |
| windows    | Phone      | Shijiazhuang | 1          |
+------------+------------+------------+------------+
```

- 示例3：通过 `rollup` 对os、device、city以按层级聚合的方式产生 `grouping sets` 。命令示例如下：

```
select os,device, city, count(*)
from requests
group by rollup (os, device, city);
--等效于如下语句。
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(os),());
```

返回结果如下：

```
+------------+------------+------------+------------+
| os         | device     | city       | _c3        |
+------------+------------+------------+------------+
| NULL       | NULL       | NULL       | 7          |
| ios        | NULL       | NULL       | 1          |
| ios        | Phone      | NULL       | 1          |
| ios        | Phone      | Shijiazhuang | 1          |
| linux      | NULL       | NULL       | 2          |
| linux      | PC         | NULL       | 1          |
| linux      | PC         | Beijing    | 1          |
| linux      | Phone      | NULL       | 1          |
| linux      | Phone      | Beijing    | 1          |
| windows    | NULL       | NULL       | 4          |
| windows    | PC         | NULL       | 3          |
| windows    | PC         | Beijing    | 2          |
| windows    | PC         | Shijiazhuang | 1          |
| windows    | Phone      | NULL       | 1          |
| windows    | Phone      | Shijiazhuang | 1          |
+------------+------------+------------+------------+
```

- 示例4：通过 `rollup` 对 `os, (os,device), city` 以按层级聚合的方式产生 `grouping sets` 。命令示例如下：

```
select os,device, city, count(*)
from requests
group by rollup (os, (os,device), city);
--等效于如下语句。
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(os),());
```

返回结果如下：

```
+------------+------------+------------+------------+
| os         | device     | city       | _c3        |
+------------+------------+------------+------------+
| NULL       | NULL       | NULL       | 7          |
| ios        | NULL       | NULL       | 1          |
| ios        | Phone      | NULL       | 1          |
| ios        | Phone      | Shijiazhuang | 1          |
| linux      | NULL       | NULL       | 2          |
| linux      | PC         | NULL       | 1          |
| linux      | PC         | Beijing    | 1          |
| linux      | Phone      | NULL       | 1          |
| linux      | Phone      | Beijing    | 1          |
| windows    | NULL       | NULL       | 4          |
| windows    | PC         | NULL       | 3          |
| windows    | PC         | Beijing    | 2          |
| windows    | PC         | Shijiazhuang | 1          |
| windows    | Phone      | NULL       | 1          |
| windows    | Phone      | Shijiazhuang | 1          |
+------------+------------+------------+------------+
```

- 示例5：通过 `group by` 、 `cube` 、 `grouping sets` 组合产生 `grouping sets` 。命令示例如下：

```
select os,device, city, count(*)
from requests
group by os, cube(os,device), grouping sets(city);
--等效于如下语句。
select os,device, city, count(*)
from requests
group by grouping sets((os,device,city),(os,city),(os,device,city));
```

返回结果如下：

```
+------------+------------+------------+------------+
| os         | device     | city       | _c3        |
+------------+------------+------------+------------+
| ios        | NULL       | Shijiazhuang | 1          |
| ios        | Phone      | Shijiazhuang | 1          |
| linux      | NULL       | Beijing    | 2          |
| linux      | PC         | Beijing    | 1          |
| linux      | Phone      | Beijing    | 1          |
| windows    | NULL       | Beijing    | 2          |
| windows    | NULL       | Shijiazhuang | 2          |
| windows    | PC         | Beijing    | 2          |
| windows    | PC         | Shijiazhuang | 1          |
| windows    | Phone      | Shijiazhuang | 1          |
+------------+------------+------------+------------+
```

GROUPING | GROUPING_ID使用示例

`grouping` 及 `grouping_id` 使用示例如下：

```
select a,b,c,count(*),
grouping(a) ga, grouping(b) gb, grouping(c) gc, grouping_id(a,b,c) groupingid
from values (1,2,3) as t(a,b,c)
group by cube(a,b,c);
```

返回结果如下：

```
+------------+------------+------------+------------+------------+------------+------------+------------+
| a          | b          | c          | _c3        | ga         | gb         | gc         | groupingid |
+------------+------------+------------+------------+------------+------------+------------+------------+
| NULL       | NULL       | NULL       | 1          | 1          | 1          | 1          | 7          |
| NULL       | NULL       | 3          | 1          | 1          | 1          | 0          | 6          |
| NULL       | 2          | NULL       | 1          | 1          | 0          | 1          | 5          |
| NULL       | 2          | 3          | 1          | 1          | 0          | 0          | 4          |
| 1          | NULL       | NULL       | 1          | 0          | 1          | 1          | 3          |
| 1          | NULL       | 3          | 1          | 0          | 1          | 0          | 2          |
| 1          | 2          | NULL       | 1          | 0          | 0          | 1          | 1          |
| 1          | 2          | 3          | 1          | 0          | 0          | 0          | 0          |
+------------+------------+------------+------------+------------+------------+------------+------------+
```

默认情况，`group by` 列表中不被使用的列，会被填充为NULL。您可以通过 `grouping` 输出更有实际意义的值。基于 `grouping sets` 示例表，命令示例如下：

```
select
  if(grouping(os) == 0, os, 'ALL') as os,
  if(grouping(device) == 0, device, 'ALL') as device,
  if(grouping(city) == 0, city, 'ALL') as city,
  count(*) as count
from requests
group by os, device, city grouping sets((os, device), (city), ());
```

返回结果如下：

```
+------------+------------+------------+------------+
| os         | device     | city       | count      |
+------------+------------+------------+------------+
| ALL        | ALL        | ALL        | 7          |
| ALL        | ALL        | Beijing    | 4          |
| ALL        | ALL        | Shijiazhuang | 3          |
| ios        | Phone      | ALL        | 1          |
| linux      | PC         | ALL        | 1          |
| linux      | Phone      | ALL        | 1          |
| windows    | PC         | ALL        | 3          |
| windows    | Phone      | ALL        | 1          |
+------------+------------+------------+------------+
```

GROUPING__ID使用示例

`grouping__id` 不带参数的命令示例如下：

```
select
a, b, c, count(*), grouping__id
from values (1,2,3) as t(a,b,c)
group by a, b, c grouping sets ((a,b,c), (a));
--等效于如下语句。
select
a, b, c, count(*), grouping_id(a,b,c)
from values (1,2,3) as t(a,b,c)
group by a, b, c grouping sets ((a,b,c), (a));
```

返回结果如下：

```
+------------+------------+------------+------------+------------+
| a          | b          | c          | _c3        | _c4        |
+------------+------------+------------+------------+------------+
| 1          | NULL       | NULL       | 1          | 3          |
| 1          | 2          | 3          | 1          | 0          |
+------------+------------+------------+------------+------------+
```

3.6.14. SELECT TRANSFORM

`select transform`语法允许您启动一个指定的子进程，将输入数据按照一定的格式通过标准输入至子进程，并且通过解析子进程的标准输出获取输出数据。`select transform`让您无需编写UDF，便可以实现MaxCompute SQL对其他脚本语言的支持。

功能介绍

`select transform`与UDTF在不同场景下的性能不同。经过多种场景对比测试，数据量较小时，大多数场景下`select transform`有优势，而数据量大时UDTF有优势。`select transform`的开发更加简便，更适合于Ad Hoc（即席查询）。

`select transform`不仅仅是语言支持的扩展。一些简单的功能，例如AWK、Python、Perl、Shell都支持直接在命令中写脚本，而不需要专门编写脚本文件、上传资源等，开发过程更简单。对于复杂的功能，您可以上传脚本文件来执行，请参见调用Python脚本使用示例、调用Java脚本使用示例。

UDTF与`select transform`的优势对比如下。

类型	`select transform`	UDTF
数据类型	子进程基于标准输入和标准输出传输数据，所有数据都当做STRING类型处理，因此`select transform`比UDTF多了一步类型转换。	UDTF的输出结果和输入参数支持多种数据类型。
数据传输	数据传输依赖于操作系统的管道，而管道的缓存仅有4 KB且不能设置。`select transform`读空或管道写满会导致进程被挂起。 数据的传输通过更底层的系统调用来读写，效率比Java高。	无管道缓存限制。
常量参数传输	常量参数需要传输。	常量参数可以不用传输。
线程	子进程和父进程是两个进程，如果计算占比较高，数据吞吐量较小，`select transform`可以利用服务器的多核特性。	单线程。
性能	`select transform`支持的部分工具，例如AWK是Native代码实现的。理论上，与Java相比，使用`select transform`会更有性能优势。	性能不高。

使用限制

由于MaxCompute计算集群上未部署PHP和Ruby，所以不支持调用这两种脚本。

命令格式

```
select transform(<arg1>, <arg2> ...)
[(row format delimited (fields terminated by <field_delimiter> (escaped by <character_escape>)) (lines separated by <line_separat
or>) (null defined as <null_value>))]
using '<unix_command_line>'
(resources '<res_name>' (',' '<res_name>') *)
[(as <col1>, <col2> ...)]
(row format delimited (fields terminated by <field_delimiter> (escaped by <character_escape>))
(lines separated by <line_separator>)
(null defined as <null_value>)
```

- select transform关键字：必填。可以用 `map` 或 `reduce` 关键字替换，语义是完全一样的。为使语法更清晰，推荐您使用select transform。
- arg1,arg2...：必填。指定输入数据。其格式和 `select` 语句类似。默认格式下，参数的各个表达式结果在隐式转换成STRING类型后，用 `\t` 拼接，输入到子进程中。
- row fromat子句：可选。允许自定义输入输出的格式。

  语法中有两个row fromat子句，第一个子句指定输入数据的格式，第二个子句指定输出数据的格式。默认情况下使用 `\t` 作为列分隔符，`\n` 作为行分隔符，使用 `\N` 表示NULL。

  > 说明
  > - field_delimiter、character_escape和line_separator只接受一个字符。如果指定的是字符串，则以第一个字符为准。
  > - MaxCompute支持Hive指定格式的语法，例如 `inputRecordReader` 、`outputRecordReader` 、`SerDe` 等，但您需要打开Hive兼容模式才能使用。打开方式为在SQL语句前加set语句 `set odps.sql.hive.compatible=true;` 。Hive支持的语法详情请参见Hive文档。
  > - 如果使用Hive的 `inputRecordReader` 、`outputRecordReader` 等自定义类，可能会降低执行性能。

- using子句：必填。指定要启动的子进程的命令。
  - 大多数的MaxCompute SQL命令中using子句指定的是资源（Resources），但此处使用using子句指定启动子进程的命令。使用using子句是为了和Hive的语法兼容。
  - using子句的格式和Shell语法类似，但并非真的启动Shell来执行，而是直接根据命令的内容创建子进程。因此，很多Shell的功能不能使用，例如输入输出重定向、管道、循环等。如果有需要，Shell本身也可以作为子进程命令来使用。
- resources子句：可选。允许指定子进程能够访问的资源，支持以下两种方式指定资源：
  - 使用resources子句指定资源。例如 `using 'sh foo.sh bar.txt' resources 'foo.sh','bar.txt'` 。
  - 使用MaxCompute属性指定资源。在SQL语句前使用 `set odps.sql.session.resources=foo.sh,bar.txt;` 来指定资源。

    此配置是全局配置，即整个SQL中所有的select transform都可以访问此资源。多个资源文件之间使用英文逗号（,）分隔。
- as子句：可选。指定输出列。例如 `as(col1:bigint, col2:boolean)` 。
  - 输出列可以不指定数据类型，默认为STRING类型。例如 `as(col1, col2)` 。
  - 由于输出数据实际是解析子进程标准输出获取的，如果指定的数据不是STRING类型，系统会隐式调用 `cast` 函数进行转换，转换过程有可能出现运行异常。
  - 输出列的数据类型不支持部分指定，例如 `as(col1, col2:bigint)` 。
  - 关键字 `as` 可以省略，此时默认标准输出数据中第一个 `\t` 之前的字段为Key，后面的部分全部为Value，相当于 `as(key, value)` 。

调用Shell命令使用示例

假设通过Shell命令生成50行数据，值是从1到50，输出为 `data` 字段。直接将Shell命令作为 `transform` 数据输入。命令示例如下：

```
select transform(script) using 'sh' as (data)
from (
        select  'for i in `seq 1 50`; do echo $i; done' as script
      ) t
;
--等效于如下语句。
select transform('for i in `seq 1 50`; do echo $i; done') using 'sh' as (data);
```

返回结果如下：

```
+------------+
| data       |
+------------+
| 1          |
| 2          |
| 3          |
| 4          |
| 5          |
| 6          |
| 7          |
| 8          |
| 9          |
| 10         |
| 11         |
| 12         |
| 13         |
| 14         |
| 15         |
| 16         |
| 17         |
| 18         |
| 19         |
| 20         |
| 21         |
| 22         |
| 23         |
| 24         |
| 25         |
| 26         |
| 27         |
| 28         |
| 29         |
| 30         |
| 31         |
| 32         |
| 33         |
| 34         |
| 35         |
| 36         |
| 37         |
| 38         |
| 39         |
| 40         |
| 41         |
| 42         |
| 43         |
| 44         |
| 45         |
| 46         |
| 47         |
| 48         |
| 49         |
| 50         |
+------------+
```

调用Python命令使用示例

假设通过Python命令生成50行数据，值是从1到50，输出为 `data` 字段。直接将Python命令作为 `transform` 数据输入。命令示例如下：

```
select transform(script) using 'python' as (data)
from (
        select  'for i in xrange(1, 51):  print i;' as script
      ) t
;
--等效于如下语句。
select transform('for i in xrange(1, 51):  print i;') using 'python' as (data);
```

返回结果如下：

```
+------------+
| data       |
+------------+
| 1          |
| 2          |
| 3          |
| 4          |
| 5          |
| 6          |
| 7          |
| 8          |
| 9          |
| 10         |
| 11         |
| 12         |
| 13         |
| 14         |
| 15         |
| 16         |
| 17         |
| 18         |
| 19         |
| 20         |
| 21         |
| 22         |
| 23         |
| 24         |
| 25         |
| 26         |
| 27         |
| 28         |
| 29         |
| 30         |
| 31         |
| 32         |
| 33         |
| 34         |
| 35         |
| 36         |
| 37         |
| 38         |
| 39         |
| 40         |
| 41         |
| 42         |
| 43         |
| 44         |
| 45         |
| 46         |
| 47         |
| 48         |
| 49         |
| 50         |
+------------+
```

调用AWK命令使用示例

创建一张测试表，假设通过AWK命令将测试表的第二列原样输出，输出为 `data` 字段。直接将AWK命令作为 `transform` 数据输入。命令示例如下：

```
--创建测试表。
create table testdata(c1 bigint,c2 bigint);
--测试表中插入测试数据。
insert into table testdata values (1,4),(2,5),(3,6);
--执行select transform语句。
select transform(*) using "awk '//{print $2}'" as (data) from testdata;
```

返回结果如下：

```
+------------+
| data       |
+------------+
| 4          |
| 5          |
| 6          |
+------------+
```

调用Perl命令使用示例

创建一张测试表，假设通过Perl命令将测试表的数据原样输出，输出为 `data` 字段。直接将Perl命令作为 `transform` 数据输入。命令示例如下：

```
--创建测试表。
create table testdata(c1 bigint,c2 bigint);
--测试表中插入测试数据。
insert into table testdata values (1,4),(2,5),(3,6);
--执行select transform语句。
select transform(testdata.c1, testdata.c2) using "perl -e 'while($input = <STDIN>){print $input;}'" from testdata;
```

返回结果如下：

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1          | 4          |
| 2          | 5          |
| 3          | 6          |
+------------+------------+
```

调用Python脚本使用示例

1. 准备Python文件，脚本文件名为*myplus.py*，命令示例如下。

   ```
   #!/usr/bin/env python
   import sys
   line = sys.stdin.readline()
   while line:
       token = line.split('\t')
       if (token[0] == '\\N') or (token[1] == '\\N'):
           print '\\N'
       else:
           print str(token[0]) +'\t' + str(token[1])
       line = sys.stdin.readline()
   ```

2. 将该Python脚本文件添加为MaxCompute资源（Resource）。

   ```
   add py ./myplus.py -f;
   ```

   > **说明** 您也可通过DataWorks控制台进行新增资源操作，请参见创建MaxCompute资源。

3. 使用 `select transform` 语法调用资源。

   ```
   --创建测试表。
   create table testdata(c1 bigint,c2 bigint);
   --测试表中插入测试数据。
   insert into table testdata values (1,4),(2,5),(3,6);
   --执行select transform语句。
   select
   transform (testdata.c1, testdata.c2)
   using 'python myplus.py'resources 'myplus.py'
   as (result1,result2)
   from testdata;
   --等效于如下语句。
   set odps.sql.session.resources=myplus.py;
   select transform (testdata.c1, testdata.c2)
   using 'python myplus.py'
   as (result1,result2)
   from testdata;
   ```

   返回结果如下：

```
+------------+------------+
| result1    | result2    |
+------------+------------+
| 1          | 4          |
|            | NULL       |
| 2          | 5          |
|            | NULL       |
| 3          | 6          |
|            | NULL       |
+------------+------------+
```

调用Java脚本使用示例

1. 准备好JAR文件，脚本文件名为*Sum.jar*，Java代码示例如下。

```
package com.aliyun.odps.test;
import java.util.Scanner;
public class Sum {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String s = sc.nextLine();
            String[] tokens = s.split("\t");
            if (tokens.length < 2) {
                throw new RuntimeException("illegal input");
            }
            if (tokens[0].equals("\\N") || tokens[1].equals("\\N")) {
                System.out.println("\\N");
            }
            System.out.println(Long.parseLong(tokens[0]) + Long.parseLong(tokens[1]));
        }
    }
}
```

2. 将JAR文件添加为MaxCompute的资源。

```
add jar ./Sum.jar -f;
```

3. 使用 `select transform` 语法调用资源。

```
--创建测试表。
create table testdata(c1 bigint,c2 bigint);
--测试表中插入测试数据。
insert into table testdata values (1,4),(2,5),(3,6);
--执行select transform语句。
select transform(testdata.c1, testdata.c2) using 'java -cp Sum.jar com.aliyun.odps.test.Sum' resources 'Sum.jar' from testdat
a;
--等效于如下语句。
set odps.sql.session.resources=Sum.jar;
select transform(testdata.c1, testdata.c2) using 'java -cp Sum.jar com.aliyun.odps.test.Sum' from testdata;
```

返回结果如下：

```
+-----+
| cnt |
+-----+
| 5   |
| 7   |
| 9   |
+-----+
```

> **说明** Java和Python虽然有现成的UDTF框架，但是用 `select transform` 编写更简单、不需要额外依赖以及没有格式要求，甚至可以实现直接使用离线脚本。Java和Python离线脚本的实际路径，可以从 `JAVA_HOME` 和 `PYTHON_HOME` 环境变量中得到。

串联使用示例

`select transform` 还可以串联使用。例如使用 `distribute by` 和 `sort by` 对输入数据做预处理。命令示例如下：

```
select transform(key, value) using 'cmd2' from
(
    select transform(*) using 'cmd1' from
    (
        select * from testdata distribute by c2 sort by c1
    ) t distribute by key sort by value
) t2;
```

或使用 `map` 、 `reduce` 关键字。

```
@a := select * from data distribute by col2 sort by col1;
@b := map * using 'cmd1' distribute by col1 sort by col2 from @a;
reduce * using 'cmd2' from @b;
```

3.7. SQL增强操作

3.7.1. LOAD

如果您需要将外部存储上的数据导入MaxCompute的表或表的分区中，可以通过LOAD命令实现该操作。本文为您介绍如何使用LOAD命令向MaxCompute中导入位于外部存储上的CSV格式或其它开源格式数据。

执行 `load` 操作前需要具备项目空间创建表权限（CreateTable）及修改表权限（Alter）。授权操作请参见权限列表。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

功能介绍

MaxCompute支持您使用 `load overwrite` 或 `load into` 命令将OSS、Amazon Redshift、BigQuery外部存储的CSV格式或其他开源格式数据导入MaxCompute的表或表的分区。其中：Amazon Redshift和BigQuery的数据需要先导入OSS，才可以通过OSS导入MaxCompute。

MaxCompute也支持将数据按照动态分区方式导入MaxCompute的分区表的分区。

`load into` 命令会直接向表或分区中追加数据。 `load overwrite` 命令会先清空表或分区中的原有数据，再向表或分区中插入数据。

前提条件

在向MaxCompute导入外部存储数据前，您需要先对MaxCompute进行授权。 `load overwrite` 或 `load into` 命令的授权模式沿用了MaxCompute外部表的授权模式，您可以一键授权，具备高安全性，详情请参见STS模式授权。

完成授权后，您需要根据导入数据的格式类型，选择对应的导入方式：

- 通过内置Extractor（StorageHandler）导入数据
- 导入其他开源格式数据

使用限制

- 只支持将外部存储的数据导入至同区域的MaxCompute项目空间中。
- 将数据下载到分区表时，外部路径下数据的Schema不包含分区列。

通过内置Extractor（StorageHandler）导入数据

- 命令格式

```
{load overwrite|into} table <table_name> [partition (<pt_spec>)]
from location <external_location>
stored by <StorageHandler>
[with serdeproperties (<Options>)];
```

- 参数说明
  - table_name：必填。需要插入数据的目标表名称。目标表需要提前创建，目标表的Schema（除分区列）需要和外部数据格式一致。
  - pt_spec：可选。需要插入数据的目标表分区信息。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。
  - external_location：必填。指定读取外部存储数据的OSS目录，格式为 `'oss://<oss_endpoint>/<object>'` ，详情请参见OSS访问域名使用规则，系统会默认读取该目录下所有的文件。
  - StorageHandler：必填。指定内置的StorageHandler名称。 `com.aliyun.odps.CsvStorageHandler` 是内置的处理CSV格式文件的StorageHandler，定义了如何读或写CSV文件。您只需要指定该参数，相关逻辑已经由系统实现。使用方法和MaxCompute外部表一致，详情请参见创建OSS外部表。

- Options：可选。指定外部表相关参数，SERDEPROPERTIES支持的属性和MaxCompute外部表一致，属性列表详情请参见创建OSS外部表。

- 使用示例

  通过内置Extractor（StorageHandler）导入数据。假设MaxCompute和OSS的Owner是同一个账号，通过阿里云内网将vehicle.csv文件的数据导入MaxCompute。

  i. 单击此处完成一键授权。

  ii. 将vehicle.csv文件保存至OSS Bucket目录下 `mc-test/data_location/` ，地域为 `oss-cn-hangzhou` ，并组织OSS目录路径。创建OSS Bucket详情请参见创建存储空间。

  vehicle.csv文件数据如下：

```
1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N
```

  根据Bucket、地域、Endpoint信息组织OSS目录路径如下：

```
oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-test/data_location/
```

  iii. 登录MaxCompute客户端创建目标表ambulance_data_csv_load。命令示例如下：

```
create table ambulance_data_csv_load (
vehicleId INT,
recordId INT,
patientId INT,
calls INT,
locationLatitute DOUBLE,
locationLongtitue DOUBLE,
recordTime STRING,
direction STRING );
```

  iv. 执行 `load overwrite` 命令，将OSS上的vehicle.csv文件导入目标表。命令示例如下：

```
load overwrite table ambulance_data_csv_load
from
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-test/data_location/'
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties (
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole',   --AliyunODPSDefaultRole的ARN信息，可通过RAM角色管理页面获取。
'odps.text.option.delimiter'=','
);
```

  v. 查看目标表ambulance_data_csv_load的导入结果。命令示例如下：

```
--开启全表扫描，仅此Session有效。
set odps.sql.allow.fullscan=true;
select * from ambulance_data_csv_load;
```

  返回结果如下：

```
+------------+------------+------------+------------+------------------+-------------------+------------+------------+
| vehicleid  | recordid   | patientid  | calls      | locationlatitute | locationlongtitue | recordtime | direction  |
+------------+------------+------------+------------+------------------+-------------------+------------+------------+
| 1          | 1          | 51         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | S          |
| 1          | 2          | 13         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | NE         |
| 1          | 3          | 48         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | NE         |
| 1          | 4          | 30         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | W          |
| 1          | 5          | 47         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | S          |
| 1          | 6          | 9          | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | S          |
| 1          | 7          | 53         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | N          |
| 1          | 8          | 63         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | SW         |
| 1          | 9          | 4          | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | NE         |
| 1          | 10         | 31         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | N          |
+------------+------------+------------+------------+------------------+-------------------+------------+------------+
```

导入其他开源格式数据

- 命令格式

```
{load overwrite|into} table <table_name> [partition (<pt_spec>)]
from location <external_location>
[row format serde '<serde_class>'
  [with serdeproperties (<Options>)]
]
stored as <file_format>;
```

- 参数说明
  - table_name：必填。需要插入数据的目标表名称。目标表需要提前创建，目标表的Schema（除分区列）需要和外部数据格式一致。
  - pt_spec：可选。需要插入数据的目标表分区信息。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。
  - external_location：必填。指定读取外部存储数据的OSS目录，格式为 `'oss://<oss_endpoint>/<object>'` ，详情请参见OSS访问域名使用规则，系统会默认读取该目录下所有的文件。
  - serde_class：可选。使用方法和MaxCompute外部表一致，详情请参见创建OSS外部表。
  - Options：必填。指定外部表相关参数，SERDEPROPERTIES支持的属性和MaxCompute外部表一致，属性列表详情请参见创建OSS外部表。
  - file_format：必填。指定导入数据文件格式。例如ORC、PARQUET、RCFILE、SEQUENCEFILE和TEXTFILE。使用方法和MaxCompute外部表一致，详情请参见创建OSS外部表。

    > **说明**
    > - `serde_class` 和 `Options` 使用默认值时，可以省略不写。
    > - 导入的单个文件大小不能超过3 GB，如果文件过大，建议拆分。

- 使用示例
  - 示例1：导入其他开源格式数据。假设MaxCompute和OSS的Owner是同一个账号，通过阿里云内网将vehicle.textfile文件的数据导入MaxCompute。
    a. 单击此处完成一键授权。
    b. 将vehicle.txtfile文件保存至OSS Bucket目录下 `mc-test/data_location/` ，地域为 `oss-cn-hangzhou` ，并组织OSS目录路径。创建OSS Bucket详情请参见创建存储空间。

    vehicle.textfile文件数据如下：

    ```
    1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
    1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
    1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
    1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
    1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
    1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
    1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
    1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
    1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
    1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N
    ```

    根据Bucket、地域、Endpoint信息组织OSS目录路径如下：

    ```
    oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-test/data_location/
    ```

    c. 登录MaxCompute客户端创建目标表ambulance_data_textfile_load_pt。命令示例如下：

    ```
    create table ambulance_data_textfile_load_pt (
    vehicleId STRING,
    recordId STRING,
    patientId STRING,
    calls STRING,
    locationLatitute STRING,
    locationLongtitue STRING,
    recordTime STRING,
    direction STRING)
    partitioned by (ds STRING);
    ```

    d. 执行 `load overwrite` 命令，将OSS上的vehicle.textfile文件导入目标表。命令示例如下：

    ```
    load overwrite table ambulance_data_textfile_load_pt partition(ds='20200910')
    from
    location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-test/data_location/'
    row format serde 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
    stored as textfile;
    ```

e. 查看目标表ambulance_data_textfile_load_pt的导入结果。命令示例如下：

```
--开启全表扫描，仅此Session有效。
set odps.sql.allow.fullscan=true;
select * from ambulance_data_textfile_load_pt;
```

返回结果如下：

```
+------------+------------+------------+------------+------------------+------------------+------------+------------+--
----------+
| vehicleid  | recordid   | patientid  | calls      | locationlatitute | locationlongtitue | recordtime | direction  | d
s         |
+------------+------------+------------+------------+------------------+------------------+------------+------------+--
----------+
| 1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
| 1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N | NULL       | NULL       | NULL       | NULL             | NULL
| NULL       | NULL       | 20200910   |
+------------+------------+------------+------------+------------------+------------------+------------+------------+--
----------+
```

- 示例2：将数据按动态分区方式导入目标表。

> **说明** 如果OSS目录下的子目录是以分区名方式组织的，则可以将数据按动态分区的方式导入到分区表。

a. 单击此处完成一键授权。

b. 将vehicle1.csv文件和vehicle2.csv文件分别保存至OSS Bucket目录 `mc-test/data_location/ds=20200909/` 和 `mc-test/data_location/ds=20200910/` ，地域为 `oss-cn-hangzhou` ，并组织OSS目录路径。创建OSS Bucket详情请参见创建存储空间。

vehicle1.csv文件和vehicle2.csv文件数据如下：

```
--vehicle1.csv
1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
--vehicle2.csv
1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N
```

根据Bucket、地域、Endpoint信息组织OSS目录路径如下：

```
oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-test/data_location/ds=20200909/'
oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-test/data_location/ds=20200910/'
```

c. 登录MaxCompute客户端创建目标表ambulance_data_csv_load_dynpt。命令示例如下：

```
create table ambulance_data_csv_load_dynpt (
vehicleId STRING,
recordId STRING,
patientId STRING,
calls STRING,
locationLatitute STRING,
locationLongtitue STRING,
recordTime STRING,
direction STRING)
partitioned by (ds STRING);
```

d. 执行 `load overwrite` 命令，将OSS上的文件导入目标表。命令示例如下：

```
load overwrite table ambulance_data_csv_load_dynpt partition(ds)
from
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-test/data_location/'
row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
stored as textfile;
```

e. 查看目标表ambulance_data_csv_load_dynpt的导入结果。命令示例如下：

```
--开启全表扫描，仅此Session有效。
set odps.sql.allow.fullscan=true;
select * from ambulance_data_csv_load_dynpt;
```

返回结果如下：

```
+------------+------------+------------+------------+------------------+------------------+------------+------------+--
----------+
| vehicleid  | recordid   | patientid  | calls      | locationlatitute | locationlongtitue | recordtime | direction  | d
s         |
+------------+------------+------------+------------+------------------+------------------+------------+------------+--
----------+
| 1          | 7          | 53         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | N
| 20200909   |
| 1          | 8          | 63         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | SW
| 20200909   |
| 1          | 9          | 4          | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | NE
| 20200909   |
| 1          | 10         | 31         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | N
| 20200909   |
| 1          | 1          | 51         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | S
| 20200910   |
| 1          | 2          | 13         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | NE
| 20200910   |
| 1          | 3          | 48         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | NE
| 20200910   |
| 1          | 4          | 30         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | W
| 20200910   |
| 1          | 5          | 47         | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | S
| 20200910   |
| 1          | 6          | 9          | 1          | 46.81006         | -92.08174         | 9/14/2014 0:00 | S
| 20200910   |
+------------+------------+------------+------------+------------------+------------------+------------+------------+--
----------+
```

3.7.2. UNLOAD

MaxCompute允许您将MaxCompute项目中的数据导出到OSS，方便您通过OSS存储结构化数据，并方便其他计算引擎使用导出至OSS的数据。本文为您介绍如何使用UNLOAD命令向OSS导出MaxCompute上的CSV格式或其他开源格式数据。

本文中的命令您可以在如下工具平台执行：

- MaxCompute客户端
- MaxCompute控制台（查询编辑器）
- DataWorks控制台
- MaxCompute Studio

前提条件

在执行UNLOAD命令前，请确认您已完成如下操作：

- 已开通OSS服务。

更多开通OSS服务信息，请参见开通OSS服务。

- 已具备MaxCompute项目中待导出表数据的读取权限（Select）。

  更多授权信息，请参见权限列表。

使用限制

UNLOAD命令的使用限制如下：

- 导出至OSS的文件的分割方式和文件名称由系统自动生成，不支持自定义导出文件名称或文件后缀。
- 导出的开源格式文件不支持添加后缀名。
- 重复导出不会覆盖已导出的文件，会追加新的导出文件。

注意事项

- UNLOAD命令本身不计费，UNLOAD命令中的查询子句需要扫描数据并使用计算资源计算结果，因此查询子句按照普通SQL作业计费。
- 通过OSS存储结构化数据在一些场景中可以节省存储费用，但需要提前做好费用估算。

  MaxCompute存储费用为0.12元/GB/月，更多存储计费信息，请参见存储费用。数据导入MaxCompute后有5倍左右的压缩率，计费依据的数据量是压缩后的数据。

  OSS存储标准型单价为0.12元/GB/月，另有其他低频访问型、归档型、冷归档型存储，请参见存储费用。

  如果您导出数据只是为了节省存储费用，建议您根据数据特征测试估算压缩率，根据导出时的查询语句估算UNLOAD费用，以及后续对导出数据的访问方式进行合理评估，避免因不必要的数据迁移产生额外费用。

OSS授权模式

在向OSS导出数据时，您需要对OSS进行授权。UNLOAD命令的授权模式沿用了MaxCompute外部表的授权模式，您需要在UNLOAD命令中，配置`with serdeproperties`属性，通过RAM角色完成OSS授权。授权流程如下：

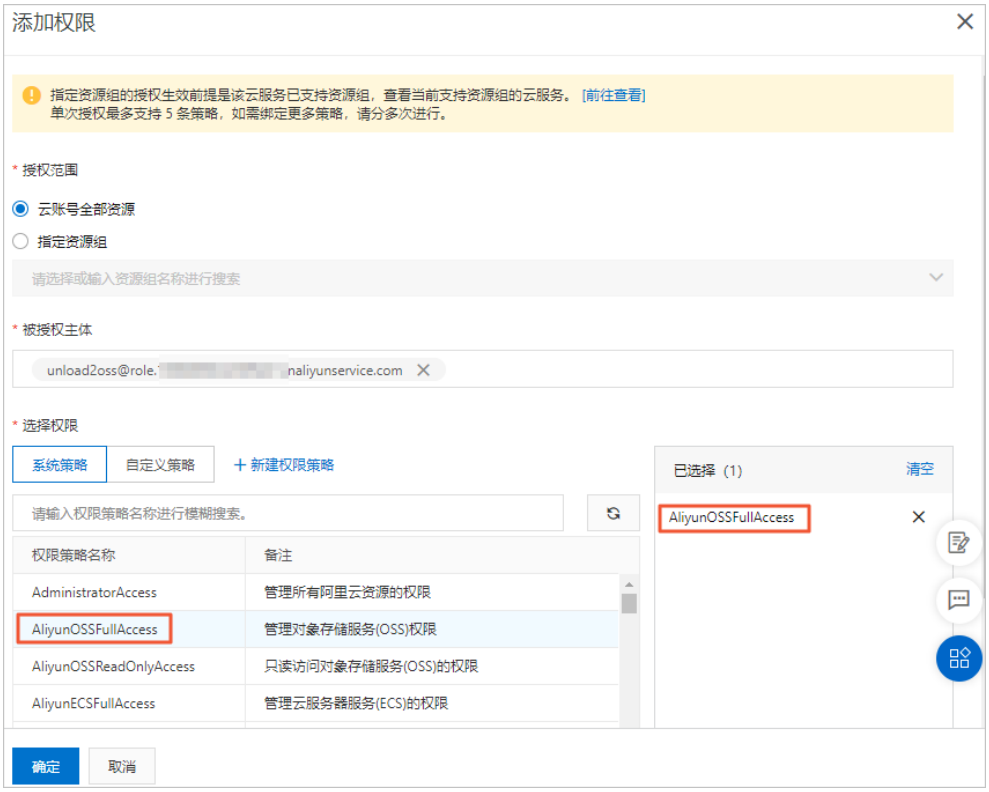
1. 登录RAM控制台，创建可信实体类型为**阿里云服务**的RAM角色。配置**角色类型**为**普通服务角色**、角色名称（例如`unload2oss`），选择受信服务为**大数据计算服务**。

   更多操作信息，请参见创建可信实体为阿里云服务的RAM角色。

2. 完成角色创建后，为新建角色添加系统策略AliyunOSSFullAccess。

   更多操作信息，请参见为RAM角色授权。



完成授权后，您需要根据导出数据的格式类型，选择对应的导出方式：

- 通过内置Extractor导出（StorageHandler）
- 导出其他开源格式数据

通过内置Extractor导出（StorageHandler）

- 命令格式

```
unload from {<select_statement>|<table_name> [partition (<pt_spec>)]}
into
location <external_location>
[stored by <StorageHandler>]
[with serdeproperties ('<property_name>'='<property_value>',...)];
```

- 参数说明
  - select_statement：`select`查询子句，从源表（分区表或非分区表）中查询需要插入目标OSS路径的数据。更多`select`信息，请参见SELECT语法。
  - table_name、pt_spec：使用表名称或表名称加分区名称的方式指定需要导出的数据。该导出方式不产生查询语句，不会产生费用。pt_spec格式为`(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)`。
  - external_location：必填。指定导出数据存储的目标OSS路径，格式为`'oss://<oss_endpoint>/<object>'`。更多OSS路径信息，请参见OSS访问域名使用规则。
  - StorageHandler：必填。指定内置的StorageHandler名称。固定取值为`com.aliyun.odps.CsvStorageHandler`或`com.aliyun.odps.TsvStorageHandler`，是内置的处理CSV、TSV格式文件的StorageHandler，定义了如何读或写CSV、TSV文件。相关逻辑已经由系统实现，您只需要指定该参数。此方法导出的文件默认添加`.csv`、`.tsv`后缀名。使用方法和MaxCompute外部表一致，请参见创建OSS外部表。
  - <property_name>'='<property_value>'：可选。property_name为属性名称，property_value为属性值。支持的属性和MaxCompute外部表一致。更多属性信息，请参见创建OSS外部表。
- 使用示例

  假设将MaxCompute项目中表sale_detail的数据导出至OSS。sale_detail的数据如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
+------------+-------------+-------------+------------+------------+
```

i. 登录OSS管理控制台，创建OSS Bucket目录 `mc-unload/data_location/` ，区域为 `oss-cn-hangzhou` ，并组织OSS路径。更多创建OSS Bucket信息，请参见创建存储空间。



根据Bucket、区域、Endpoint信息组织OSS路径如下：

```
oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-unload/data_location
```

ii. 登录MaxCompute客户端，执行UNLOAD命令，将sale_detail表的数据导出至OSS。命令示例如下：

- 示例1：将sale_detail表中的数据导出为CSV格式并压缩为GZIP。命令示例如下。

```
--控制导出文件个数：设置单个Worker读取MaxCompute表数据的大小，单位为MB。由于MaxCompute表有压缩，导出到OSS的数据一般会膨胀4倍左右。
set odps.stage.mapper.split.size=256;
--导出数据。
unload from
(select * from sale_detail)
into
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-unload/data_location'
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties ('odps.properties.rolearn'='acs:ram::139699392458****:role/unload2oss', 'odps.text.option.gzip.outp
ut.enabled'='true');
--等效于如下语句。
set odps.stage.mapper.split.size=256;
unload from sale_detail
into
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-unload/data_location'
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties ('odps.properties.rolearn'='acs:ram::139699392458****:role/unload2oss', 'odps.text.option.gzip.outp
ut.enabled'='true');
```

- 示例2：将sale_detail表中分区为sale_date='2013'，region='china'的数据导出为TSV格式并压缩为GZIP。

```
--控制导出文件个数：设置单个Worker读取MaxCompute表数据的大小，单位为MB。由于MaxCompute表有压缩，导出到OSS的数据一般会膨胀4倍左右。
set odps.stage.mapper.split.size=256;
--导出数据。
unload from sale_detail partition (sale_date='2013',region='china')
into
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-unload/data_location'
stored by 'com.aliyun.odps.TsvStorageHandler'
with serdeproperties ('odps.properties.rolearn'='acs:ram::139699392458****:role/unload2oss', 'odps.text.option.gzip.outp
ut.enabled'='true');
```

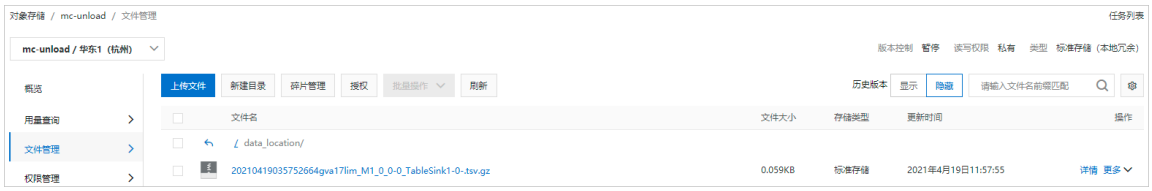
`'odps.text.option.gzip.output.enabled'='true'` 用于指定导出文件为GZIP压缩格式，当前仅支持GZIP压缩格式。

iii. 登录OSS管理控制台，查看目标OSS路径的导入结果。

- 示例1结果如下。



- 示例2结果如下。



导出其他开源格式数据

- 命令格式

```
unload from {<select_statement>|<table_name> [partition (<pt_spec>)]}
into
location <external_location>
[row format serde '<serde_class>'
  [with serdeproperties ('<property_name>'='<property_value>',...)]
]
storeds as <file_format>
[properties('<tbproperty_name>'='<tbproperty_value>')];
```

- 参数说明
  - select_statement：`select` 查询子句，从源表（分区表或非分区表）中查询需要插入目标OSS路径的数据。更多 `select` 信息，请参见SELECT语法。
  - table_name、pt_spec：使用表名称或表名称加分区名称的方式指定需要导出的数据。该导出方式不产生查询语句，不会产生费用。pt_spec格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。
  - external_location：必填。指定导出数据存储的目标OSS路径，格式为 `'oss://<oss_endpoint>/<object>'` 。更多OSS路径信息，请参见OSS访问域名使用规则。
  - serde_class：可选。使用方法和MaxCompute外部表一致，请参见创建OSS外部表。
  - '<property_name>'='<property_value>'：可选。property_name为属性名称，property_value为属性值。支持的属性和MaxCompute外部表一致。更多属性信息，请参见创建OSS外部表。
  - file_format：必填。指定导出数据文件格式。例如ORC、PARQUET、RCFILE、SEQUENCEFILE和TEXTFILE。使用方法和MaxCompute外部表一致，请参见创建OSS外部表。
  - '<tbproperty_name>'='<tbproperty_value>'：可选。tbproperty_name为外部表扩展信息属性名称，tbproperty_value为外部表扩展信息属性值。例如开源数据支持导出SNAPPY或LZO压缩格式，设置压缩属性为 `'mcfed.parquet.compression'='SNAPPY'` 或 `'mcfed.parquet.compression'='LZO'` 。

- 使用示例

  假设将MaxCompute项目中表sale_detail的数据导出至OSS。sale_detail的数据如下：

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
+------------+-------------+-------------+------------+------------+
```

  i. 登录OSS管理控制台，创建OSS Bucket目录 `mc-unload/data_location/` ，区域为 `oss-cn-hangzhou` ，并组织OSS路径。更多创建OSS Bucket信息，请参见创建存储空间。



  根据Bucket、区域、Endpoint信息组织OSS路径如下：

```
oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-unload/data_location
```

  ii. 登录MaxCompute客户端，执行UNLOAD命令，将sale_detail表的数据导出至OSS。命令示例如下：

- 示例1：将sale_detail表中的数据导出为PARQUET格式并压缩为SNAPPY。命令示例如下。

```
--控制导出文件个数：设置单个Worker读取MaxCompute表数据的大小，单位为MB。由于MaxCompute表有压缩，导出到OSS的数据一般会膨胀4倍左右。
set odps.stage.mapper.split.size=256;
--导出数据。
unload from
(select * from sale_detail)
into
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-unload/data_location'
row format serde 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
with serdeproperties ('odps.properties.rolearn'='acs:ram::139699392458****:role/unload2oss')
stored as parquet
properties('mcfed.parquet.compression'='SNAPPY');
```

- 示例2：将sale_detail表中分区为sale_date='2013'，region='china'的数据导出为PARQUET格式并压缩为SNAPPY。命令示例如下。

```
--控制导出文件个数：设置单个Worker读取MaxCompute表数据的大小，单位为MB。由于MaxCompute表有压缩，导出到OSS的数据一般会膨胀4倍左右。
set odps.stage.mapper.split.size=256;
--导出数据。
unload from sale_detail partition (sale_date='2013',region='china')
into
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/mc-unload/data_location'
row format serde 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
with serdeproperties ('odps.properties.rolearn'='acs:ram::139699392458****:role/unload2oss')
stored as parquet
properties('mcfed.parquet.compression'='SNAPPY');
```

iii. 登录OSS管理控制台，查看目标OSS路径的导入结果。

- 示例1结果如下。



- 示例2结果如下。



> **说明** 以SNAPPY或LZO压缩格式导出数据时，导出文件不支持显示.snappy或.lzo后缀名。

Unload函数导出设置前后缀

使用 `unload` 命令将MaxCompute的表导出为文件时，有的业务场景需要指定文件的前缀与后缀，按照下列的操作可以自定义文件的前缀，以及默认生成对应文件格式的后缀。

- 语法说明。
  - 内置解析器导出csv、tsv等格式文件。

```
--内置解析器，导出csv，tsv等格式
unload from {<select_statement>|<table_name> [partition (<pt_spec>)]}
into
location <external_location>
[stored by <StorageHandler>]
[with serdeproperties ('<property_name>'='<property_value>',...)];
```

  - 设置前缀的property_name为：odps.external.data.prefix，值可以自定义，长度不超过10个字符。
  - 设置后缀的property_name为：odps.external.data.enable.extension，值为 `true` 即后缀显示文件格式。
  - 其他参数请参见通过内置Extractor导出（StorageHandler）。

- 导出orc、parquet等开源格式文件。

```
unload from {<select_statement>|<table_name> [partition (<pt_spec>)]}
into
location <external_location>
[row format serde '<serde_class>'
  [with serdeproperties ('<property_name>'='<property_value>',...)]
]
storeds as <file_format>
[properties('<tbproperty_name>'='<tbproperty_value>')];
```

  - 设置前缀的tbproperty_name为：odps.external.data.prefix，值可以自定义，长度不超过10个字符。
  - 设置后缀的tbproperty_name为：odps.external.data.enable.extension，值为 `true` 即后缀显示文件格式。
  - 其他参数请参见导出其他开源格式数据。

- 后缀参考。

文件格式	SerDe	后缀
SEQUENCEFILE	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	.sequencefile
TEXTFILE	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	.txt
RCFILE	org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe	.rcfile
ORC	org.apache.hadoop.hive.ql.io.orc.OrcSerde	.orc
PARQUET	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe	.parquet
AVRO	org.apache.hadoop.hive.serde2.avro.AvroSerDe	.avro
JSON	org.apache.hive.hcatalog.data.JsonSerDe	.json
CSV	org.apache.hadoop.hive.serde2.OpenCSVSerde	.csv

- 使用示例。
  - 导出text格式文件，并添加 `mf_` 前缀和后缀。

```
set odps.sql.hive.compatible=true;
set odps.sql.split.hive.bridge=true;
unload from (select col_tinyint,col_binary from mf_fun_datatype limit 1)
into
location 'oss://oss-cn-beijing-internal.aliyuncs.com/mfosscostfee/demo6/'
stored as textfile
properties ('odps.external.data.prefix'='mf_', 'odps.external.data.enable.extension'='true');
```

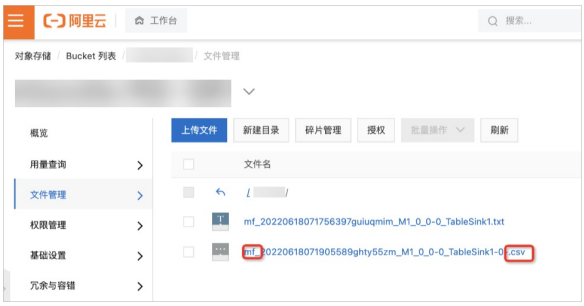
在指定导出数据存储的目标OSS路径查看导出结果。



- 导出csv格式文件，并添加 `mf_` 前缀和后缀。

```
set odps.sql.hive.compatible=true;
set odps.sql.split.hive.bridge=true;
unload from (select col_tinyint,col_binary from mf_fun_datatype limit 2)
into
location 'oss://oss-cn-beijing-internal.aliyuncs.com/mfosscostfee/demo6/'
stored by 'com.aliyun.odps.CsvStorageHandler'
properties ('odps.external.data.prefix'='mf_', 'odps.external.data.enable.extension'='true');
```

在指定导出数据存储的目标OSS路径查看导出结果。



3.7.3. EXPLAIN

在实际开发过程中，通常需要分析查询语句或表结构来分析性能瓶颈，MaxCompute SQL为您提供 `explain` 语句实现此功能。本文为您介绍 `explain` 的功能、命令格式及使用示例。

功能介绍

`explain` 语句可以显示MaxCompute SQL对应的DML语句执行计划（执行SQL语义的程序）的结构，帮助您了解SQL语句的处理过程，为优化SQL语句提供帮助。一个查询语句作业会对应多个Job，一个Job对应多个Task。

> **说明** 如果查询语句足够复杂，`explain` 的结果较多，则会触发API的限制，无法得到完整的EXPLAIN结果。此时您可以拆分查询语句，对各部分分别执行 `explain` 语句，以了解Job的结构。

命令格式

```
explain <dml query>;
```

dml query：必填。 `select` 语句，更多信息请参见SELECT语法。

返回说明

`explain` 的执行结果包含如下信息：

- Job间的依赖关系

  例如 `job0 is root job` 。如果查询只需要一个Job（ `job0` ），只会显示一行信息。

- Task间的依赖关系

```
In Job job0:
root Tasks: M1_Stg1, M2_Stg1
J3_1_2_Stg1 depends on: M1_Stg1, M2_Stg1
```

  `job0` 包含三个Task， `M1_Stg1` 、 `M2_Stg1` 和 `J3_1_2_Stg1` 。系统会先执行 `M1_Stg1` 和 `M2_Stg1` 两个Task，执行完成后，再执行 `J3_1_2_Stg1` 。

  Task的命名规则如下：

  - 在MaxCompute中，共有四种Task类型：MapTask、ReduceTask、JoinTask和LocalWork。Task名称的第一个字母表示了当前Task的类型，例如 `M2Stg1` 就是一个MapTask。
  - 紧跟着第一个字母后的数字，代表了当前Task的ID。这个ID在当前查询对应的所有Task中是唯一的。
  - 用下划线（_）分隔的数字代表当前Task的直接依赖，例如 `J3_1_2_Stg1` 表示当前Task ID为3，依赖ID为1（M1_Stg1）和ID为2（M2_Stg1）的两个Task。

- Task中所有Operator的依赖结构。

  Operator串描述了一个Task的执行语义。结构示例如下：

```
In Task M2:
    Data source: mf_mc_bj.sale_detail_jt/sale_date=2013/region=china  # "Data source"描述了当前Task的输入内容。
    TS: mf_mc_bj.sale_detail_jt/sale_date=2013/region=china           # TableScanOperator
        FIL: ISNOTNULL(customer_id)                                   # FilterOperator
            RS: order: +                                              # ReduceSinkOperator
                nullDirection: *
                optimizeOrderBy: False
                valueDestLimit: 0
                dist: HASH
                keys:
                      customer_id
                values:
                      customer_id (string)
                      total_price (double)
                partitions:
                      customer_id
In Task J3_1_2:
    JOIN:                                                             # JoinOperator
         StreamLineRead1 INNERJOIN StreamLineRead2
         keys:
             0:customer_id
             1:customer_id
        AGGREGATE: group by:customer_id                               # GroupByOperator
         UDAF: SUM(total_price) (__agg_0_sum)[Complete],SUM(total_price) (__agg_1_sum)[Complete]
            RS: order: +
                nullDirection: *
                optimizeOrderBy: True
                valueDestLimit: 10
                dist: HASH
                keys:
                      customer_id
                values:
                      customer_id (string)
                      __agg_0 (double)
                      __agg_1 (double)
                partitions:
In Task R4_3:
    SEL: customer_id,__agg_0,__agg_1                                  # SelectOperator
        LIM:limit 10                                                  # LimitOperator
            FS: output: Screen                                        # FileSinkOperator
                schema:
                  customer_id (string) AS ashop
                  __agg_0 (double) AS ap
                  __agg_1 (double) AS bp
```

各Operator的含义如下：

- TableScanOperator（TS）：描述查询语句中的 `from` 语句块的逻辑。 `explain` 结果中会显示输入表的名称（Alias）。
- SelectOperator（SEL）：描述查询语句中的 `select` 语句块的逻辑。 `explain` 结果中会显示向下一个Operator传递的列，多个列由逗号分隔。
  - 如果是列的引用，则显示为 `<alias>.<column_name>` 。
  - 如果是表达式的结果，则显示为函数形式，例如 `func1(arg1_1, arg1_2, func2(arg2_1, arg2_2))` 。
  - 如果是常量，则直接显示常量值。
- FilterOperator（FIL）：描述查询语句中的 `where` 语句块的逻辑。 `explain` 结果中会显示一个 `where` 条件表达式，形式类似SelectOperator的显示规则。
- JoinOperator（JOIN）：描述查询语句中的 `join` 语句块的逻辑。 `explain` 结果中会显示哪些表以哪种方式Join在一起。
- GroupByOperator（例如AGGREGATE）：描述聚合操作的逻辑。如果查询中使用了聚合函数，就会出现该结构， `explain` 结果中会显示聚合函数的内容。
- ReduceSinkOperator（RS）：描述Task间数据分发操作的逻辑。如果当前Task的结果会传递给另一个Task，则必然需要在当前Task的最后，使用ReduceSinkOperator执行数据分发操作。 `explain` 的结果中会显示输出结果的排序方式、分发的Key、Value以及用来求Hash值的列。
- FileSinkOperator（FS）：描述最终数据的存储操作。如果查询中有 `insert` 语句块， `explain` 结果中会显示目标表名称。
- LimitOperator（LIM）：描述查询语句中的 `limit` 语句块的逻辑。 `explain` 结果中会显示 `limit` 数。
- MapjoinOperator（HASHJOIN）：类似JoinOperator，描述大表的 `join` 操作。

示例数据

为便于理解，本文为您提供源数据，基于源数据提供相关示例。创建表sale_detail和sale_detail_jt，并添加数据，命令示例如下：

```
--创建分区表sale_detail和sale_detail_jt。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
create table if not exists sale_detail_jt
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china') partition (sale_date='2014', region='shanghai');
alter table sale_detail_jt add partition (sale_date='2013', region='china');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100.3)
;
insert into sale_detail partition (sale_date='2014', region='shanghai') values ('null','c5',null),('s6','c6',100.4),('s7','c7',10
0.5);
insert into sale_detail_jt partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s5','c2',100
.2);
--创建做关联的表。
create table shop as select shop_name, customer_id, total_price from sale_detail;
```

使用示例

基于示例数据，执行命令如下。

```
--查询语句。
select a.customer_id as ashop, sum(a.total_price) as ap,count(b.total_price) as bp
from (select * from sale_detail_jt where sale_date='2013' and region='china') a
inner join (select * from sale_detail where sale_date='2013' and region='china') b
on a.customer_id=b.customer_id
group by a.customer_id
order by a.customer_id
limit 10;
--获取查询语句语义。
explain
select a.customer_id as ashop, sum(a.total_price) as ap,count(b.total_price) as bp
from (select * from sale_detail_jt where sale_date='2013' and region='china') a
inner join (select * from sale_detail where sale_date='2013' and region='china') b
on a.customer_id=b.customer_id
group by a.customer_id
order by a.customer_id
limit 10;
```

返回结果如下。

```
job0 is root job
In Job job0:
root Tasks: M1
In Task M1_U0:
    TS: doc_test_dev.sale_detail_jt/sale_date=2013/region=china
        FIL: ISNOTNULL(customer_id)
            HASHJOIN:
                    Filter1 INNERJOIN Filter2
                    keys:
                        0:customer_id
                        1:customer_id
                    non-equals:
                        0:
                        1:
                    bigTable: Filter1
                LocalSortBy: order: +
                             nullDirection: *
                             keys:customer_id
                   AGGREGATE: group by:customer_id
                    UDAF: SUM(total_price) (__agg_0_sum)[Complete],COUNT(total_price) (__agg_1_count)[Complete]
                       LIM:limit 10
                           FS: output: Screen
                               schema:
                                 customer_id (string) AS ashop
                                 __agg_0 (double) AS ap
                                 __agg_1 (bigint) AS bp
In Task M1_U1:
    TS: doc_test_dev.sale_detail/sale_date=2013/region=china
        FIL: ISNOTNULL(customer_id)
            HASHJOIN:
                    Filter1 INNERJOIN Filter2
                    keys:
                        0:customer_id
                        1:customer_id
                    non-equals:
                        0:
                        1:
                    bigTable: Filter1
                LocalSortBy: order: +
                             nullDirection: *
                             keys:customer_id
                   AGGREGATE: group by:customer_id
                    UDAF: SUM(total_price) (__agg_0_sum)[Complete],COUNT(total_price) (__agg_1_count)[Complete]
                       LIM:limit 10
                           FS: output: Screen
                               schema:
                                 customer_id (string) AS ashop
                                 __agg_0 (double) AS ap
                                 __agg_1 (bigint) AS bp
```

基于示例数据，执行命令如下。

```
--查询语句。
select /*+ mapjoin(a) */
       a.customer_id as ashop, sum(a.total_price) as ap,count(b.total_price) as bp
 from (select * from sale_detail_jt
where sale_date='2013' and region='china') a
inner join (select * from sale_detail where sale_date='2013' and region='china') b
on a.total_price<b.total_price
group by a.customer_id
order by a.customer_id
limit 10;
--获取查询语句语义。
explain
select /*+ mapjoin(a) */
       a.customer_id as ashop, sum(a.total_price) as ap,count(b.total_price) as bp
 from (select * from sale_detail_jt
where sale_date='2013' and region='china') a
inner join (select * from sale_detail where sale_date='2013' and region='china') b
on a.total_price<b.total_price
group by a.customer_id
order by a.customer_id
limit 10;
```

返回结果如下。

```
job0 is root job
In Job job0:
root Tasks: M1
In Task M1_U0:
    TS: doc_test_dev.sale_detail_jt/sale_date=2013/region=china
        HASHJOIN:
                 TableScan1 INNERJOIN TableScan2
                 keys:
                     0:
                     1:
                 non-equals:
                     0:
                     1:
                 bigTable: TableScan2
            FIL: LT(total_price,total_price)
                LocalSortBy: order: +
                             nullDirection: *
                             keys:customer_id
                   AGGREGATE: group by:customer_id
                    UDAF: SUM(total_price) (__agg_0_sum)[Complete],COUNT(total_price) (__agg_1_count)[Complete]
                      LIM:limit 10
                          FS: output: Screen
                              schema:
                                customer_id (string) AS ashop
                                __agg_0 (double) AS ap
                                __agg_1 (bigint) AS bp
In Task M1_U1:
    TS: doc_test_dev.sale_detail/sale_date=2013/region=china
        HASHJOIN:
                 TableScan1 INNERJOIN TableScan2
                 keys:
                     0:
                     1:
                 non-equals:
                     0:
                     1:
                 bigTable: TableScan2
            FIL: LT(total_price,total_price)
                LocalSortBy: order: +
                             nullDirection: *
                             keys:customer_id
                   AGGREGATE: group by:customer_id
                    UDAF: SUM(total_price) (__agg_0_sum)[Complete],COUNT(total_price) (__agg_1_count)[Complete]
                      LIM:limit 10
                          FS: output: Screen
                              schema:
                                customer_id (string) AS ashop
                                __agg_0 (double) AS ap
                                __agg_1 (bigint) AS bp
```

3.7.4. COMMON TABLE EXPRESSION（CTE）

CTE表示公用表表达式，是一个临时命名结果集，用于简化SQL。MaxCompute支持标准SQL的CTE，可以更好地提高SQL语句的可读性与执行效率。本文为您介绍CTE的功能、命令格式及使用示例。

功能介绍

CTE可以被认为是在单个DML语句的执行范围内定义的临时结果集。CTE类似于派生表，它不作为对象存储，并且仅在查询期间持续。开发过程中结合CTE，可以提高SQL语句可读性，便于轻松维护复杂查询。

命令格式

```
with
     <cte_name> as
    (
        <cte_query>
    )
    [,<cte_name2>  as
     (
     <cte_query2>
     )
    ,……]
```

- cte_name：必填。CTE的名称，不能与当前 `with` 子句中的其他CTE的名称相同。查询中任何使用到cte_name标识符的地方，均指CTE。
- cte_query：必填。一个 `select` 语句。 `select` 的结果集用于填充CTE。

使用示例

假设现有如下代码：

```
insert overwrite table srcp partition (p='abc')
select * from (
    select a.key, b.value
    from (
        select * from src where key is not null    ) a
    join (
        select * from src2 where value > 0    ) b
    on a.key = b.key
) c
union all
select * from (
    select a.key, b.value
    from (
        select * from src where key is not null    ) a
    left outer join (
        select * from src3 where value > 0    ) b
    on a.key = b.key and b.key is not null
)d;
```

顶层的 `union` 两侧各为一个 `join` ， `join` 的左表是相同的查询语句。通过写子查询的方式，只能重复这段代码。

使用CTE的方式重写以上语句，命令示例如下：

```
with
  a as (select * from src where key is not null),
  b as (select  * from src2 where value > 0),
  c as (select * from src3 where value > 0),
  d as (select a.key, b.value from a join b on a.key=b.key),
  e as (select a.key,c.value from a left outer join c on a.key=c.key and c.key is not null)
insert overwrite table srcp partition (p='abc')
select * from d union all select * from e;
```

重写后， `a` 对应的子查询只需写一次，便可在后面进行重用。您可以在CTE的 `with` 子句中指定多个子查询，像使用变量一样在整个语句中反复重用。除重用外，不必反复嵌套。

3.7.5. CLONE TABLE

当您需要将一张表中的数据复制到另一个表中时，可以使用MaxCompute的 `clone table` 功能，可以提高数据迁移效率。本文为您介绍 `clone table` 的功能、使用限制、命令格式及使用示例。

功能介绍

支持高效地将源表中的数据复制到目标表中。将数据复制到目标表之后，建议您执行数据验证以确保复制后数据的准确性。例如，执行 `select` 命令查看表的数据、执行 `desc` 命令查看表大小。

使用限制

- 目标表与源表的Schema需要兼容。
- 支持分区表和非分区表，支持对聚簇表使用 `clone table` 命令复制表数据。
- 目标表已存在时，一次性复制分区的数量上限为10000个。
- 目标表不存在时，无分区数量限制，满足原子性。
- 对同一个非分区表或分区表的同一个分区，执行 `clone table` 命令的次数不能超过6次。
- 不支持在跨区域的MaxCompute项目之间使用 `clone table` 命令复制表数据。
- 不支持对外部表使用 `clone table` 命令复制表数据。

命令格式

```
clone table <[<src_project_name>.]<src_table_name>> [partition(<pt_spec>), ...]
 to <[<dest_project_name>.]<dest_table_name>> [if exists [overwrite | ignore]] ;
```

- src_project_name：可选。源表所属MaxCompute项目名称。不指定时，默认为当前项目。当源表与目标表不属于同一个MaxCompute项目时，需要携带此参数。
- src_table_name：必填。源表名称。

- pt_spec：可选。源表的分区信息。格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。partition_col是分区字段，partition_col_value是分区值。
- dest_project_name：可选。目标表所属MaxCompute项目名称。不指定时，默认为当前项目。当目标表与源表不属于同一个MaxCompute项目时，需要携带此参数。
- desc_table_name：必填。目标表名称。
  - 当目标表不存在时， `clone table` 命令会创建目标表，创建目标表使用的是 `create table like` 语义。更多 `create table like` 信息，请参见创建表。
  - 当目标表已存在并指定 `if exists overwrite` 时， `clone table` 命令会覆盖目标表或对应分区的数据。
  - 当目标表已存在并指定 `if exists ignore` 时， `clone table` 命令会跳过已存在分区，不覆盖目标表已有分区的数据。

示例数据

为便于理解，本文为您提供源数据，基于源数据提供相关示例。创建分区表sale_detail和非分区表sale_detail_np，并添加数据，命令示例如下：

- 分区表sale_detail

```
--创建一张分区表sale_detail。
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string, region string);
--向源表增加分区。
alter table sale_detail add partition (sale_date='2013', region='china') partition (sale_date='2014', region='shanghai');
--向源表追加数据。
insert into sale_detail partition (sale_date='2013', region='china') values ('s1','c1',100.1),('s2','c2',100.2),('s3','c3',100
.3);
insert into sale_detail partition (sale_date='2014', region='shanghai') values ('null','c5',null),('s6','c6',100.4),('s7','c7'
,100.5);
```

查询分区表sale_detail中的数据，命令示例如下：

```
select * from sale_detail;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
+------------+-------------+-------------+------------+------------+
```

- 非分区表sale_detail_np

```
--创建一张非分区表sale_detail_np。
create table if not exists sale_detail_np
(
shop_name     string,
customer_id   string,
total_price   double
);
--向源表追加数据。
insert into sale_detail_np values ('s4','c4',100.4);
```

查询非分区表sale_detail_np中的数据，命令示例如下：

```
select * from sale_detail_np;
--返回结果。
+------------+-------------+-------------+
| shop_name  | customer_id | total_price |
+------------+-------------+-------------+
| s4         | c4          | 100.4       |
+------------+-------------+-------------+
```

使用示例

基于示例数据， `clone table` 命令的使用示例如下：

- 示例1：全量复制非分区表sale_detail_np的数据至目标表sale_detail_np_clone。命令示例如下：

```
--复制表数据。
clone table sale_detail_np to sale_detail_np_clone;
--查看复制后目标表sale_detail_np_clone的信息，验证数据准确性。
select * from sale_detail_np_clone;
--返回结果。
+------------+-------------+-------------+
| shop_name  | customer_id | total_price |
+------------+-------------+-------------+
| s4         | c4          | 100.4       |
+------------+-------------+-------------+
```

- 示例2：复制分区表sale_detail指定分区的数据至目标表sale_detail_clone。命令示例如下：

```
--复制表数据。
clone table sale_detail partition (sale_date='2013', region='china') to sale_detail_clone if exists overwrite;
--查看复制后目标表sale_detail_clone的信息，验证数据准确性。
select * from sale_detail_clone;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
+------------+-------------+-------------+------------+------------+
```

- 示例3：全量复制分区表sale_detail的数据至目标表sale_detail_clone（示例2已生成的表）并跳过目标表中已存在的分区。命令示例如下：

```
--复制表数据。
clone table sale_detail to sale_detail_clone if exists ignore;
--查看复制后目标表sale_detail_clone的信息，验证数据准确性。
select * from sale_detail_clone;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
+------------+-------------+-------------+------------+------------+
```

- 示例4：全量复制分区表sale_detail的数据至目标表sale_detail_clone1。命令示例如下：

```
--复制表数据。
clone table sale_detail to sale_detail_clone1;
--查看复制后目标表sale_detail_clone1的信息，验证数据准确性。
select * from sale_detail_clone1;
--返回结果。
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
+------------+-------------+-------------+------------+------------+
```

3.7.6. 参数化视图

参数化视图支持传入任意表或其它变量，定制视图的行为。本文为您介绍MaxCompute SQL引擎支持的参数化视图功能。

功能介绍

MaxCompute传统的视图（VIEW）中，底层封装一段逻辑复杂的SQL脚本，调用者可以像读普通表一样调用视图，无需关心底层的实现。传统的视图实现了一定程度的封装与重用，因此被广泛地使用。但是传统的视图并不接受调用者传递的任何参数（例如调用者无法对视图读取的底层表进行数据过滤或传递其他参数），导致代码重用能力低下。MaxCompute当前的SQL引擎支持带参数的视图，支持传入任意表或者其它变量，定制视图的行为。

命令格式

```
create [or replace] [if not exists] <view_name>( <variable_name> <variable_type> [, <variable_name> <variable_type> ...])
[returns <return_variable> table (<col_name> <col_type> comment <col_comment> [,<col_name> <col_type> comment <col_comment>])]
[comment <view_comment>]
as
{<select_statement> | begin <statements> end}
```

- view_name：必填。视图名称。
- variable_name：必填。视图变量名称。
- variable_type：必填。视图变量参数类型。
- return_variable：可选。视图返回的变量名称。
- col_name：可选。视图返回列的名称。
- col_type：可选。视图返回列的类型。
- col_comment：可选。视图返回列的注释。
- view_comment：可选。视图的注释。
- select_statement：条件必选。`select`子句。
- statements：条件必选。视图脚本。

定义参数化视图

创建带参数的视图，语法如下。

```
-- view with parameters
-- param @a -a table parameter
-- param @b -a string parameter
-- returns a table with schema (key string, value string)
create view if not exists pv1(@a table (k string,v bigint), @b string)
as
select srcp.key,srcp.value from srcp join @a on srcp.key=a.k and srcp.p=@b;
```

语法说明：

- 因为定义了参数，所以定义参数化视图需要通过**脚本模式**操作。
- 创建的视图pv1有两个参数，即表参数和STRING参数，参数可以是任意的表或基本数据类型。
- 支持使用子查询作为参数的值，例如`select * from view_name( (select 1 from src where a > 0), 1);`。
- 定义视图时，您可以为参数指定ANY类型，表示任意数据类型。例如`create view paramed_view (@a any) as select * from src where case when @a is null then key1 else key2 end = key3;`，定义了视图的第一个参数可以接受任意类型。

  但是ANY类型不能参与类似`+`、`and`需要明确类型才能执行的运算。ANY类型通常用在Table参数中作为PassThrough列，示例如下。

  ```
  create view paramed_view (@a table(name string, id any, age bigint)) as select * from @a where name = 'foo' and age < 25;
  --调用示例。
  select * from param_view((select name, id, age from students));
  ```

  > **说明** 使用`create view`创建视图后，可以执行`desc`命令获取视图的描述。此描述中包含视图的返回类型信息。
  >
  > 视图的返回类型是在调用时重新推算的，它可能与创建视图时不一致，例如ANY类型。

- 定义视图时，Table的参数还支持使用星号（*）表示任意多个列。星号（*）可以指定数据类型，也可以使用ANY类型，示例如下。

  ```
  create view paramed_view (@a table(key string, * any), @b table(key string, * string)) as select a.* from @a join @b ON a.key
  = b.key;
  -- 调用示例。
  select name, address from param_view((select school, name, age, address from student), school) where age < 20;
  ```

  示例中的视图接受两个表值参数。第一个表值参数第一列是STRING类型，后面可以是任意多个任意类型的列；第二个表值参数的第一列是STRING类型，后面可以是任意多个STRING类型的列。注意事项如下：

  - 变长部分必须写在表值参数定义语句的最后位置，即在星号（*）的后面不允许再出现其它列。因此，一个表值参数中最多只有一个变长列列表。
  - 由于变长部分必须写在表值参数定义语句的最后位置，有时输入表的列不一定是按照这种顺序排列的，这时需要重排输入表的列，可以以子查询作为参数（参考上述示例），子查询外面必须加一层括号。
  - 因为表值参数中变长部分没有名字，所以在视图定义过程中无法获得对这部分数据的引用，也无法对这些数据做运算。
  - 虽然无法对变长部分做运算，但可以使用`select *`这种通配符将变长部分的列传递出去。
  - 表值参数的列与定义视图时指定的定长列部分不一定会完全一致。如果名字不一致，编译器会自动做重命名；如果类型不一致，编译器会做隐式转换（不能隐式转换时，会发生报错）。

调用参数化视图

调用已经定义的pv1视图的示例如下。

```
@a := select * from src where value >0;
--call view with table variable and scalar
@b := select * from pv1(@a,'20170101');
@another_day := '20170102';
--call view with table name and scalar variable
@c := select * from pv1(src2, @another_day);
@d := select * from @c union all select * from @b;
with
t as(select * from src3)
select * from @c
union all
select * from @d
union all
select * from pv1(t,@another_day);
```

说明 您可以使用不同的参数调用pv1：

- 表参数可以是物理表、VIEW、表变量或者CTE中的表别名。
- 普通参数可以是变量或常量。

参数化视图说明

- 参数化视图中，脚本中只能使用DML语句，不能使用 `insert` 或 `create table` 语句，也不能使用屏幕显示语句。
- 参数化视图不一定只有一个SQL语句，也可以像脚本一样，包含多个语句。

```
-- view with parameters
-- param @a -a table parameter
-- param @b -a string parameter
-- returns a table with schema (key string,value string)
create view if not exists pv2(@a table (k string,v bigint), @b string) as
begin
@srcp := select * from srcp where p=@b;
@pv2 := select srcp.key,srcp.value from @srcp join @a on srcp.key=a.k;
end;
```

说明 `begin` 到 `end` 之间的语句，就是这个视图的脚本。 `@pv2 :=...` 语句相当于其他语言中的 `return` 语句，用于向一个与视图同名的隐含表变量赋值。

- 在视图参数匹配时，实参和形参匹配的规则和普通的弱类型语言一样，如果传入的视图参数可以被隐式转换，则可与所定义的参数匹配。例如，BIGINT的值可以匹配DOUBLE类型的参数。对于表变量，如果表a的Schema可以被用于插入到表b中，则意味着表a可以用来匹配和表b的Schema相同的表类型参数。
- 您可以明确地声明返回类型，以提升代码的可读性。

```
create view if not exists pv3(@a table (k string,v bigint), @b string)
returns @ret table (x string comment 'This is the x',y string comment 'This is the y')
comment 'This is view pv3'
as
begin
    @srcp := select * from srcp where p=@b;
    @ret := select srcp.key,srcp.value from @srcp join @a on srcp.key=a.k;
end;
```

说明 `returns @ret table (x string,y string)` 定义了：

- 返回类型为 `table (x string,y string)` ，即返回给调用者的类型，可以在此处定制表的Schema。
- 返回参数为 `@ret` ，为它赋值的操作会在视图的脚本中进行，此处相当于定义了返回的参数名。

您可以将没有 `begin/end` 和返回变量的视图，看作是此形式的简化形式。

3.8. SQL常见问题

3.8.1. DDL操作常见问题

本文为您介绍对表、分区、列执行DDL操作过程中的常见问题。

问题类别	常见问题

问题类别	常见问题
表操作	• MaxCompute支持虚拟表吗？例如MySQL中的DUAL表？ • MaxCompute的表有无索引？ • 如何修改表的Hash Clustering属性？ • 如何将非分区表修改为分区表？ • 因误操作删除的表可以恢复吗？ • 如何查询某个用户创建的表？ • 如何查看指定的表是否存在？ • 如何获取项目下的所有表名称？ • 如何快速查看项目下哪些表是分区表？ • 如何查看MaxCompute表的最近访问时间？ • 如何查看表的数据量？ • 如何查看表的行数？ • 创建的表为什么在DataWorks上看不到？
分区操作	• 分区和分区列的区别是什么？ • 一张表的分区的数量是否越多越好？ • 如果源表没有分区字段，是否可以增加或更改分区？ • 如何查看指定的分区是否存在？ • 如何查看分区数量？
列操作	• 是否可以添加或删除列？ • 如何添加列？ • 如何设置自增长列？ • MaxCompute单表可以存放的最大列数是多少？ • 待创建表的列名与关键字相同，如何解决？ • 如何修改列的数据类型？ • 如何删除列？
生命周期操作	设置表的生命周期为3天，每个表的分区存储量很大，如何清理分区表旧数据？

MaxCompute支持虚拟表吗？例如MySQL中的DUAL表？

不支持虚拟表，您可以手动创建DUAL表。

MaxCompute的表有无索引？

没有索引，Hash Clustering可以提供类似数据库中Cluster index的效果，详情请参见表操作。

如何修改表的Hash Clustering属性？

- 增加表的Hash Clustering属性：`alter table table_name [clustered by (col_name [, col_name, ...]) [sorted by (col_name [asc | desc] [, col_name [asc | desc] ...])] into number_of_buckets buckets];` 。
- 去除表的Hash Clustering属性：`alter table table_name not clustered;` 。

如何将非分区表修改为分区表？

不支持更改非分区表为分区表，也不支持增加分区列。您可以重新创建一张分区表，详情请参见表操作。

因误操作删除的表可以恢复吗？

MaxCompute提供备份与恢复功能。但免费周期只有24小时，您可以自定义设置备份数据保留周期。如果误删除24小时内，可以通过备份与恢复命令恢复表数据。更多备份恢复信息，请参见备份与恢复。

如何查询某个用户创建的表？

只查询某个用户创建的表可以使用元数据视图TABLES，通过owner_name字段过滤。更多TABLES信息，请参见TABLES。

如何查看指定的表是否存在？

可以使用函数TABLE_EXISTS查看指定的表是否存在，更多函数信息，请参见TABLE_EXISTS。

如何获取项目下的所有表名称？

您可以通过MaxCompute客户端执行 `show tables;` 命令或使用MaxCompute的元数据服务。更多信息请参见表操作或Information Schema。

如何快速查看项目下哪些表是分区表？

您可以通过MaxCompute客户端执行如下命令查看项目下的分区表信息。

```
select table_name from information_schema.columns where is_partition_key = true group by table_name;
```

如何查看MaxCompute表的最近访问时间？

您可以在DataWorks数据地图中查询表，进入表详情页面获取表的最近访问时间。



如何查看表的数据量？

查看表的数据量包含查看数据条数和占用的物理空间大小：

- 您可以执行 `desc` 命令查看全量表的物理空间。执行SQL语句 `select count() as cnt from table_name;` 查看表的数据条数。



- 您可以执行 `desc` 命令和WHERE条件组合方式，查看分区表单个分区占用的物理空间大小。执行SQL语句 `select count() as cnt from table_name where ...;` 查看分区数据条数。



如何查看表的行数？

您可以通过MaxCompute客户端执行 `select count(*) from table_name;` 命令查看分区表或非分区表的行数。

创建的表为什么在DataWorks上看不到？

如果表不是通过DataWorks可视化创建方式创建的，而是通过SQL语句创建，DataWorks界面不会立即显示创建的表，会存在2~3个小时的延时。

分区和分区列的区别是什么？

MaxCompute中的表可以分区，分区表有分区列。您可以通过分区列创建分区。

例如分区 `ds=20150101` ，此处 `ds` 是一个分区列，而 `ds=20150101` 是一个分区。

一张表的分区的数量是否越多越好？

在MaxCompute中，一张表最多允许有60000个分区，同时每个分区的容量没有上限。但是分区数量过多，不便于统计和分析。

MaxCompute限制单个作业中最多不能超过一定数量的Instance，而作业中的Instance数量和输入的数据量以及分区数量是密切相关的，所以您需要根据业务情况，选择合适的分区策略。

如果源表没有分区字段，是否可以增加或更改分区？

MaxCompute不支持在源表上直接增加或修改分区字段，分区字段一旦创建就无法修改。您可以重新创建一张分区表，使用动态分区SQL将源表数据导入至新分区表，详情请参见插入或覆写动态分区数据（DYNAMIC PARTITION）。

如何查看指定的分区是否存在？

可以使用函数PARTITION_EXISTS查看指定的分区是否存在，更多函数信息，请参见PARTITION_EXISTS。

如何查看分区数量？

您可以通过Information Schema的PARTITIONS视图，获取到分区名，进而获取到分区数量。

是否可以添加或删除列？

在MaxCompute中，可以添加列，但不可以删除列。

如何添加列？

添加列的命令示例如下。如果表中已经存在一部分数据，则新添加列的值为NULL。

```
alter table table_name add columns (col_name1 type1, col_name2 type2…);
```

更多添加列语法信息，请参见添加列。

如何设置自增长列？

MaxCompute不支持自增长列功能，如果您有此需求，且数据量比较小，建议使用ROW_NUMBER实现。

MaxCompute单表可以存放的最大列数是多少？

MaxCompute单表可以存放的最大列数为1200列。如果您的列数超过限制，可以参考如下方式处理：

- 对数据进行降维，缩减到1200列以内。
- 修改数据的保存方式，例如设备证书、稀疏或稠密矩阵。

待创建表的列名与关键字相同，如何解决？

在对表、列或分区命名时如果使用关键字，需要给关键字加 `` 符号进行转义，否则会报错。

如何修改列的数据类型？

不支持修改列的数据类型，只能添加列。生产环境中的表不允许删除表字段、修改表字段和分区字段。如果必须修改，请删除表之后重新建表。您也可以创建外部表，删除并重建表后，可以重新加载数据。数据类型详情请参见数据类型版本说明。

如何删除列？

MaxCompute不支持删除表的列。如果您有删除列的需求，可以通过如下方法实现：

1. 创建一张新表，命令示例如下。

   ```
   create table new_table_name as select c1,c2,c3 from old_table_name;
   ```

2. 删除旧表，并重命名新表，命令示例如下。

   ```
   drop table old_table_name;
   alter table new_table_name rename to old_table_name;
   ```

设置表的生命周期为3天，每个表的分区存储量很大，如何清理分区表旧数据？

设置了生命周期的表超过设定时间没有修改，MaxCompute会自动回收。

您可以执行 `desc table_name partition(pt_spec);` 命令查看旧的分区修改时间是否在生命周期内修改过。通过 `desc tablename;` 命令查看生命周期时间，MaxComptue每天17:00点进行回收，DataWorks上的数据显示有延迟，一般会延迟一天。

3.8.2. DML操作常见问题

本文为您介绍对表执行DML操作过程中的常见问题。

问题类别	常见问题
插入或更新数据	• 执行INSERT操作过程中出现错误，会损坏原有数据吗？ • 执行INSERT INTO或INSERT OVERWRITE操作时，提示Table xxx has n columns，but query has m columns，如何解决？ • 执行INSERT INTO或INSERT OVERWRITE操作时，报错a single instance cannot output data to more than 10000 partitions，如何解决？ • 向MaxCompute表中插入动态分区时，报错invalid dynamic partition value，如何解决？ • 向MaxCompute表中插入FLOAT类型的数据报错，如何解决？ • 对相同数据执行INSERT SELECT操作和SELECT操作的结果为什么不一致？ • 目标表的字段类型为VARCHAR(10)，插入数据溢出时会报错吗？ • 在执行MaxCompute SQL过程中，报错Transaction timeout because cannot acquire exclusive lock，如何解决？ • 如何更新MaxCompute表或分区中的数据？

问题类别	常见问题
删除数据	• 如何删除MaxCompute表或分区中的数据？ • 如果表数据量较大，如何删除非分区表中的重复数据？

执行INSERT操作过程中出现错误，会损坏原有数据吗？

不会损坏原有数据。MaxCompute满足原子性，INSERT操作执行成功则更新数据，INSERT操作执行失败则回滚数据。

执行INSERT INTO或INSERT OVERWRITE操作时，提示 `Table xxx has n columns, but query has m columns` ，如何解决？

执行INSERT INTO或INSERT OVERWRITE操作插入数据时，需要保证SELECT得到的字段和目标表的字段匹配，匹配内容包括顺序、字段类型和总的字段数量。MaxCompute不支持插入表的指定字段，其他字段为NULL或者其他默认值时，您可以在SELECT时设置为NULL，例如 `select 'a', null, col_name from table_name;` 。

执行INSERT INTO或INSERT OVERWRITE操作时，报错a single instance cannot output data to more than 10000 partitions，如何解决？

- 问题现象

  在INSERT INTO或INSERT OVERWRITE操作时，返回报错如下。

  ```
  FAILED: ODPS-0123031:Partition exception - a single instance cannot output data to more than 10000 partitions
  ```

- 产生原因

  虽然单个MaxCompute表允许有6万个分区，但是单个作业涉及的输出表分区数量只允许有10000个。出现这个错误，通常是因为分区字段设置有误，例如根据ID字段分区造成分区过多。

- 解决措施

  一般作业输出动态分区数达到几千已经很大，超过10000可能存在业务逻辑或SQL语法问题。如无逻辑或语法问题，建议修改分区表的分区字段，或将业务逻辑拆分为多个作业，避免出现该错误。

向MaxCompute表中插入动态分区时，报错invalid dynamic partition value，如何解决？

- 问题现象

  执行插入动态分区操作时，返回报错如下。

  ```
  FAILED: ODPS-0123031:Partition exception - invalid dynamic partition value: province=上海
  ```

- 产生原因

  使用了非法的动态分区。动态分区是根据指定字段进行分区，不支持特殊字符和中文动态分区字段。

- 解决措施

  插入动态分区时，需要注意如下情况：

  - 在分布式环境下执行插入动态分区操作时，单个进程最多只能输出512个动态分区。
  - 任意动态分区SQL不允许生成超过2000个动态分区。
  - 动态生成的分区值不允许为NULL。
  - 如果目标表有多级分区，在执行INSERT操作时，允许指定部分分区为静态，但是静态分区必须是高级分区。

向MaxCompute表中插入FLOAT类型的数据报错，如何解决？

MaxCompute 2.0支持的基本数据类型请参见数据类型版本说明。其中：FLOAT数据类型没有常量定义，若要插入该类型数据，可以使用CAST函数转换数据类型。例如 `cast(5.1 as float)` 将字符串 `'5.1'` 转为FLOAT类型 `5.1` 。

MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP或BINARY）时，需要执行如下语句开启新数据类型开关：

- Session级别：如果使用新数据类型，您需要在SQL语句前加上 `set odps.sql.type.system.odps2=true;` ，并与SQL语句一起提交执行。
- Project级别：执行 `setproject odps.sql.type.system.odps2=true;` 打开Project级别的新数据类型。该命令需要项目所有者执行。

对相同数据执行INSERT SELECT操作和SELECT操作的结果为什么不一致？

- 问题现象

  对相同的STRING类型字段分别执行SQL语句，出现小数位不统一的现象。执行SELECT操作保留2位小数，执行INSERT SELECT操作，结果显示多个小数位。

- 产生原因

  对于INSERT SELECT操作，原始字段类型是STRING，在隐式转换为目标类型DECIMAL的过程中，先转换为DOUBLE类型，然后在DOUBLE类型数据的基础上执行ROUND操作。由于DOUBLE类型本身是不精确的，虽然执行了ROUND操作，但是依然可能显示多个小数位。

- 解决措施

建议使用显示转换方式，增加如下语句通过CAST显示转换为DECIMAL类型。

```
case when pcm.abc is null then 0
                  else round(cast(pcm.abc as decimal) ,2)
               end abc
```

目标表的字段类型为VARCHAR(10)，插入数据溢出时会报错吗？

对 `VARCHAR(10)` 数据类型的字段插入数据时，数据长度溢出时会截断并不报错。

在执行MaxCompute SQL过程中，报错Transaction timeout because cannot acquire exclusive lock，如何解决？

- 问题现象

  执行MaxCompute SQL的过程中，返回报错如下。

  ```
  Failed to run ddltask - Modify DDL meta encounter exception : ODPS-0121096:MetaStore transaction conflict - Reached maximum re
  try times because of OTSStorageTxnLockKeyFail(Inner exception: Transaction timeout because cannot acquire exclusive lock.)
  ```

- 产生原因

  MaxCompute对正在操作的表没有锁机制，该错误是由元数据产生竞争导致。您需要检查是否存在同时多次对表或表分区执行读写操作的情况。

- 解决措施

  在MaxCompute还没有锁机制的情况下，不要同时对一张表或表分区执行多次读写操作。

如何更新MaxCompute表或分区中的数据？

MaxCompute支持通过 `update` 操作，在行级别更新Transactional表中的数据。

如果表非Transactional表，您需要把源分区或源表中的数据导入到新分区或新表中，在导入过程中执行相应的更新逻辑操作。新分区或新表可以与源分区或源表相同，即就地更新。

如何删除MaxCompute表或分区中的数据？

MaxCompute支持通过 `delete` 操作，在行级别删除Transactional表中的数据。

如果表非Transactional表，您可以通过如下方法删除：

- 执行 `drop` 命令删除表，达到删除数据的目的。
- 如果是非分区表，您可以执行 `truncate table table_name;` 命令清空表数据或通过 `insert overwrite` 命令实现类似的功能。
  - 示例一：删除TableA表中Col=1的数据，命令示例如下。

    ```
    insert overwrite table TableA select a,b,c.... from TableA where Col <> 1;
    ```

  - 示例二：删除全部数据。

    ```
    insert overwrite table TableA select a,b,c.... from TableA where 1=2;
    ```

- 如果是分区表，您可以执行 `alter table table_name drop if exists partition(分区名='具体分区值')` 命令，删除对应的分区，即可删除分区对应的数据。

  例如，表testtable的分区列为ds，执行如下命令删除 `ds='20170520'` 的分区。

  ```
  alter table testtable drop if exists partition (ds='20170520');
  ```

- 使用INSERT和WHERE条件，将需要的数据导入到至另一个新分区或新表中。INSERT支持源表和目标表相同。

  ```
  insert overwrite table sale_detail select * from sale_detail where name='mengyonghui';
  ```

如果表数据量较大，如何删除非分区表中的重复数据？

如果每一列都一样，您可以对所有列执行GROUP BY操作。例如，非分区表*table1*的列为c1，c2和c3，您可以执行如下命令。

```
insert overwrite table table1 select c1, c2, c3 from table1 group by
 c1, c2, c3;
```

> **说明** 建议您在执行此操作前，做好数据备份工作并根据数据量评估此方式的代价是否比重新导入的代价低。

3.8.3. DQL操作常见问题

本文为您介绍在MaxCompute中执行DQL操作过程中的常见问题。

问题类别	常见问题
GROUP BY	• 在执行MaxCompute SQL过程中，报错Repeated key in GROUP BY，如何解决？ • 在执行MaxCompute SQL过程中，报错Expression not in GROUP BY key，如何解决？ • 对表A执行GROUP BY生成表B，表B比表A的行数少，但表B的物理存储量是表A的10倍，是什么原因造成的？ • 使用GROUP BY分组查询100亿条数据会不会影响性能？GROUP BY对数据量有没有限制？
ORDER BY	• MaxCompute查询得到的数据是根据什么排序的？ • MaxCompute是否支持ORDER BY FIELD NULLS LAST语法？ • 执行MaxCompute SQL过程中，报错ORDER BY must be used with a LIMIT clause，如何解决？
子查询	在执行MaxCompute SQL过程中，使用NOT IN后面接子查询，子查询返回的结果是上万级别的数据量，但当IN和NOT IN后面的子查询返回的是分区时，返回的数量上限为1000。在必须使用NOT IN的情况下，该如何实现此查询？
交集、并集和补集	• 如何合并两个没有任何关联关系的表？ • 在执行UNION ALL操作时，报错ValidateJsonSize error，如何解决？
JOIN	• 在执行JOIN操作时，报错Both left and right aliases encountered in JOIN，如何解决？ • 在执行JOIN操作时，报错Maximum 16 join inputs allowed，如何解决？ • 在执行JOIN操作时，发现JOIN结果数据条数比原表多，如何解决？ • 在执行JOIN操作时，分区裁剪条件放在ON中分区裁剪会生效，还是放在WHERE中才会生效？
MAPJOIN	• 如何用MAPJOIN缓存多张小表？ • MAPJOIN中的大表和小表是否可以互换位置？
其他	• MaxCompute SQL设置过滤条件后，报错提示输入的数据超过100 GB，如何解决？ • MaxCompute SQL中模糊查询的WHERE条件是否支持正则表达式？ • 如果只同步100条数据，如何在过滤条件WHERE中通过LIMIT实现？ • 如何能提高查询效率？分区设置能调整吗？ • MaxCompute SQL支持WITH AS语句吗？ • 如何将一行数据拆分为多行数据？ • 在客户端的odps_config.ini文件中设置use_instance_tunnel=false，instance_tunnel_max_record=10，为什么Select还是能输出很多记录？ • 如何用正则表达式判断字段是否为中文？

在执行MaxCompute SQL过程中，报错Repeated key in GROUP BY，如何解决？

- 问题现象

  在执行MaxCompute SQL过程中，返回报错如下。

  ```
  FAILED: ODPS-0130071:Semantic analysis exception - Repeated key in GROUP BY。
  ```

- 产生原因

  SELECT DISTINCT后不能跟常量。

- 解决措施

  将SQL拆分为两层，内层处理没有常量的DISTINCT逻辑，外层加入常量数据。

在执行MaxCompute SQL过程中，报错Expression not in GROUP BY key，如何解决？

- 问题现象

  执行MaxCompute SQL时，返回报错如下。

  ```
  FAILED: ODPS-0130071:Semantic analysis exception - Expression not in GROUP BY key : line 1:xx 'xxx'
  ```

- 产生原因

  不支持直接引用非GROUP BY的列。详情请参见GROUP BY分组查询（col_list）。

- 解决措施

  修改SQL语句，确保在GROUP BY子句中，SELECT查询的列，必须是GROUP BY中的列或聚合函数（例如SUM或COUNT）加工过的列。

对表A执行GROUP BY生成表B，表B比表A的行数少，但表B的物理存储量是表A的10倍，是什么原因造成的？

数据在MaxCompute中是列式压缩存储的，如果同一列的前后数据的内容是相似的，压缩比会比较高。当 `odps.sql.groupby.skewindata=true` 打开时，使用SQL写入数据，数据比较分散，压缩比较小。如果希望数据的压缩比较高，您可以在使用SQL写入数据时进行局部排序。

使用GROUP BY分组查询100亿条数据会不会影响性能？GROUP BY对数据量有没有限制？

使用GROUP BY分组查询100亿条数据不会影响性能。GROUP BY对数据量无限制。GROUP BY分组查询详情请参见GROUP BY分组查询（col_list）。

MaxCompute查询得到的数据是根据什么排序的？

MaxCompute中表的读取是无序的。如果您没有进行自定义设置，查询获取的结果也是无序的。

如果您对数据的顺序有要求，需要对数据进行排序。例如，在SQL中需要加上 `order by xx limit n` 对数据进行排序。

如果您需要对数据进行全排序，只需要将 `limit` 面的 `n` 设置为 `数据总条数+1` 即可。

> **说明** 海量数据的全排序，对性能的影响非常大，而且很容易造成内存溢出问题，请尽量避免执行该操作。

MaxCompute是否支持ORDER BY FIELD NULLS LAST语法？

MaxCompute不支持此语法。MaxCompute支持的语法请参见与其他SQL语法的差异。

执行MaxCompute SQL过程中，报错ORDER BY must be used with a LIMIT clause，如何解决？

- 问题现象

  执行MaxCompute SQL过程中，返回报错如下。

  ```
  FAILED: ODPS-0130071:[1,27] Semantic analysis exception - ORDER BY must be used with a LIMIT clause, please set odps.sql.validate.orderby.limit=false to use it.
  ```

- 产生原因

  ORDER BY需要对单个执行节点做全局排序，所以默认带LIMIT限制，避免误用导致单点处理大量数据。

- 解决措施

  如果您的使用场景确实需要ORDER BY放开LIMIT限制，可以通过如下两种方式实现：

  - Project级别：设置 `setproject odps.sql.validate.orderby.limit=false;` 关闭 `order by` 必须带 `limit` 的限制。
  - Session级别：设置 `set odps.sql.validate.orderby.limit=false;` 关闭 `order by` 必须带 `limit` 的限制，需要与SQL语句一起提交。

    > **说明** 如果关闭 `order by` 必须带 `limit` 的限制，在单个执行节点有大量数据排序的情况下，资源消耗或处理时长等性能表现会受到影响。

更多ORDER BY信息，请参见ORDER BY全局排序（order_condition）。

在执行MaxCompute SQL过程中，使用NOT IN后面接子查询，子查询返回的结果是上万级别的数据量，但当IN和NOT IN后面的子查询返回的是分区时，返回的数量上限为1000。在必须使用NOT IN的情况下，该如何实现此查询？

您可以使用 `left outer join` 命令查询。

```
select * from a where a.ds not in (select ds from b);
改成如下语句。
select a.* from a left outer join (select distinct ds from b) bb on a.ds=bb.ds where bb.ds is not null;
```

如何合并两个没有任何关联关系的表？

您可以通过 `union all` 运算完成纵向合并。横向合并可以通过 `row_number` 函数实现，两个表都新加一个ID列，进行ID关联，然后取两张表的字段。更多信息，请参见并集或ROW_NUMBER。

在执行UNION ALL操作时，报错ValidateJsonSize error，如何解决？

- 问题现象

  执行包含200个UNION All的SQL语句 `select count(1) as co from client_table union all ...` 时，返回报错如下。

  ```
  FAILED: build/release64/task/fuxiWrapper.cpp(344): ExceptionBase: Submit fuxi Job failed, {
      "ErrCode": "RPC_FAILED_REPLY",
      "ErrMsg": "exception: ExceptionBase:build/release64/fuxi/fuximaster/fuxi_master.cpp(1018): ExceptionBase: StartAppFail: ExceptionBase:build/release64/fuxi/fuximaster/app_master_mgr.cpp(706): ExceptionBase: ValidateJsonSize error: the size of compressed plan is larger than 1024KB\nStack
  ```

- 产生原因

  - 原因一：SQL语句转化为执行计划后，超过了底层架构限制的1024 KB，导致SQL执行报错。执行计划的长度与SQL语句长度没有直接换算关系，暂时无法预估。

- 原因二：由于分区量过大导致执行计划超过限制。
- 原因三：由于小文件比较多导致SQL运行失败。

- 解决措施
  - 原因一的解决措施：对于过长的SQL语句，建议拆分成多次运行，避免触发长度限制。
  - 原因二的解决措施：如果分区过大，需要调整分区个数，详情请参见分区。
  - 原因三的解决措施：如果是由于小文件较多导致，请参见小文件优化及作业诊断常见问题。

在执行JOIN操作时，报错Both left and right aliases encountered in JOIN，如何解决？

- 问题现象

  执行MaxCompute SQL过程中，返回报错如下。

```
FAILED: ODPS-0130071:Semantic analysis exception - Both left and right aliases encountered in JOIN : line 3:3 'xx': . I f you really want to perform this join, try mapjoin
```

- 产生原因
  - 原因一：SQL关联条件ON中包含非等值连接，例如 `table1.c1>table2.c3` 。
  - 原因二：SQL中JOIN条件的某一侧数据来自两张表，例如 `table1.col1 = concat(table1.col2,table2.col3)` 。

- 解决措施
  - 原因一的解决措施：修改SQL语句，关联条件需要为等值连接。
  - 原因二的解决措施：如果其中一张表比较小，您可以使用MAPJOIN方法。

在执行JOIN操作时，报错Maximum 16 join inputs allowed，如何解决？

- 问题现象

  在执行MaxCompute SQL过程中，返回报错如下。

```
FAILED: ODPS-0123065:Join exception - Maximum 16 join inputs allowed
```

- 产生原因

  MaxCompute SQL最多支持6张小表的MAPJOIN，并且连续JOIN的表不能超过16张。

- 解决措施

  将部分小表JOIN成一张临时表作为输入表，减少输入表的个数。

在执行JOIN操作时，发现JOIN结果数据条数比原表多，如何解决？

- 问题现象

  执行如下MaxCompute SQL语句后，查询返回结果的条数大于*table1*的数据条数。

```
select count(*) from table1 a left outer join table2 b on a.ID = b.ID;
```

- 产生原因

  示例SQL是table1通过ID字段和table2的ID字段做左外关联，所以会出现以下情况：
  - 如果table2表中找不到关联数据，table1也会返回一条数据。
  - 如果table1找不到但是table2能找到关联数据，则不返回结果。
  - 如果table1和table2都能找到关联数据，该关联逻辑和普通的内关联一样。如果同样的ID字段在table2中能找到数据，返回结果为table1和table2的笛卡尔积。

  假设table1的数据如下。

id	values
1	a
1	b
2	c

  假设table2的数据如下。

id	values
1	A
1	B
3	D

执行 `select count(*) from table1 a left outer join table2 b on a.ID = b.ID;` 命令返回的结果如下。

id1	values1	id2	values2
1	b	1	B
1	b	1	A
1	a	1	B
1	a	1	A
2	c	NULL	NULL

- id=1的数据两边都有，执行笛卡尔积，返回4条数据。
- id=2的数据只有table1有，返回了1条数据。
- id=3的数据只有table2有，table1里没数据，不返回数据。

- 解决措施

  确认出现数据条数增加是否是因为table2的数据导致。命令示例如下。此处增加 `limit 10` 是考虑到如果table2中的数据条数很多，会刷屏。如果只是确认问题，验证前几条数据即可。

```
select id, count() as cnt from table2 group by id having cnt>1 limit 10;
```

  如果是在重复的情况下不希望执行笛卡尔积，希望有类似SQL里IN的功能，可以改写SQL为如下语句。

```
select * from table1 a left outer join (select distinct id from table2) b on a.id = b.id;
```

在执行JOIN操作时，已经指定了分区条件，为何提示禁止全表扫描？

- 问题现象

  在两个项目里执行如下同一段代码，一个项目中成功，一个项目中失败。

```
select t.stat_date
from fddev.tmp_001 t
left outer join (select '20180830' as ds from fddev.dual ) t1
on t.ds = 20180830
group by t.stat_date;
```

  失败报错如下。

```
Table(fddev, tmp_001) is full scan with all partisions,please specify partition predicates.
```

- 产生原因

  在执行SELECT操作时，如果需要指定分区请使用WHERE子句。使用ON属于非标准用法。

  执行成功的项目设置了允许非标准SQL的行为，即执行了 `set odps.sql.outerjoin.supports.filters=true` 命令，该配置会把ON里的条件转为换成过滤条件，可用于兼容HIVE语法，但不符合SQL标准。

- 解决措施

  建议将分区过滤条件置于WHERE子句。

在执行JOIN操作时，分区裁剪条件放在ON中分区裁剪会生效，还是放在WHERE中才会生效？

如果分区剪裁条件置于WHERE语句中，分区剪裁会生效。如果分区剪裁条件置于ON语句中，从表的分区剪裁会生效，主表的分区剪裁不会生效即会全表扫描。更多分区裁剪信息，请参见分区剪裁合理性评估。

如何用MAPJOIN缓存多张小表？

您可以在MAPJOIN中填写表的别名。

假设项目中存在一张表iris，表数据如下。

```
+-----------------------------------------+
| Field          | Type       | Label | Comment                                  |
+-----------------------------------------+
| sepal_length   | double     |       |                                          |
| sepal_width    | double     |       |                                          |
| petal_length   | double     |       |                                          |
| petal_width    | double     |       |                                          |
| category       | string     |       |                                          |
+-----------------------------------------+
```

使用MAPJOIN实现缓存小表的命令示例如下。

```
select
  /*+ mapjoin(b,c) */
  a.category,
  b.cnt as cnt_category,
  c.cnt as cnt_all
from iris a
join
(
  select count() as cnt,category from iris group by category
) b
on a.category = b.category
join
(
  select count(*) as cnt from iris
) c;
```

MAPJOIN中的大表和小表是否可以互换位置？

MAPJOIN中的大表和小表是根据表占用空间Size大小区分的。

系统会将您指定的小表全部加载到执行JOIN操作的程序的内存中，继而加快JOIN的执行速度。如果将大表和小表互换位置，系统不会报错，但是性能会变差。

MaxCompute SQL设置过滤条件后，报错提示输入的数据超过100 GB，如何解决？

先过滤分区，再取数据。取数据后，再过滤其他非分区字段。输入表的大小是取决于过滤分区过滤后，过滤其他字段前表的大小。

MaxCompute SQL中模糊查询的WHERE条件是否支持正则表达式？

支持，例如 `select * from user_info where address rlike '[0-9]{9}';` ，表示查找9位数字组成的ID。

如果只同步100条数据，如何在过滤条件WHERE中通过LIMIT实现？

LIMIT不支持在过滤条件中使用。您可以先使用SQL筛选出100条数据，再执行同步操作。

如何能提高查询效率？分区设置能调整吗？

当利用分区字段对表进行分区时，新增分区、更新分区和读取分区数据均不需要做全表扫描，可以提高处理效率。详情请参见表操作和MaxCompute的分区配置和使用。

MaxCompute SQL支持WITH AS语句吗？

支持，MaxCompute支持SQL标准的CTE，提高SQL语句的可读性与执行效率。更多信息，请参见COMMON TABLE EXPRESSION（CTE）。

如何将一行数据拆分为多行数据？

Lateral View和表生成函数（例如Split和Explode）结合使用，可以将一行数据拆成多行数据，并对拆分后的数据进行聚合。

在客户端的odps_config.ini文件中设置use_instance_tunnel=false，instance_tunnel_max_record=10，为什么Select还是能输出很多记录？

需要修改 `use_instance_tunnel=false` 为 `use_instance_tunnel=true` ，才能通过 `instance_tunnel_max_record` 控制输出记录数。

如何用正则表达式判断字段是否为中文？

命令示例如下。

```
select '字段' rlike '[\\x{4e00}-\\x{9fa5}]+';
```

3.8.4. SQL其他常见问题

本文为您介绍在MaxCompute中执行SQL时，与数据类型、SQL限制等相关的常见问题。

问题类别	常见问题
数据类型	• MaxCompute的时间类型字段是否可以不带时分秒？ • 在执行MaxCompute SQL过程中，对DOUBLE类型的数据进行等值比较，为什么结果不符合预期？ • 如何解决DECIMAL数据类型精度溢出问题？ • 新创建的MaxCompute项目不支持数据类型自动隐式转换，如何解决？ • 查询数据时，提示XXXtypeisnotenabled incurrentmode，如何解决？

问题类别	常见问题
SQL限制	• 如何处理单字段大于8 MB的限制？ • 在执行MaxCompute SQL过程中，报错partitions exceeds the specified limit，如何解决？
SQL作业运行	• 如何非交互式运行MaxCompute SQL？ • SQL能将MaxCompute的配置转移到另外一个阿里云账号上吗？ • 如何将开发环境的表数据同步至生产环境的表中？ • 如何判断一个字段是否为空？ • 是否可以通过DataWorks的Shell节点调取MaxCompute SQL？ • 如何在SQL中实现循环？ • 如何在SQL中调用赋值节点？
SQL作业查看	• 如何查看MaxCompute日执行的所有SQL？ • 如何查看SQL作业消耗的资源量？ • 如何查看SQL执行费用？
SQL作业调优	SQL作业运行过慢，如何优化？

MaxCompute的时间类型字段是否可以不带时分秒？

时间类型字段可以使用DATE数据类型。使用该数据类型时，您需要打开MaxCompute 2.0数据类型开关。更多MaxCompute 2.0数据类型信息，请参见2.0数据类型版本。

在执行MaxCompute SQL过程中，对DOUBLE类型的数据进行等值比较，为什么结果不符合预期？

由于MaxCompute中DOUBLE类型的数值存在一定的精度差，因此不建议直接使用等号（=）对两个DOUBLE类型的数据进行比较。

请对两个DOUBLE类型数据相减，然后取绝对值，当绝对值足够小时，判定两个DOUBLE类型的数据数值相等。

如何解决DECIMAL数据类型精度溢出问题？

您可以在Session级别执行 `set odps.sql.decimal.odps2=true;` 命令，打开2.0数据类型开关。但您需要注意，MaxCompute支持的DECIMAL类型数据的最大长度为38位，但实际业务处理过程中如果数据存储为最大长度，在数据处理过程中很容易出现数据溢出问题，建议调小数据长度。

新创建的MaxCompute项目不支持数据类型自动隐式转换，如何解决？

关闭MaxCompute 2.0数据类型之后可进行隐式转换。更多数据类型转换信息，请参见数据类型转换。

查询数据时，提示XXXtypeisnotenabled incurrentmode，如何解决？

您可以在Session级别执行 `set odps.sql.decimal.odps2=true;` 命令，打开2.0数据类型开关。

如何处理单字段大于8 MB的限制？

由于存储机制限制，MaxCompute表中单个字段的最大长度不能超过8 MB。对于超过8 MB的字段，建议您拆分成多个字段。具体的拆分逻辑您可以根据业务特性设计，保证每个字段不超过8 MB即可。

由于复杂结构的超大字段在数据开发和分析中会严重影响计算性能，因此建议根据数据仓库建设规范来设计您的数据架构，避免出现超大字段：

- 具有复杂结构的原始数据，作为ODS层，最好以压缩的方式归档。
- 定时（例如每天）对ODS层的增量数据做数据清洗，复杂字段拆分为多个简单字段，然后存储在CDM层的表中，便于统计和分析数据。

在执行MaxCompute SQL过程中，报错partitions exceeds the specified limit，如何解决？

- 问题现象

  在执行MaxCompute SQL过程中，返回报错如下。

  ```
  FAILED: ODPS-0010000:System internal error - OTS filtering exception - Ots read range partitions exceeds the specified limit:10000:  tableName:xxxx , please check hive conf key
  ```

- 产生原因

  MaxCompute单张表支持6万个分区，但是单次查询最多只支持1万个分区。该报错常见原因如下：

  - 原因一：分区未写分区条件。
  - 原因二：使用类似用户ID的字段作为分区字段，导致分区数量过多。

- 解决措施
  - 原因一的解决措施：如果未写分区条件，补上分区条件即可。
  - 原因二的解决措施：如果分区列不合适，导致分区数量太多，请考虑更改分区列。

如何非交互式运行MaxCompute SQL？

在操作系统中，您可以通过Shell非交互式运行MaxCompute SQL:

- 使用 `odps -f filename` 方式，读取并处理SQL文件。

  如果运行SQL，Filename文件的第一行是 `SQL` 表示已经进入SQL模式。命令示例如下。

  ```
  SQL
    select ... from table_name where xxx;
  ```

- 如果只运行一个SQL语句，您可以使用MaxCompute SQL中的 `sqltext` 方法，命令示例如下。

  ```
  ./odpscmd -e "select * from dual;"
  ```

  您可以通过 `odps -help` 获得更多的信息。

SQL能将MaxCompute的配置转移到另外一个阿里云账号上吗？

您可以通过Package方法实现。更多Package授权信息，请参见MaxCompute多团队协同数据开发项目管理最佳实践。

如何将开发环境的表数据同步至生产环境的表中？

您可以通过MaxCompute客户端执行如下命令。

```
insert into project.table select * from project_dev.table;
```

如果没有生产环境中表的读写权限，需要完成账号授权。更多授权信息，请参见权限列表。

如何判断一个字段是否为空？

您可以通过MaxCompute SQL的运算符判断字段是否为空，更多运算符信息，请参见运算符。

是否可以通过DataWorks的Shell节点调取MaxCompute SQL？

不可以。Shell节点仅支持标准Shell语法，不支持交互性语法。如果作业较多，您可以使用ODPS SQL节点执行作业。更多ODPS SQL节点信息，请参见创建ODPS SQL节点。

如何在SQL中实现循环？

您可以通过DataWorks的do-while节点实现。

如何在SQL中调用赋值节点？

您可以通过DataWorks的for-each节点实现。

如何查看MaxCompute日执行的所有SQL？

您可以执行 `show p -all -limit <number>;` 命令查看MaxCompute项目下所有成员历史执行的SQL信息。更多查看实例信息，请参见查看实例信息。

如何查看SQL作业消耗的资源量？

您可以通过账单查看SQL作业消耗的资源量。更多查看账单实践信息，请参见MaxCompute账单分析最佳实践。

如何查看SQL执行费用？

您可以使用 `cost sql` 命令预估查询费用。更多命令信息，请参见计量预估。

SQL作业运行过慢，如何优化？

SQL作业可以通过Logview进行定位，定位方法请参见通过Logview诊断慢作业。

更多优化SQL作业信息，请参见计算优化最佳实践。

3.9. 内建函数

3.9.1. 内建函数概述

MaxCompute自身预置了诸多函数，可以满足大部分业务场景的数据处理需求。本文为您介绍MaxCompute提供的函数类型及函数使用相关说明。

背景信息

MaxCompute预置的函数类型如下。

函数类型	说明
日期函数	支持处理DATE、DATETIME、TIMESTAMP等日期类型数据，实现加减日期、计算日期差值、提取日期字段、获取当前时间、转换日期格式等业务处理能力。
数学函数	支持处理BIGINT、DOUBLE、DECIMAL、FLOAT等数值类型数据，实现转换进制、数学运算、四舍五入、获取随机数等业务处理能力。

函数类型	说明
窗口函数	支持在指定的开窗列中，实现求和、求最大最小值、求平均值、求中间值、数值排序、数值偏移、抽样等业务处理能力。
聚合函数	支持将多条输入记录聚合成一条输出值，实现求和、求平均值、求最大最小值、求平均值、参数聚合、字符串连接等业务处理能力。
字符串函数	支持处理STRING类型字符串，实现截取字符串、替换字符串、查找字符串、转换大小写、转换字符串格式等业务处理能力。
复杂类型函数	支持处理MAP、ARRAY、STRUCT及JSON类型数据，实现去重元素、聚合元素、元素排序、合并元素等业务处理能力。
其他函数	除上述函数之外，提供支持其他业务场景的函数。

各类型函数与开源函数的对照关系，请参见与Hive、MySQL、Oracle内建函数对照表。

注意事项

在使用内建函数时，需要注意：

- 内建函数的入参类型、入参数量、函数格式必须满足函数语法要求，否则MaxCompute无法成功解析函数，SQL运行会报错。
- 如果内建函数的入参涉及2.0新数据类型（例如TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP或BINARY），您需要打开2.0新数据类型开关，否则运行会报错。开关打开方式如下：
  - Session级别：您需要在SQL语句前加上 `set odps.sql.type.system.odps2=true;` ，与SQL语句一起提交执行。该配置仅对本次运行的SQL有效。
  - Project级别：Project Owner可根据需要对MaxCompute项目进行设置，等待10~15分钟后才会生效。该配置对后续运行的所有SQL有效。

    ```
    setproject odps.sql.type.system.odps2=true;
    ```

- 当MaxCompute项目打开2.0新数据类型开关时，部分隐式类型转换会被禁用，包括STRING转换为BIGINT、STRING转换为DATETIME、DOUBLE转换为BIGINT、DECIMAL转换为DOUBLE、DECIMAL转换为BIGINT，都有精度损失或报错的风险。此时，您可以通过CAST函数执行强制转换解决该问题，或关闭2.0新数据类型开关。
- 当自定义函数的名称与内建函数的名称相同时，自定义函数会覆盖同名的内建函数。例如，MaxCompute中存在一个名称为CONCAT的自定义函数，则系统默认会调用自定义的CONCAT，不会调用内建函数CONCAT。如果您希望调用内建函数，需要在内建函数前增加 `::` 符号，例如 `select :: concat('ab', 'c');` 。
- 如果MaxCompute项目的全局属性设置不同，内建函数运行的结果可能会不一致，您可以通过 `setproject;` 命令查看项目的全局属性信息。

3.9.2. 与Hive、MySQL、Oracle内建函数对照表

本文为您提供MaxCompute、Hive、MySQL以及Oracle的内建函数对照表，方便您根据Hive、MySQL、Oracle内建函数查找对应的MaxCompute内建函数。

日期函数

MaxCompute	Hive	MySQL	Oracle
DATEADD	无	无	无
DATE_ADD	DATE_ADD	DATE_ADD	无
DATE_FORMAT	DATE_FORMAT	DATE_FORMAT	无
DATE_SUB	DATE_SUB	DATE_SUB	无
DATEDIFF	DATEDIFF	DATEDIFF	MONTHS_BETWEEN
DATEPART	无	DATE_FORMAT	EXTRACT (DATETIME)
DATETRUNC	TRUNC	DATE_FORMAT	EXTRACT (DATETIME)
FROM_UNIXTIME	FROM_UNIXTIME	FROM_UNIXTIME	无
GETDATE	CURRENT_DATE	NOW	CURRENT_DATE
ISDATE	无	STR_TO_DATE（返回FALSE则说明不能转换为DATE）	无
LASTDAY	LAST_DAY	LAST_DAY	LAST_DAY
TO_DATE	TO_DATE	STR_TO_DATE	DATE

MaxCompute	Hive	MySQL	Oracle
TO_CHAR	无	DATE_FORMAT	TO_CHAR (DATETIME)
UNIX_TIMESTAMP	UNIX_TIMESTAMP	UNIX_TIMESTAMP	无
WEEKDAY	无	WEEKDAY	无
WEEKOFYEAR	WEEKOFYEAR	WEEKOFYEAR	无
ADD_MONTHS	ADD_MONTHS	ADDDATE	ADD_MONTHS
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP
DAY	DAY	DAY	DAY
DAYOFMONTH	DAYOFMONTH	DAYOFMONTH	无
EXTRACT	EXTRACT	EXTRACT	EXTRACT
FROM_UTC_TIMESTAMP	FROM_UTC_TIMESTAMP	无	无
HOUR	HOUR	HOUR	HOUR
LAST_DAY	LAST_DAY	LAST_DAY	无
MINUTE	MINUTE	MINUTE	MINUTE
MONTH	MONTH	MONTH	MONTH
MONTHS_BETWEEN	MONTHS_BETWEEN	TIMESTAMPDIFF	MONTHS_BETWEEN
NEXT_DAY	NEXT_DAY	无	NEXT_DAY
QUARTER	QUARTER	QUARTER	QUARTER
SECOND	SECOND	SECOND	无
TO_MILLIS	无	无	无
YEAR	YEAR	YEAR	无

说明 默认模式为MaxCompute模式，如果要切换至Hive模式，执行如下命令切换。

```
--Project级别切换为Hive模式。
setproject odps.sql.hive.compatible=True;
--Session级别切换为Hive模式。
set odps.sql.hive.compatible=True;
```

数学函数

MaxCompute	Hive	MySQL	Oracle
ABS	ABS	ABS	ABS
ACOS	ACOS	ACOS	ACOS
ASIN	ASIN	ASIN	ASIN
ATAN	ATAN	ATAN	ATAN
CEIL	CEIL	CEIL	CEIL
CONV	CONV	CONV	无
COS	COS	COS	COS
COSH	COSH	无	COSH
COT	COT	COT	COT
EXP	EXP	EXP	EXP

MaxCompute	Hive	MySQL	Oracle
FLOOR	FLOOR	FLOOR	FLOOR
LN	LN	LN	LN
LOG	LOG	LOG	LOG
POW	POW	POW	POWER
RAND	RAND	RAND	无
ROUND	ROUND	ROUND	ROUND
SIN	SIN	SIN	SIN
SINH	SINH	无	SINH
SQRT	SQRT	SQRT	SQRT
TAN	TAN	TAN	TAN
TANH	TANH	无	TANH
TRUNC	TRUNC	TRUNCATE	TRUNC
BIN	BIN	BIN	BITAND
CBRT	CBRT	无	无
CORR	CORR	CORR	CORR
DEGREES	DEGREES	DEGREES	DEGREES
E	E	无	无
FACTORIAL	FACTORIAL	无	无
FORMAT_NUMBER	FORMAT_NUMBER	FORMAT	无
HEX	HEX	HEX	RAWTOHEX
LOG2	LOG2	LOG2	LOG
LOG10	LOG10	LOG10	LOG
PI	PI	PI	PI
RADIANS	RADIANS	RADIANS	RADIANS
SIGN	SIGN	SIGN	SIGN
SHIFTLEFT	SHIFTLEFT	<<	无
SHIFTRIGHT	SHIFTRIGHT	>>	无
SHIFTRIGHTUNSIGNED	SHIFTRIGHTUNSIGNED	>>>	无
UNHEX	UNHEX	UNHEX	HEXTORAW
WIDTH_BUCKET	WIDTH_BUCKET	无	WIDTH_BUCKET

说明 默认模式为MaxCompute模式，如果要切换至Hive模式，执行如下命令切换。

```
--Project级别切换为Hive模式。
setproject odps.sql.hive.compatible=True;
--Session级别切换为Hive模式。
set odps.sql.hive.compatible=True;
```

窗口函数

MaxCompute	Hive	MySQL	Oracle
COUNT	COUNT	COUNT	COUNT
窗口函数	AVG	AVG	AVG
MAX	MAX	MAX	MAX
MIN	MIN	MIN	MIN
窗口函数	无	无	MEDIAN
STDDEV	无	STDDEV	STDDEV
STDDEV_SAMP	无	STDDEV_SAMP	STDDEV_SAMP
SUM	SUM	SUM	SUM
窗口函数	DENSE_RANK	DENSE_RANK	DENSE_RANK
RANK	RANK	RANK	RANK
LAG	LAG	LAG	LAG
LEAD	LEAD	LEAD	LEAD
PERCENT_RANK	PERCENT_RANK	PERCENT_RANK	PERCENT_RANK
ROW_NUMBER	ROW_NUMBER	ROW_NUMBER	ROW_NUMBER
CLUSTER_SAMPLE	无	无	无
窗口函数	CUME_DIST	CUME_DIST	CUME_DIST
NTILE	NTILE	NTILE	NTILE

聚合函数

MaxCompute	Hive	MySQL	Oracle
AVG	AVG	AVG	AVG
COUNT	COUNT	COUNT	COUNT
COUNT_IF	无	无	无
MAX	MAX	MAX	MAX
MIN	MIN	MIN	MIN
MEDIAN	无	无	MEDIAN
STDDEV	STDDEV	STDDEV	STDDEV
STDDEV_SAMP	STDDEV_SAMP	STDDEV_SAMP	STDDEV_SAMP
SUM	SUM	SUM	SUM
WM_CONCAT	无	GROUP_CONCAT	WM_CONCAT
ANY_VALUE	无	无	无
APPROX_DISTINCT	无	无	无
ARG_MAX	无	无	无
ARG_MIN	无	无	无
COLLECT_LIST	COLLECT LIST	无	COLLECT
COLLECT_SET	COLLECT SET	无	COLLECT
COVAR_POP	COVAR_POP	无	COVAR_POP
COVAR_SAMP	COVAR_SAMP	无	COVAR_SAMP

MaxCompute	Hive	MySQL	Oracle
NUMERIC_HISTOGRAM	NUMERIC_HISTOGRAM	无	无
PERCENTILE	PERCENTILE	无	无
PERCENTILE_APPROX	PERCENTILE_APPROX	无	无
VARIANCE/VAR_POP	VARIANCE/VAR_POP	VAR_POP	VARIANCE/VAR_POP
VAR_SAMP	VAR_SAMP	VAR_SAMP	VAR_SAMP

说明 默认模式为MaxCompute模式，如果要切换至Hive模式，执行如下命令切换。

```
--Project级别切换为Hive模式。
setproject odps.sql.hive.compatible=True;
--Session级别切换为Hive模式。
set odps.sql.hive.compatible=True;
```

字符串函数

MaxCompute	Hive	MySQL	Oracle
ASCII	ASCII	ASCII	ASCII
CHAR_MATCHCOUNT	无	无	无
CHR	CHR	CHAR	CHR
CONCAT	CONCAT	CONCAT	CONCAT
ENCODE	ENCODE	无	无
FIND_IN_SET	FIND_IN_SET	FIND_IN_SET	无
FORMAT_NUMBER	FORMAT_NUMBER	FORMAT	无
FROM_JSON	无	无	无
GET_JSON_OBJECT	GET_JSON_OBJECT	JSON_EXTRACT	无
INSTR	INSTR	INSTR	INSTR
IS_ENCODING	无	无	无
KEYVALUE	无	无	无
LENGTH	LENGTH	LENGTH	LENGTH
LENGTHB	LENGTHB	LENGTHB	LENGTHB
LOCATE	LOCATE	LOCATE	无
LTRIM	LTRIM	LTRIM	LTRIM
MD5	MD5	MD5	无
PARSE_URL	PARSE_URL	无	无
PARSE_URL_TUPLE	PARSE_URL_TUPLE	无	无
REGEXP_COUNT	无	无	REGEXP_COUNT
REGEXP_EXTRACT	REGEXP_EXTRACT	无	无
REGEXP_INSTR	无	REGEXP_INSTR	REGEXP_INSTR
REGEXP_REPLACE	REGEXP_REPLACE	REGEXP_REPLACE	REGEXP_REPLACE
REGEXP_SUBSTR	无	REGEXP_SUBSTR	REGEXP_SUBSTR
REPEAT	REPEAT	REPEAT	REPEAT

MaxCompute	Hive	MySQL	Oracle
REVERSE	REVERSE	REVERSE	REVERSE
RTRIM	RTRIM	RTRIM	RTRIM
SPACE	SPACE	SPACE	SPACE
SPLIT_PART	无	无	无
SUBSTR	SUBSTR	SUBSTR	SUBSTR
SUBSTRING	SUBSTRING	SUBSTRING	SUBSTR
TO_CHAR	无	无	无
TO_JSON	无	无	无
TOLOWER	LOWER	LOWER	LOWER
TOUPPER	UPPER	UPPER	UPPER
TRIM	TRIM	TRIM	TRIM
URL_DECODE	无	无	无
URL_ENCODE	无	无	PERCENTILE_CONT
CONCAT_WS	CONCAT_WS	CONCAT_WS	无
JSON_TUPLE	JSON_TUPLE	无	无
LPAD	LPAD	LPAD	LPAD
RPAD	RPAD	RPAD	RPAD
REPLACE	REPLACE	REPLACE	REPLACE
SOUNDEX	SOUNDEX	SOUNDEX	SOUNDEX
SUBSTRING_INDEX	SUBSTRING_INDEX	SUBSTRING_INDEX	无
TRANSLATE	TRANSLATE	无	TRANSLATE

说明 默认模式为MaxCompute模式，如果要切换至Hive模式，执行如下命令切换。

```
--Project级别切换为Hive模式。
setproject odps.sql.hive.compatible=True;
--Session级别切换为Hive模式。
set odps.sql.hive.compatible=True;
```

其他函数

MaxCompute	Hive	MySQL	Oracle
BASE64	BASE64	TO_BASE64	UTL_ENCODE.BASE64_ENCODE
BETWEEN AND表达式	BETWEEN AND	BETWEEN AND	BETWEEN AND
CASE WHEN表达式	CASE WHEN	CASE WHEN	CASE WHEN
CAST	CAST	CAST	CAST
COALESCE	COALESCE	COALESCE	COALESCE
COMPRESS	无	COMPRESS	UTL_COMPRESS.LZ_COMPRESS
CRC32	CRC32	CRC32	无
DECODE	DECODE	无	DECODE
DECOMPRESS	无	UNCOMPRESS	UTL_COMPRESS.LZ_UNCOMPRESS

MaxCompute	Hive	MySQL	Oracle
GET_IDCARD_AGE	无	无	无
GET_IDCARD_BIRTHDAY	无	无	无
GET_IDCARD_SEX	无	无	无
GET_USER_ID	CURRENT_USER	CURRENT_USER	UID
GREATEST	GREATEST	GREATEST	无
HASH	HASH	无	ORA_HASH
IF	IF	IF	IF
LEAST	LEAST	LEAST	LEAST
MAX_PT	无	无	无
NULLIF	NULLIF	NULLIF	NULLIF
NVL	NVL	IFNULL	无
ORDINAL	无	无	无
PARTITION_EXISTS	无	无	无
SAMPLE	无	无	无
SHA	SHA	SHA	无
SHA1	SHA1	SHA1	无
SHA2	SHA2	SHA2	无
SIGN	SIGN	SIGN	SIGN
SPLIT	SPLIT	SPLIT	无
STACK	STACK	无	无
STR_TO_MAP	STR_TO_MAP	无	无
TABLE_EXISTS	无	无	无
TRANS_ARRAY	无	无	无
TRANS_COLS	无	无	无
UNBASE64	UNBASE64	FROM_BASE64	UTL_ENCODE.BASE64_DECODE
UNIQUE_ID	无	无	无
UUID	无	UUID	UID

说明 默认模式为MaxCompute模式，如果要切换至Hive模式，执行如下命令切换。

```
--Project级别切换为Hive模式。
setproject odps.sql.hive.compatible=True;
--Session级别切换为Hive模式。
set odps.sql.hive.compatible=True;
```

复杂类型函数

函数类别	MaxCompute	Hive	MySQL	Oracle
	ALL_MATCH	无	无	无
	ANY_MATCH	无	无	无
	ARRAY	ARRAY	无	无

函数类别	MaxCompute	Hive	MySQL	Oracle
ARRAY	ARRAY_CONTAINS	ARRAY_CONTAINS	无	无
	ARRAY_DISTINCT	无	无	无
	ARRAY_EXCEPT	无	无	无
	ARRAY_INTERSECT	无	无	无
	ARRAY_JOIN	无	无	无
	ARRAY_MAX	无	无	无
	ARRAY_MIN	无	无	无
	ARRAY_POSITION	无	无	无
	ARRAY_REMOVE	无	无	无
	ARRAY_REDUCE	无	无	无
	ARRAY_REPEAT	无	无	无
	ARRAY_SORT	无	无	无
	ARRAY_UNION	无	无	无
	ARRAYS_OVERLAP	无	无	无
	ARRAYS_ZIP	无	无	无
	CONCAT	CONCAT	无	无
	EXPLODE	EXPLODE	无	无
	FILTER	无	无	无
	INDEX	[]运算符	无	无
	POSEXPLODE	POSEXPLODE	无	无
	SIZE	SIZE	无	无
	SLICE	无	无	无
	SORT_ARRAY	SORT_ARRAY	无	无
	TRANSFORM	无	无	无
	ZIP_WITH	无	无	无
MAP	EXPLODE	EXPLODE	无	无
	INDEX	[]运算符	无	无
	MAP	MAP	无	无
	MAP_CONCAT	无	无	无
	MAP_ENTRIES	无	无	无
	MAP_FILTER	无	无	无
	MAP_FROM_ARRAYS	无	无	无
	MAP_FROM_ENTRIES	无	无	无
	MAP_KEYS	MAP_KEYS	无	无
	MAP_VALUES	MAP_VALUES	无	无
	MAP_ZIP_WITH	无	无	无
	SIZE	SIZE	无	无

函数类别	MaxCompute	Hive	MySQL	Oracle
	TRANSFORM_KEYS	无	无	无
	TRANSFORM_VALUES	无	无	无
STRUCT	FIELD	.运算符	无	无
	INLINE	INLINE	无	无
	STRUCT	STRUCT	无	无
	NAMED_STRUCT	无	无	无
JSON	FROM_JSON	无	无	无
	GET_JSON_OBJECT	GET_JSON_OBJECT	JSON_EXTRACT	无
	JSON_TUPLE	JSON_TUPLE	无	无
	TO_JSON	无	无	无

说明 默认模式为MaxCompute模式，如果要切换至Hive模式，执行如下命令切换。

```
--Project级别切换为Hive模式。
setproject odps.sql.hive.compatible=True;
--Session级别切换为Hive模式。
set odps.sql.hive.compatible=True;
```

3.9.3. 日期函数

MaxCompute SQL提供了常见的日期函数，您可以根据实际需要选择合适的日期函数，完成日期计算、日期转换。本文为您提供MaxCompute SQL支持的日期函数的命令格式、参数说明及示例，指导您使用日期函数完成开发。

函数	功能
DATEADD	按照指定的单位和幅度修改日期值。
DATE_ADD	按照指定的幅度增减天数，与 `date_sub` 的增减逻辑相反。
DATE_FORMAT	将日期值转换为指定格式的字符串。
DATE_SUB	按照指定的幅度增减天数，与 `date_add` 的增减逻辑相反。
DATEDIFF	计算两个日期的差值并按照指定的单位表示。
DATEPART	提取日期中符合指定时间单位的字段值。
DATETRUNC	提取日期按照指定时间单位截取后的值。
FROM_UNIXTIME	将数字型的UNIX值转换为日期值。
GETDATE	获取当前系统时间。
ISDATE	判断一个日期字符串能否根据指定的格式串转换为一个日期值。
LASTDAY	获取日期所在月的最后一天。
TO_DATE	将指定格式的字符串转换为日期值。
TO_CHAR	将日期按照指定格式转换为字符串。
UNIX_TIMESTAMP	将日期转换为整型的UNIX格式的日期值。
WEEKDAY	返回日期值是当前周的第几天。
WEEKOFYEAR	返回日期值位于当年的第几周。
ADD_MONTHS	计算日期值增加指定月数后的日期。
CURRENT_TIMESTAMP	返回当前TIMESTAMP类型的时间戳。

函数	功能
DAY	返回日期值的天。
DAYOFMONTH	返回日部分的值。
EXTRACT	获取日期TIMESTAMP中指定单位的部分。
FROM_UTC_TIMESTAMP	将一个UTC时区的时间戳转换为一个指定时区的时间戳。
HOUR	返回日期小时部分的值。
LAST_DAY	返回日期值所在月份的最后一天日期。
MINUTE	返回日期分钟部分的值。
MONTH	返回日期值所属月份。
MONTHS_BETWEEN	返回指定日期值间的月数。
NEXT_DAY	返回大于日期值且与指定周相匹配的第一个日期。
QUARTER	返回日期值所属季度。
SECOND	返回日期秒数部分的值。
TO_MILLIS	将指定日期转换为以毫秒为单位的UNIX时间戳。
YEAR	返回日期值的年。

注意事项

升级到MaxCompute 2.0后，产品扩展了部分函数。如果您用到的函数涉及新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP或BINARY），在使用扩展函数时，需要执行如下语句开启新数据类型开关：

- Session级别：如果使用新数据类型，您需要在SQL语句前加上语句 `set odps.sql.type.system.odps2=true;` ，并与SQL语句一起提交执行。
- Project级别：Project Owner可根据需要对Project进行设置，等待10~15分钟后才会生效。命令如下。

```
setproject odps.sql.type.system.odps2=true;
```

对 `setproject` 的详细说明请参见项目空间操作。关于开启Project级别数据类型的注意事项，请参见数据类型版本说明。

示例数据

为便于理解各函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表mf_date_fun_t，并添加数据，命令示例如下。

```
create table if not exists mf_date_fun_t(
    id      int,
    date1   date,
    datetime1   datetime,
    timestamp1 timestamp,
    date2   date,
    datetime2   datetime,
    timestamp2 timestamp,
    date3 string,
    date4 bigint);
insert into mf_date_fun_t values
(1,DATE'2021-11-29',DATETIME'2021-11-29 00:01:00',TIMESTAMP'2021-01-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-20',123456780),
(2,DATE'2021-11-28',DATETIME'2021-11-28 00:02:00',TIMESTAMP'2021-02-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-21',123456781),
(3,DATE'2021-11-27',DATETIME'2021-11-27 00:03:00',TIMESTAMP'2021-03-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-22',123456782),
(4,DATE'2021-11-26',DATETIME'2021-11-26 00:04:00',TIMESTAMP'2021-04-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-23',123456783),
(5,DATE'2021-11-25',DATETIME'2021-11-25 00:05:00',TIMESTAMP'2021-05-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-24',123456784),
(6,DATE'2021-11-24',DATETIME'2021-11-24 00:06:00',TIMESTAMP'2021-06-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-25',123456785),
(7,DATE'2021-11-23',DATETIME'2021-11-23 00:07:00',TIMESTAMP'2021-07-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-26',123456786),
(8,DATE'2021-11-22',DATETIME'2021-11-22 00:08:00',TIMESTAMP'2021-08-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-27',123456787),
(9,DATE'2021-11-21',DATETIME'2021-11-21 00:09:00',TIMESTAMP'2021-09-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29 0
0:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-28',123456788),
(10,DATE'2021-11-20',DATETIME'2021-11-20 00:10:00',TIMESTAMP'2021-10-11 00:00:00.123456789',DATE'2021-10-29',DATETIME'2021-10-29
00:00:00',TIMESTAMP'2021-10-11 00:00:00.123456789','2021-11-29',123456789);
```

DATEADD

- 命令格式

```
date|datetime dateadd(date|datetime|timestamp <date>, bigint <delta>, string <datepart>)
```

- 命令说明

  按照指定的单位datepart和幅度delta修改date的值。如果您需要获取在当前时间基础上指定变动幅度的日期，请结合GETDATE函数使用。

- 参数说明
  - date：必填。日期值，DATE、DATETIME或TIMESTAMP类型。

    如果参数为STRING类型，格式符合DATETIME类型的格式，即 `yyyy-mm-dd hh:mi:ss` ，例如 `2021-08-28 00:00:00` ，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。
  - delta：必填。修改幅度，BIGINT类型。如果delta大于0，则增，否则减。

    如果参数为STRING或DOUBLE类型，则会隐式转换为BIGINT类型后参与运算。

    > **说明**
    > - 按照指定的单位增减delta时，会导致更高单位的进位或退位。年、月、时、分秒分别按照10进制、12进制、24进制、60进制进行计算。
    > - 当delta的单位是月时，如果delta的月部分在增加delta值之后不造成Day溢出，则保持Day值不变，否则将Day值设置为结果月份的最后1天。

  - datepart：必填。指定修改的单位，STRING类型常量。非常量、不支持的格式或其他类型会返回报错。

    此字段的取值遵循STRING与DATETIME类型转换的约定，即 `yyyy` 表示年， `mm` 表示月， `dd` 表示天。关于类型转换的规则，详情请参见数据类型转换。该字段也支持扩展的日期格式：年 `-year` 、月 `-month` 或 `-mon` 、日 `-day` 和小时 `-hour` 。

- 返回值说明

  返回DATE或DATETIME类型，格式为 `yyyy-mm-dd` 或 `yyyy-mm-dd hh:mi:ss` 。返回规则如下：
  - date非DATE、DATETIME或TIMESTAMP类型时，返回报错。
  - date值为NULL时，返回报错。
  - delta或datepart值为NULL时，返回NULL。

- 示例

- 静态数据示例
  - 示例1：常见用法。

```
--返回2005-03-01 00:00:00。加1天，结果超出当年2月份的最后一天，实际值为下个月的第1天。
select dateadd(datetime '2005-02-28 00:00:00', 1, 'dd');
--返回2005-02-27 00:00:00。减1天。
select dateadd(datetime '2005-02-28 00:00:00', -1, 'dd');
--返回2006-10-28 00:00:00。加20个月，月份溢出，年份加1。
select dateadd(datetime '2005-02-28 00:00:00', 20, 'mm');
--返回2005-03-28 00:00:00。
select dateadd(datetime '2005-02-28 00:00:00', 1, 'mm');
--返回2005-02-28 00:00:00。2005年2月没有29日，日期截取至当月最后1天。
select dateadd(datetime '2005-01-29 00:00:00', 1, 'mm');
--返回2005-02-28 00:00:00。
select dateadd(datetime '2005-03-30 00:00:00', -1, 'mm');
--返回2005-03-18。
select dateadd(date '2005-02-18', 1, 'mm');
--开启2.0新类型。此命令需要与SQL语句一起提交。返回2005-03-18 00:00:00.0。
set odps.sql.type.system.odps2=true;
select dateadd(timestamp '2005-02-18 00:00:00', 1, 'mm');
--假设当前时间为2020-11-17 16:31:44，返回2020-11-16 16:31:44。
select dateadd(getdate(),-1,'dd');
--返回NULL。
select dateadd(date '2005-02-18', 1, null);
```

  - 示例2：用常量表示。

    在MaxCompute SQL中，DATETIME类型没有直接的常量表示方式，**错误命令示例**如下：

```
select dateadd(2005-03-30 00:00:00, -1, 'mm');
```

    如果需要描述DATETIME类型的常量，**正确命令示例**如下：

```
--将STRING类型常量显式转换为DATETIME类型，返回2005-02-28 00:00:00。
select dateadd(cast("2005-03-30 00:00:00" as datetime), -1, 'mm');
```

  - 示例3：输入参数为STRING类型。

```
--如下输入参数为STRING类型，但不符合DATETIME类型格式，会返回报错。
select dateadd('2021-08-27',1,'dd');
--如下输入参数为STRING类型，符合DATETIME类型格式，且MaxCompute项目的数据类型版本为1.0，返回2005-03-01 00:00:00。
set odps.sql.type.system.odps2=false;
select dateadd('2005-02-28 00:00:00', 1, 'dd');
```

- 表数据示例

  基于示例数据，对date1、datetime1和timestamp1列按照指定单位和幅度变化，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, dateadd(date1,1,'dd') as date1_dateadd, datetime1, dateadd(datetime1,1,'mm') as datetime1_dateadd, timestamp1,
dateadd(timestamp1,-1,'yyyy') as timestamp1_dateadd from mf_date_fun_t;
```

  返回结果如下。

```
+------------+--------------+---------------------+---------------------+-----------------------------+-----------------
-------------+
| date1      | date1_dateadd | datetime1           | datetime1_dateadd   | timestamp1                  | timestamp1_datead
d            |
+------------+--------------+---------------------+---------------------+-----------------------------+-----------------
-------------+
| 2021-11-29 | 2021-11-30    | 2021-11-29 00:01:00 | 2021-12-29 00:01:00 | 2021-01-11 00:00:00.123456789 | 2020-01-11 00:00:
00.123456789 |
| 2021-11-28 | 2021-11-29    | 2021-11-28 00:02:00 | 2021-12-28 00:02:00 | 2021-02-11 00:00:00.123456789 | 2020-02-11 00:00:
00.123456789 |
| 2021-11-27 | 2021-11-28    | 2021-11-27 00:03:00 | 2021-12-27 00:03:00 | 2021-03-11 00:00:00.123456789 | 2020-03-11 00:00:
00.123456789 |
| 2021-11-26 | 2021-11-27    | 2021-11-26 00:04:00 | 2021-12-26 00:04:00 | 2021-04-11 00:00:00.123456789 | 2020-04-11 00:00:
00.123456789 |
| 2021-11-25 | 2021-11-26    | 2021-11-25 00:05:00 | 2021-12-25 00:05:00 | 2021-05-11 00:00:00.123456789 | 2020-05-11 00:00:
00.123456789 |
| 2021-11-24 | 2021-11-25    | 2021-11-24 00:06:00 | 2021-12-24 00:06:00 | 2021-06-11 00:00:00.123456789 | 2020-06-11 00:00:
00.123456789 |
| 2021-11-23 | 2021-11-24    | 2021-11-23 00:07:00 | 2021-12-23 00:07:00 | 2021-07-11 00:00:00.123456789 | 2020-07-11 00:00:
00.123456789 |
| 2021-11-22 | 2021-11-23    | 2021-11-22 00:08:00 | 2021-12-22 00:08:00 | 2021-08-11 00:00:00.123456789 | 2020-08-11 00:00:
00.123456789 |
| 2021-11-21 | 2021-11-22    | 2021-11-21 00:09:00 | 2021-12-21 00:09:00 | 2021-09-11 00:00:00.123456789 | 2020-09-11 00:00:
00.123456789 |
| 2021-11-20 | 2021-11-21    | 2021-11-20 00:10:00 | 2021-12-20 00:10:00 | 2021-10-11 00:00:00.123456789 | 2020-10-11 00:00:
00.123456789 |
+------------+--------------+---------------------+---------------------+-----------------------------+-----------------
-------------+
```

DATE_ADD

- 命令格式

```
date date_add(date|timestamp|string <startdate>, bigint <delta>)
```

- 命令说明

  按照delta幅度增减startdate日期的天数。如果您需要获取在当前时间基础上指定变动幅度的日期，请结合GETDATE函数使用。

  本命令与DATE_SUB的增减逻辑相反。

- 参数说明
  - startdate：必填。起始日期值。支持DATE、DATETIME或STRING类型。

    如果参数为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATE类型后参与运算，且STRING参数格式至少要包含 `'yyyy-mm-dd'` 。例如 `'2019-12-27'` 。
  - delta：必填。修改幅度。BIGINT类型。如果delta大于0，则**增**；delta小于0，则**减**；delta等于0，不增不减。
- 返回值说明

  返回DATE类型，格式为 `yyyy-mm-dd` 。返回规则如下：
  - startdate非DATE、DATETIME或STRING类型时，返回报错。
  - startdate值为NULL时，返回报错。
  - delta值为NULL时，返回NULL。
- 示例

- 静态数据示例

```
--返回2005-03-01。加1天，结果超出当年2月份的最后1天，实际值为下个月的第1天。
select date_add(datetime '2005-02-28 00:00:00', 1);
--返回2005-02-27。减1天。
select date_add(date '2005-02-28', -1);
--返回2005-03-20。
set odps.sql.type.system.odps2=false;
select date_add('2005-02-28 00:00:00', 20);
--假设当前时间为2020-11-17 16:31:44，返回2020-11-16。
select date_add(getdate(),-1);
--返回NULL。
select date_add('2005-02-28 00:00:00', null);
```

- 表数据示例

  基于示例数据，对date1、datetime1和timestamp1列按照幅度变化天数，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, date_add(date1,1) as date1_date_add, datetime1, date_add(datetime1,-1) as datetime1_date_add, timestamp1, date
_add(timestamp1,0) as timestamp1_date_add from mf_date_fun_t;
```

  返回结果如下。

```
+------------+----------------+---------------------+-------------------+-------------------------------+-------------------
-+
| date1      | date1_date_add | datetime1           | datetime1_date_add| timestamp1                    | timestamp1_date_ad
d|
+------------+----------------+---------------------+-------------------+-------------------------------+-------------------
-+
| 2021-11-29 | 2021-11-30     | 2021-11-29 00:01:00 | 2021-11-28        | 2021-01-11 00:00:00.123456789 | 2021-01-11
|
| 2021-11-28 | 2021-11-29     | 2021-11-28 00:02:00 | 2021-11-27        | 2021-02-11 00:00:00.123456789 | 2021-02-11
|
| 2021-11-27 | 2021-11-28     | 2021-11-27 00:03:00 | 2021-11-26        | 2021-03-11 00:00:00.123456789 | 2021-03-11
|
| 2021-11-26 | 2021-11-27     | 2021-11-26 00:04:00 | 2021-11-25        | 2021-04-11 00:00:00.123456789 | 2021-04-11
|
| 2021-11-25 | 2021-11-26     | 2021-11-25 00:05:00 | 2021-11-24        | 2021-05-11 00:00:00.123456789 | 2021-05-11
|
| 2021-11-24 | 2021-11-25     | 2021-11-24 00:06:00 | 2021-11-23        | 2021-06-11 00:00:00.123456789 | 2021-06-11
|
| 2021-11-23 | 2021-11-24     | 2021-11-23 00:07:00 | 2021-11-22        | 2021-07-11 00:00:00.123456789 | 2021-07-11
|
| 2021-11-22 | 2021-11-23     | 2021-11-22 00:08:00 | 2021-11-21        | 2021-08-11 00:00:00.123456789 | 2021-08-11
|
| 2021-11-21 | 2021-11-22     | 2021-11-21 00:09:00 | 2021-11-20        | 2021-09-11 00:00:00.123456789 | 2021-09-11
|
| 2021-11-20 | 2021-11-21     | 2021-11-20 00:10:00 | 2021-11-19        | 2021-10-11 00:00:00.123456789 | 2021-10-11
|
+------------+----------------+---------------------+-------------------+-------------------------------+-------------------
-+
```

DATE_FORMAT

- 命令格式

```
string date_format(date|datetime|timestamp|string <date>, string <format>)
```

- 命令说明

  将date按照format指定的格式转换为字符串。

- 参数说明
  - date：必填。待转换的日期值。支持DATE、DATETIME、TIMESTAMP或STRING类型。
    - DATE、DATETIME和STRING类型只能在Hive兼容模式下使用，您可以执行 `set odps.sql.hive.compatible=true;` 命令打开Hive模式。
    - 如果参数为STRING类型，格式至少要包含 `'yyyy-MM-dd'` 。例如 `'2019-12-27'` 。

- format：必填。STRING类型常量。format可由如下日期字段组成，例如 `yyyy-MM-dd hh:mm:ss:SSS` 或 `yyyy-MM-dd hh:mi:ss:SSS`：
  - `YYYY` 或 `yyyy` 代表年份。
  - `MM` 代表月份。
  - `mm` 代表分钟。
  - `dd` 代表天。
  - `HH` 代表24小时制时。
  - `hh` 代表12小时制时。
  - `mi` 代表分钟。
  - `ss` 代表秒。
  - `SSS` 代表毫秒。

  > 说明
  > - 在非Hive兼容模式下，只能使用 `yyyy-MM-dd hh:mi:ss`，如果使用 `yyyy-MM-dd hh:mm:ss`，`mm` 部分会取月份值。
  > - 在Hive兼容模式下，只能使用 `yyyy-MM-dd hh:mm:ss`，如果使用 `yyyy-MM-dd hh:mi:ss` 会返回NULL。

- 返回值说明

  返回STRING类型。返回规则如下：
  - date非DATE、DATETIME或TIMESTAMP类型时，返回NULL。
  - date值为NULL时，返回报错。
  - format值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--打开Hive模式。与SQL语句一起提交。
set odps.sql.hive.compatible=true;
--假设当前时间为2022-04-24 15:49。返回2022-49-24 03:49:01.902。
select date_format(from_utc_timestamp(current_timestamp(), 'UTC'),'yyyy-mm-dd hh:mm:ss.SSS');
--返回2022-04-24。
select date_format('2022-04-24','yyyy-MM-dd');
```

  - 表数据示例

    基于示例数据，对datetime1和timestamp1转换为指定格式的字符串，命令示例如下。

```
--打开Hive模式并开启2.0新类型。与SQL语句一起提交。
set odps.sql.hive.compatible=true;
set odps.sql.type.system.odps2=true;
select datetime1, timestamp1, date_format(datetime1,'yyyy/mm/dd'),
       date_format(datetime1,'yyyy/mm/dd HH:mm:ss'),
       date_format(timestamp1,'yyyy/mm/dd HH:mm:ss')
from mf_date_fun_t;
```

    返回结果如下。

```
+------------+------------+-----+-----+-----+
| datetime1  | timestamp1 | _c2 | _c3 | _c4 |
+------------+------------+-----+-----+-----+
| 2021-11-29 00:01:00 | 2021-01-11 00:00:00.123456789 | 2021/01/29 | 2021/01/29 00:01:00 | 2021/00/11 00:00:00 |
| 2021-11-28 00:02:00 | 2021-02-11 00:00:00.123456789 | 2021/02/28 | 2021/02/28 00:02:00 | 2021/00/11 00:00:00 |
| 2021-11-27 00:03:00 | 2021-03-11 00:00:00.123456789 | 2021/03/27 | 2021/03/27 00:03:00 | 2021/00/11 00:00:00 |
| 2021-11-26 00:04:00 | 2021-04-11 00:00:00.123456789 | 2021/04/26 | 2021/04/26 00:04:00 | 2021/00/11 00:00:00 |
| 2021-11-25 00:05:00 | 2021-05-11 00:00:00.123456789 | 2021/05/25 | 2021/05/25 00:05:00 | 2021/00/11 00:00:00 |
| 2021-11-24 00:06:00 | 2021-06-11 00:00:00.123456789 | 2021/06/24 | 2021/06/24 00:06:00 | 2021/00/11 00:00:00 |
| 2021-11-23 00:07:00 | 2021-07-11 00:00:00.123456789 | 2021/07/23 | 2021/07/23 00:07:00 | 2021/00/11 00:00:00 |
| 2021-11-22 00:08:00 | 2021-08-11 00:00:00.123456789 | 2021/08/22 | 2021/08/22 00:08:00 | 2021/00/11 00:00:00 |
| 2021-11-21 00:09:00 | 2021-09-11 00:00:00.123456789 | 2021/09/21 | 2021/09/21 00:09:00 | 2021/00/11 00:00:00 |
| 2021-11-20 00:10:00 | 2021-10-11 00:00:00.123456789 | 2021/10/20 | 2021/10/20 00:10:00 | 2021/00/11 00:00:00 |
+------------+------------+-----+-----+-----+
```

DATE_SUB

- 命令格式

```
date date_sub(date|timestamp|string <startdate>, bigint <delta>)
```

- 命令说明

  按照delta幅度增减startdate日期的天数。如果您需要获取在当前时间基础上指定变动幅度的日期，请结合GETDATE函数使用。

本命令与DATE_ADD的增减逻辑相反。

- 参数说明
  - startdate：必填。起始日期值。支持DATE、DATETIME或STRING类型。

    如果参数为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATE类型后参与运算，且STRING参数格式至少要包含 `'yyyy-mm-dd'` 。例如 `'2019-12-27'` 。
  - delta：必填。修改幅度。BIGINT类型。如果delta大于0，则**减**；delta小于0，则**增**；delta等于0，不增不减。
- 返回值说明

  返回DATE类型，格式为 `yyyy-mm-dd` 。返回规则如下：
  - startdate非DATE、DATETIME或STRING类型时，返回报错。
  - startdate值为NULL时，返回报错。
  - delta值为NULL时，返回NULL。
- 示例
  - 静态数据示例

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
--返回2005-02-28。减1天，实际值为上个月的最后1天。
select date_sub(datetime '2005-03-01 00:00:00', 1);
--返回2005-03-01。增1天。
select date_sub(date '2005-02-28', -1);
--返回2005-02-27。减2天。
set odps.sql.type.system.odps2=false;
select date_sub('2005-03-01 00:00:00', 2);
--假设当前时间为2021-09-10 16:31:44，返回2021-09-09。
select date_sub(getdate(),1);
--返回NULL。
select date_sub('2005-03-01 00:00:00', null);
```

  - 表数据示例

    基于示例数据，对date1、datetime1和timestamp1列按照幅度变化天数，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, date_sub(date1,1) as date1_date_sub, datetime1, date_sub(datetime1,-1) as datetime1_date_sub, timestamp1, date_sub(timestamp1,0) as timestamp1_date_sub from mf_date_fun_t;
```

    返回结果如下。

```
+------------+----------------+---------------------+---------------------+-------------------------------+-------------------------+
| date1      | date1_date_sub | datetime1           | datetime1_date_sub | timestamp1                    | timestamp1_date_sub |
+------------+----------------+---------------------+---------------------+-------------------------------+-------------------------+
| 2021-11-29 | 2021-11-28     | 2021-11-29 00:01:00 | 2021-11-30         | 2021-01-11 00:00:00.123456789 | 2021-01-11
|
| 2021-11-28 | 2021-11-27     | 2021-11-28 00:02:00 | 2021-11-29         | 2021-02-11 00:00:00.123456789 | 2021-02-11
|
| 2021-11-27 | 2021-11-26     | 2021-11-27 00:03:00 | 2021-11-28         | 2021-03-11 00:00:00.123456789 | 2021-03-11
|
| 2021-11-26 | 2021-11-25     | 2021-11-26 00:04:00 | 2021-11-27         | 2021-04-11 00:00:00.123456789 | 2021-04-11
|
| 2021-11-25 | 2021-11-24     | 2021-11-25 00:05:00 | 2021-11-26         | 2021-05-11 00:00:00.123456789 | 2021-05-11
|
| 2021-11-24 | 2021-11-23     | 2021-11-24 00:06:00 | 2021-11-25         | 2021-06-11 00:00:00.123456789 | 2021-06-11
|
| 2021-11-23 | 2021-11-22     | 2021-11-23 00:07:00 | 2021-11-24         | 2021-07-11 00:00:00.123456789 | 2021-07-11
|
| 2021-11-22 | 2021-11-21     | 2021-11-22 00:08:00 | 2021-11-23         | 2021-08-11 00:00:00.123456789 | 2021-08-11
|
| 2021-11-21 | 2021-11-20     | 2021-11-21 00:09:00 | 2021-11-22         | 2021-09-11 00:00:00.123456789 | 2021-09-11
|
| 2021-11-20 | 2021-11-19     | 2021-11-20 00:10:00 | 2021-11-21         | 2021-10-11 00:00:00.123456789 | 2021-10-11
|
+------------+----------------+---------------------+---------------------+-------------------------------+-------------------------+
```

DATEDIFF

- 命令格式

```
bigint datediff(date|datetime|timestamp <date1>, date|datetime|timestamp <date2>, string <datepart>)
```

- 命令说明

  计算两个时间date1、date2的差值，将差值以指定的时间单位datepart表示。

- 参数说明
  - date1、date2：必填。DATE、DATETIME或TIMESTAMP类型。被减数和减数。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。
  - datepart：可选。时间单位，STRING类型常量。

    如果您打开了MaxCompute新数据类型开关，可以不填写datepart参数，默认日期格式为天。该字段也支持扩展的日期格式：年 `-year` 、月 `-month` 或 `-mon` 、日 `-day` 和小时 `-hour` 。

    > **说明** 系统计算时会按照datepart切掉低单位部分，然后再计算结果。

- 返回值说明

  返回BIGINT类型。返回规则如下：
  - date1、date2非DATE、DATETIME或TIMESTAMP类型时，返回报错。
  - 如果date1小于date2，返回值为负数。
  - date1或date2值为NULL时，返回NULL。
  - datepart值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--假设start=2005-12-31 23:59:59，end=2006-01-01 00:00:00，则：
    --返回1。
    select datediff(end, start, 'dd');
    --返回1。
    select datediff(end, start, 'mm');
    --返回1。
    select datediff(end, start, 'yyyy');
    --返回1。
    select datediff(end, start, 'hh');
    --返回1。
    select datediff(end, start, 'mi');
    --返回1。
    select datediff(end, start, 'ss');
--返回1800。
select datediff(datetime'2013-05-31 13:00:00', datetime'2013-05-31 12:30:00', 'ss');
--返回30。
set odps.sql.type.system.odps2=false;
select datediff('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'mi');
--返回11。
select datediff(date '2013-05-21', date '2013-05-10', 'dd');
--开启2.0新类型。此命令需要与SQL语句一起提交。返回1800。
set odps.sql.type.system.odps2=true;
select datediff(timestamp '2013-05-31 13:00:00', timestamp '2013-05-31 12:30:00', 'ss');
--假设start = 2018-06-04 19:33:23.234，end = 2018-06-04 19:33:23.250，含毫秒的日期不属于标准DATETIME式样，不能直接隐式转换，此处需进行显示转换。返回16。
select datediff(to_date('2018-06-04 19:33:23.250', 'yyyy-mm-dd hh:mi:ss.ff3'), to_date('2018-06-04 19:33:23.234', 'yyyy-mm-dd hh:mi:ss.ff3') , 'ff3');
--返回NULL。
select datediff(date '2013-05-21', date '2013-05-10', null);
--返回NULL。
select datediff(date '2013-05-21', null, 'dd');
```

- 表数据示例

  基于示例数据，计算date1和date2、datetime1和datetime2、timestamp1和timestamp2列的差值，并以指定单位表示，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, date2, datediff(date1,date2,'dd') as date1_date2_datediff, datetime1, datetime2, datediff(datetime1, datetime2
,'dd') as datetime1_datetime2_datediff, timestamp1, timestamp2, datediff(timestamp1, timestamp2,'mm') as timestamp1_timestam
p2_datediff from mf_date_fun_t;
```

  返回结果如下。

```
+------------+------------+---------------------+--------------------+--------------------+-----------------------------
+-------------------------------+-------------------------------+--------------------------------+
| date1      | date2      | date1_date2_datediff | datetime1          | datetime2          | datetime1_datetime2_datediff
| timestamp1                    | timestamp2                    | timestamp1_timestamp2_datediff |
+------------+------------+---------------------+--------------------+--------------------+-----------------------------
+-------------------------------+-------------------------------+--------------------------------+
| 2021-11-29 | 2021-10-29 | 31                  | 2021-11-29 00:01:00 | 2021-10-29 00:00:00 | 31
| 2021-01-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -9                             |
| 2021-11-28 | 2021-10-29 | 30                  | 2021-11-28 00:02:00 | 2021-10-29 00:00:00 | 30
| 2021-02-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -8                             |
| 2021-11-27 | 2021-10-29 | 29                  | 2021-11-27 00:03:00 | 2021-10-29 00:00:00 | 29
| 2021-03-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -7                             |
| 2021-11-26 | 2021-10-29 | 28                  | 2021-11-26 00:04:00 | 2021-10-29 00:00:00 | 28
| 2021-04-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -6                             |
| 2021-11-25 | 2021-10-29 | 27                  | 2021-11-25 00:05:00 | 2021-10-29 00:00:00 | 27
| 2021-05-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -5                             |
| 2021-11-24 | 2021-10-29 | 26                  | 2021-11-24 00:06:00 | 2021-10-29 00:00:00 | 26
| 2021-06-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -4                             |
| 2021-11-23 | 2021-10-29 | 25                  | 2021-11-23 00:07:00 | 2021-10-29 00:00:00 | 25
| 2021-07-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -3                             |
| 2021-11-22 | 2021-10-29 | 24                  | 2021-11-22 00:08:00 | 2021-10-29 00:00:00 | 24
| 2021-08-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -2                             |
| 2021-11-21 | 2021-10-29 | 23                  | 2021-11-21 00:09:00 | 2021-10-29 00:00:00 | 23
| 2021-09-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -1                             |
| 2021-11-20 | 2021-10-29 | 22                  | 2021-11-20 00:10:00 | 2021-10-29 00:00:00 | 22
| 2021-10-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | 0                              |
+------------+------------+---------------------+--------------------+--------------------+-----------------------------
+-------------------------------+-------------------------------+--------------------------------+
```

DATEPART

- 命令格式

```
bigint datepart(date|datetime|timestamp <date>, string <datepart>)
```

- 命令说明

  提取日期date中符合指定时间单位datepart的值。

- 参数说明
  - date：必填。DATE、DATETIME或TIMESTAMP类型。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。
  - datepart：必填。STRING类型常量，支持扩展的日期格式。

    此字段的取值遵循STRING与DATETIME类型转换的约定，即 `yyyy` 表示年， `mm` 表示月， `dd` 表示天。关于类型转换的规则，详情请参见数据类型转换。该字段也支持扩展的日期格式：年 `-year` 、月 `-month` 或 `-mon` 、日 `-day` 和小时 `-hour` 。

- 返回值说明

  返回BIGINT类型。返回规则如下：
  - date非DATE、DATETIME或TIMESTAMP类型时，返回报错。
  - date值为NULL时，返回报错。
  - datepart值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--返回2013。
select datepart(datetime'2013-06-08 01:10:00', 'yyyy');
--返回6。
select datepart(datetime'2013-06-08 01:10:00', 'mm');
--返回2013。
select datepart(date '2013-06-08', 'yyyy');
--开启2.0新类型。此命令需要与SQL语句一起提交。返回2013。
set odps.sql.type.system.odps2=true;
select datepart(timestamp '2013-06-08 01:10:00', 'yyyy');
--返回2013。
set odps.sql.type.system.odps2=false;
select datepart('2013-06-08 01:10:00', 'yyyy');
--返回NULL。
select datepart(date '2013-06-08', null);
```

- 表数据示例

  基于示例数据，提取date1、datetime1和timestamp1中符合指定时间单位的值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, datepart(date1,'yyyy') as date1_datepart, datetime1, datepart(datetime1,'dd') as datetime1_datepart, timestamp
1, datepart(timestamp1,'mm') as timestamp1_datepart from mf_date_fun_t;
```

  返回结果如下。

```
+------------+----------------+---------------------+-------------------------+-------------------------------+-------------
--------+
| date1      | date1_datepart | datetime1           | datetime1_datepart      | timestamp1                    | timestamp1_d
atepart |
+------------+----------------+---------------------+-------------------------+-------------------------------+-------------
--------+
| 2021-11-29 | 2021           | 2021-11-29 00:01:00 | 29                      | 2021-01-11 00:00:00.123456789 | 1
|
| 2021-11-28 | 2021           | 2021-11-28 00:02:00 | 28                      | 2021-02-11 00:00:00.123456789 | 2
|
| 2021-11-27 | 2021           | 2021-11-27 00:03:00 | 27                      | 2021-03-11 00:00:00.123456789 | 3
|
| 2021-11-26 | 2021           | 2021-11-26 00:04:00 | 26                      | 2021-04-11 00:00:00.123456789 | 4
|
| 2021-11-25 | 2021           | 2021-11-25 00:05:00 | 25                      | 2021-05-11 00:00:00.123456789 | 5
|
| 2021-11-24 | 2021           | 2021-11-24 00:06:00 | 24                      | 2021-06-11 00:00:00.123456789 | 6
|
| 2021-11-23 | 2021           | 2021-11-23 00:07:00 | 23                      | 2021-07-11 00:00:00.123456789 | 7
|
| 2021-11-22 | 2021           | 2021-11-22 00:08:00 | 22                      | 2021-08-11 00:00:00.123456789 | 8
|
| 2021-11-21 | 2021           | 2021-11-21 00:09:00 | 21                      | 2021-09-11 00:00:00.123456789 | 9
|
| 2021-11-20 | 2021           | 2021-11-20 00:10:00 | 20                      | 2021-10-11 00:00:00.123456789 | 10
|
+------------+----------------+---------------------+-------------------------+-------------------------------+-------------
--------+
```

DATETRUNC

- 命令格式

```
date|datetime datetrunc (date|datetime|timestamp <date>, string <datepart>)
```

- 命令说明

  返回将日期date按照datepart指定的时间单位进行截取后的日期值。

- 参数说明
  - date：必填。DATE、DATETIME或TIMESTAMP类型。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。
  - datepart：必填。STRING类型常量。支持扩展的日期格式。

    此字段的取值遵循STRING与DATETIME类型转换的约定，即 `yyyy` 表示年， `mm` 表示月， `dd` 表示天。关于类型转换的规则，详情请参见数据类型转换。该字段也支持扩展的日期格式：年 `-year` 、月 `-month` 或 `-mon` 、日 `-day` 和小时 `-hour` 。

- 返回值说明

返回DATE或DATETIME类型，格式为 `yyyy-mm-dd` 或 `yyyy-mm-dd hh:mi:ss` 。返回规则如下：

- date非DATE、DATETIME或TIMESTAMP类型时，返回报错。
- date值为NULL时，返回报错。
- datepart值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2011-01-01 00:00:00。
select datetrunc(datetime'2011-12-07 16:28:46', 'yyyy');
--返回2011-12-01 00:00:00。
select datetrunc(datetime'2011-12-07 16:28:46', 'month');
--返回2011-12-07 00:00:00。
select datetrunc(datetime'2011-12-07 16:28:46', 'DD');
--返回2011-01-01。
select datetrunc(date '2011-12-07', 'yyyy');
--开启2.0新类型。此命令需要与SQL语句一起提交。返回2011-01-01 00:00:00.0。
set odps.sql.type.system.odps2=true;
select datetrunc(timestamp '2011-12-07 16:28:46', 'yyyy');
--返回2011-01-01 00:00:00.0。
set odps.sql.type.system.odps2=false;
select datetrunc('2011-12-07 16:28:46', 'yyyy');
--返回NULL。
select datetrunc(date '2011-12-07', null);
```

  - 表数据示例

    基于示例数据，返回date1、datetime1和timestamp1按照指定时间单位截取后的值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, datetrunc(date1,'yyyy') as date1_datetrunc, datetime1, datetrunc(datetime1,'dd') as datetime1_datetrunc, times
tamp1, datetrunc(timestamp1,'mm') as timestamp1_datetrunc from mf_date_fun_t;
```

    返回结果如下。

```
+------------+-----------------+---------------------+---------------------+-------------------------------+----------------
------+
| date1      | date1_datetrunc | datetime1           | datetime1_datetrunc | timestamp1                    | timestamp1_date
trunc |
+------------+-----------------+---------------------+---------------------+-------------------------------+----------------
------+
| 2021-11-29 | 2021-01-01      | 2021-11-29 00:01:00 | 2021-11-29 00:00:00 | 2021-01-11 00:00:00.123456789 | 2021-01-01 00:0
0:00  |
| 2021-11-28 | 2021-01-01      | 2021-11-28 00:02:00 | 2021-11-28 00:00:00 | 2021-02-11 00:00:00.123456789 | 2021-02-01 00:0
0:00  |
| 2021-11-27 | 2021-01-01      | 2021-11-27 00:03:00 | 2021-11-27 00:00:00 | 2021-03-11 00:00:00.123456789 | 2021-03-01 00:0
0:00  |
| 2021-11-26 | 2021-01-01      | 2021-11-26 00:04:00 | 2021-11-26 00:00:00 | 2021-04-11 00:00:00.123456789 | 2021-04-01 00:0
0:00  |
| 2021-11-25 | 2021-01-01      | 2021-11-25 00:05:00 | 2021-11-25 00:00:00 | 2021-05-11 00:00:00.123456789 | 2021-05-01 00:0
0:00  |
| 2021-11-24 | 2021-01-01      | 2021-11-24 00:06:00 | 2021-11-24 00:00:00 | 2021-06-11 00:00:00.123456789 | 2021-06-01 00:0
0:00  |
| 2021-11-23 | 2021-01-01      | 2021-11-23 00:07:00 | 2021-11-23 00:00:00 | 2021-07-11 00:00:00.123456789 | 2021-07-01 00:0
0:00  |
| 2021-11-22 | 2021-01-01      | 2021-11-22 00:08:00 | 2021-11-22 00:00:00 | 2021-08-11 00:00:00.123456789 | 2021-08-01 00:0
0:00  |
| 2021-11-21 | 2021-01-01      | 2021-11-21 00:09:00 | 2021-11-21 00:00:00 | 2021-09-11 00:00:00.123456789 | 2021-09-01 00:0
0:00  |
| 2021-11-20 | 2021-01-01      | 2021-11-20 00:10:00 | 2021-11-20 00:00:00 | 2021-10-11 00:00:00.123456789 | 2021-10-01 00:0
0:00  |
+------------+-----------------+---------------------+---------------------+-------------------------------+----------------
------+
```

FROM_UNIXTIME

- 命令格式

```
datetime from_unixtime(bigint <unixtime>)
```

- 命令说明

  将数字型的UNIX时间日期值unixtime转为日期值。

- 参数说明

unixtime：必填。BIGINT类型，秒数，UNIX格式的日期时间值。如果输入为STRING、DOUBLE或DECIMAL类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为BIGINT后参与运算。

- 返回值说明

  返回DATETIME类型，格式为 `yyyy-mm-dd hh:mi:ss` 。unixtime值为NULL时，返回NULL。

  > **说明** Hive兼容模式下（ `set odps.sql.hive.compatible=true;` ），如果输入参数是STRING类型，返回值也是STRING类型。

- 示例
  - 静态数据示例

```
--返回1973-11-30 05:33:09。
select from_unixtime(123456789);
--返回1973-11-30 05:33:09。
set odps.sql.type.system.odps2=false;
select from_unixtime('123456789');
--返回NULL。
select from_unixtime(null);
```

  - 表数据示例

    基于示例数据，将date4转换为日期值，命令示例如下。

```
select date4, from_unixtime(date4) as date4_from_unixtime from mf_date_fun_t;
```

    返回结果如下。

```
+------------+---------------------+
| date4      | date4_from_unixtime |
+------------+---------------------+
| 123456780  | 1973-11-30 05:33:00 |
| 123456781  | 1973-11-30 05:33:01 |
| 123456782  | 1973-11-30 05:33:02 |
| 123456783  | 1973-11-30 05:33:03 |
| 123456784  | 1973-11-30 05:33:04 |
| 123456785  | 1973-11-30 05:33:05 |
| 123456786  | 1973-11-30 05:33:06 |
| 123456787  | 1973-11-30 05:33:07 |
| 123456788  | 1973-11-30 05:33:08 |
| 123456789  | 1973-11-30 05:33:09 |
+------------+---------------------+
```

GETDATE

- 命令格式

```
datetime getdate()
```

- 命令说明

  获取当前系统时间。使用东八区时间作为MaxCompute标准时间。

- 返回值说明

  返回当前日期和时间，DATETIME类型。

  > **说明** 在MaxCompute SQL中， `getdate` 总是返回一个固定的值。返回结果会是MaxCompute SQL执行期间的任意时间，时间精度精确到秒。如果开启了新数据类型2.0，时间精度精确到毫秒。

ISDATE

- 命令格式

```
boolean isdate(string <date>, string <format>)
```

- 命令说明

  判断一个日期字符串能否根据指定的格式串转换为一个日期值。如果能转换成功，返回True；否则返回False。

- 参数说明
  - date：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，会隐式转换为STRING类型后参与运算。
  - format：必填。STRING类型常量，不支持日期扩展格式。如果format中出现多余的格式串，则只取第一个格式串对应的日期数值，其余的会被视为分隔符。例如 `isdate("1234-yyyy", "yyyy-yyyy")` ，会返回True。

- 返回值说明

  返回BOOLEAN类型。date或format值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回True。
select isdate('2021-10-11','yyyy-mm-dd');
--返回false。
set odps.sql.type.system.odps2=false;
select isdate(1678952314,'yyyy-mm-dd');
```

  - 表数据示例

    基于示例数据，判断date3是否能转换为指定格式的日期值，命令示例如下。

```
select date3, isdate(date3,'yyyy-mm-dd') as date3_isdate from mf_date_fun_t;
```

    返回结果如下。

```
+------------+--------------+
| date3      | date3_isdate |
+------------+--------------+
| 2021-11-20 | true         |
| 2021-11-21 | true         |
| 2021-11-22 | true         |
| 2021-11-23 | true         |
| 2021-11-24 | true         |
| 2021-11-25 | true         |
| 2021-11-26 | true         |
| 2021-11-27 | true         |
| 2021-11-28 | true         |
| 2021-11-29 | true         |
+------------+--------------+
```

LASTDAY

- 命令格式

```
datetime lastday(datetime <date>)
```

- 命令说明

  取date所在月的最后一天，截取到天，时分秒部分为 `00:00:00` 。

- 参数说明

  date：DATETIME类型日期值，格式为 `yyyy-mm-dd hh:mi:ss` 。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。

- 返回值说明

  返回DATETIME类型，格式为 `yyyy-mm-dd hh:mi:ss` 。返回规则如下：

  - date非DATETIME或STRING类型，或格式不符合要求时，会返回报错。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2013-06-30 00:00:00。
select lastday (datetime '2013-06-08 01:10:00');
--返回2013-06-30 00:00:00。
set odps.sql.type.system.odps2=false;
select lastday ('2013-06-08 01:10:00');
--返回NULL。
select lastday (null);
```

- 表数据示例

  基于示例数据，返回datetime1列日期所在月的最后一天，命令示例如下。

```
select datetime1, lastday(datetime1) as datetime1_lastday from mf_date_fun_t;
```

  返回结果如下。

```
+---------------------+---------------------+
| datetime1           | datetime1_lastday   |
+---------------------+---------------------+
| 2021-11-29 00:01:00 | 2021-11-30 00:00:00 |
| 2021-11-28 00:02:00 | 2021-11-30 00:00:00 |
| 2021-11-27 00:03:00 | 2021-11-30 00:00:00 |
| 2021-11-26 00:04:00 | 2021-11-30 00:00:00 |
| 2021-11-25 00:05:00 | 2021-11-30 00:00:00 |
| 2021-11-24 00:06:00 | 2021-11-30 00:00:00 |
| 2021-11-23 00:07:00 | 2021-11-30 00:00:00 |
| 2021-11-22 00:08:00 | 2021-11-30 00:00:00 |
| 2021-11-21 00:09:00 | 2021-11-30 00:00:00 |
| 2021-11-20 00:10:00 | 2021-11-30 00:00:00 |
+---------------------+---------------------+
```

TO_DATE

- 命令格式

```
datetime to_date(string <date>, string <format>)
```

- 命令说明

  将date转换成符合format格式的日期值。

- 参数说明
  - date：必填。STRING类型，要转换的字符串格式的日期值。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - format：必填。STRING类型常量，日期格式。format不支持日期扩展格式，其他字符在解析时当作无用字符忽略。

    format：参数至少包含 `yyyy` ，否则会返回NULL。如果format中出现多余的格式串，则只取第一个格式串对应的日期数值，其余的会被视为分隔符。例如 `to_date("1234-2234", "yyyy-yyyy")` 会返回 `1234-01-01 00:00:00` 。

    format格式： `yyyy` 为4位数的年， `mm` 为2位数的月， `dd` 为2位数的日， `hh` 为24小时制的时， `mi` 为2位数的分钟， `ss` 为2位数秒， `ff3` 为3位精度毫秒。

- 返回值说明

  返回DATETIME类型，格式为 `yyyy-mm-dd hh:mi:ss` 。date或format值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2010-12-03 00:00:00。
select to_date('阿里巴巴2010-12*03', '阿里巴巴yyyy-mm*dd');
--返回2008-07-18 00:00:00。
select to_date('20080718', 'yyyymmdd');
--返回2008-07-18 20:30:00。
select to_date('200807182030','yyyymmddhhmi');
--'2008718'无法转为标准日期值，引发异常，应该为'20080718'。
select to_date('2008718', 'yyyymmdd');
--'阿里巴巴2010-12*3'无法转为标准日期值，引发异常，应该为'阿里巴巴2010-12*03'。
select to_date('阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd');
--'2010-24-01'无法转为标准日期值，引发异常，应该为'2010-01-24'。
select to_date('2010-24-01', 'yyyy');
--返回2018-10-30 15:13:12.
select to_date('20181030 15-13-12.345','yyyymmdd hh-mi-ss.ff3');
--返回NULL。
select to_date(null,'yyyymmdd hh-mi-ss.ff3');
--返回NULL。
select to_date('20181030 15-13-12.345',null);
```

- 表数据示例

  基于示例数据，将date3列日期转换为满足指定格式的日期值，命令示例如下。

```
select date3, to_date(date3, 'yyyy-mm-dd') as date3_to_date from mf_date_fun_t;
```

  返回结果如下。

```
+------------+---------------------+
| date3      | date3_to_date       |
+------------+---------------------+
| 2021-11-20 | 2021-11-20 00:00:00 |
| 2021-11-21 | 2021-11-21 00:00:00 |
| 2021-11-22 | 2021-11-22 00:00:00 |
| 2021-11-23 | 2021-11-23 00:00:00 |
| 2021-11-24 | 2021-11-24 00:00:00 |
| 2021-11-25 | 2021-11-25 00:00:00 |
| 2021-11-26 | 2021-11-26 00:00:00 |
| 2021-11-27 | 2021-11-27 00:00:00 |
| 2021-11-28 | 2021-11-28 00:00:00 |
| 2021-11-29 | 2021-11-29 00:00:00 |
+------------+---------------------+
```

TO_CHAR

- 命令格式

```
string to_char(datetime <date>, string <format>)
```

- 命令说明

  将日期类型date按照format指定的格式转成字符串。

- 参数说明
  - date：必填。DATETIME类型日期值，格式为 `yyyy-mm-dd hh:mi:ss` 。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，会隐式转换为DATETIME类型后参与运算。
  - format：必填。STRING类型常量。format中的日期格式部分会被替换成相应的数据，其他字符直接输出。

- 返回值说明

  返回STRING类型。返回规则如下：
  - date非DATETIME或STRING类型时，返回报错。
  - date值为NULL时，返回报错。
  - format值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回阿里金融2010-12*03。需要注意在Windows环境的MaxCompute客户端运行时返回结果中的中文字符显示存在问题。
select to_char(datetime'2010-12-03 00:00:00', '阿里金融yyyy-mm*dd');
--返回20080718。
select to_char(datetime'2008-07-18 00:00:00', 'yyyymmdd');
--返回20080718。
set odps.sql.type.system.odps2=false;
select to_char('2008-07-18 00:00:00', 'yyyymmdd');
--'阿里巴巴2010-12*3'无法转换为符合要求的格式，会引发异常，应该为'阿里巴巴2010-12*03'。
select to_char(datetime'阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd');
--'2010-24-01'不是标准日期值，引发异常，应该为'2010-01-24'。
select to_char(datetime'2010-24-01', 'yyyy');
--'2008718'不是标准日期值，引发异常，应该为'20080718'。
select to_char(datetime'2008718', 'yyyymmdd');
--返回NULL。
select to_char(datetime'2010-12-03 00:00:00', null);
```

- 表数据示例

  基于示例数据，将datetime1列日期转换为满足指定格式的字符串，命令示例如下。

```
select datetime1, to_char(datetime1, 'yyyy-mm-dd') as datetime1_to_char from mf_date_fun_t;
```

  返回结果如下。

```
+---------------------+-------------------+
| datetime1           | datetime1_to_char |
+---------------------+-------------------+
| 2021-11-29 00:01:00 | 2021-11-29        |
| 2021-11-28 00:02:00 | 2021-11-28        |
| 2021-11-27 00:03:00 | 2021-11-27        |
| 2021-11-26 00:04:00 | 2021-11-26        |
| 2021-11-25 00:05:00 | 2021-11-25        |
| 2021-11-24 00:06:00 | 2021-11-24        |
| 2021-11-23 00:07:00 | 2021-11-23        |
| 2021-11-22 00:08:00 | 2021-11-22        |
| 2021-11-21 00:09:00 | 2021-11-21        |
| 2021-11-20 00:10:00 | 2021-11-20        |
+---------------------+-------------------+
```

UNIX_TIMESTAMP

- 命令格式

```
bigint unix_timestamp(datetime|date|timestamp|string <date>)
```

- 命令说明

  将日期date转化为整型的UNIX格式的日期时间值。

- 参数说明

  date：必填。DATETIME、DATE、TIMESTAMP或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。当打开新数据类型属性时，隐式转换会失败，此时需要通过 `cast` 函数转换或关闭新数据类型，例如 `unix_timestamp(cast(... as datetime))` 。

- 返回值说明

  返回BIGINT类型，表示UNIX格式日期值。返回规则如下：

  - date非DATETIME、DATE、TIMESTAMP或STRING类型，或格式不符合要求时，返回报错。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回1237518660。
select unix_timestamp(datetime'2009-03-20 11:11:00');
--返回1237518660。
set odps.sql.type.system.odps2=false;
select unix_timestamp('2009-03-20 11:11:00');
--返回NULL。
select unix_timestamp(null);
```

- 表数据示例

  基于示例数据，将date1、datetime1和timestamp1列日期转换为整型的UNIX格式的日期值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, unix_timestamp(date1) as date1_unix_timestamp, datetime1, unix_timestamp(datetime1) as datetime1_unix_timestam
p, timestamp1, unix_timestamp(timestamp1) as timestamp1_unix_timestamp from mf_date_fun_t;
```

  返回结果如下。

```
+------------+---------------------+--------------------+-------------------------+------------------------------+-------
--------------------+
| date1      | date1_unix_timestamp | datetime1          | datetime1_unix_timestamp | timestamp1                   | timest
amp1_unix_timestamp |
+------------+---------------------+--------------------+-------------------------+------------------------------+-------
--------------------+
| 2021-11-29 | 1638115200           | 2021-11-29 00:01:00 | 1638115260              | 2021-01-11 00:00:00.123456789 | 16102
94400                |
| 2021-11-28 | 1638028800           | 2021-11-28 00:02:00 | 1638028920              | 2021-02-11 00:00:00.123456789 | 16129
72800                |
| 2021-11-27 | 1637942400           | 2021-11-27 00:03:00 | 1637942580              | 2021-03-11 00:00:00.123456789 | 16153
92000                |
| 2021-11-26 | 1637856000           | 2021-11-26 00:04:00 | 1637856240              | 2021-04-11 00:00:00.123456789 | 16180
70400                |
| 2021-11-25 | 1637769600           | 2021-11-25 00:05:00 | 1637769900              | 2021-05-11 00:00:00.123456789 | 16206
62400                |
| 2021-11-24 | 1637683200           | 2021-11-24 00:06:00 | 1637683560              | 2021-06-11 00:00:00.123456789 | 16233
40800                |
| 2021-11-23 | 1637596800           | 2021-11-23 00:07:00 | 1637597220              | 2021-07-11 00:00:00.123456789 | 16259
32800                |
| 2021-11-22 | 1637510400           | 2021-11-22 00:08:00 | 1637510880              | 2021-08-11 00:00:00.123456789 | 16286
11200                |
| 2021-11-21 | 1637424000           | 2021-11-21 00:09:00 | 1637424540              | 2021-09-11 00:00:00.123456789 | 16312
89600                |
| 2021-11-20 | 1637337600           | 2021-11-20 00:10:00 | 1637338200              | 2021-10-11 00:00:00.123456789 | 16338
81600                |
+------------+---------------------+--------------------+-------------------------+------------------------------+-------
--------------------+
```

WEEKDAY

- 命令格式

```
bigint weekday (datetime <date>)
```

- 命令说明

  返回date日期是当前周的第几天。

- 参数说明

  date：必填。DATETIME类型日期值。格式为 `yyyy-mm-dd hh:mi:ss` 。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - 周一作为一周的第一天，返回值为0。其他日期依次递增，周日返回6。
  - date非DATETIME或STRING类型，或格式不符合要求时，返回报错。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回4。
select weekday (datetime '2009-03-20 11:11:00');
--返回4。
set odps.sql.type.system.odps2=false;
select weekday ('2009-03-20 11:11:00');
--返回NULL。
select weekday (null);
```

- 表数据示例

  基于示例数据，计算datetime1列日期为所在周的第几天，命令示例如下。

```
select datetime1, weekday(datetime1) as datetime1_weekday from mf_date_fun_t;
```

  返回结果如下。

```
+---------------------+-------------------+
| datetime1           | datetime1_weekday |
+---------------------+-------------------+
| 2021-11-29 00:01:00 | 0                 |
| 2021-11-28 00:02:00 | 6                 |
| 2021-11-27 00:03:00 | 5                 |
| 2021-11-26 00:04:00 | 4                 |
| 2021-11-25 00:05:00 | 3                 |
| 2021-11-24 00:06:00 | 2                 |
| 2021-11-23 00:07:00 | 1                 |
| 2021-11-22 00:08:00 | 0                 |
| 2021-11-21 00:09:00 | 6                 |
| 2021-11-20 00:10:00 | 5                 |
+---------------------+-------------------+
```

WEEKOFYEAR

- 命令格式

```
bigint weekofyear (datetime <date>)
```

- 命令说明

  返回日期date位于那一年的第几周。周一作为一周的第一天。

  > **说明** 这一周算上一年还是下一年，取决于这一周的大多数日期（4天以上）在哪一年。算在前一年，就是前一年的最后一周；算在后一年就是后一年的第一周。

- 参数说明

  date：必填。DATETIME类型日期值。格式为 `yyyy-mm-dd hh:mi:ss` 。如果输入为STRING类型，且MaxCompute项目的数据类型版本是1.0，则会隐式转换为DATETIME类型后参与运算。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - date非DATETIME或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回1。虽然20141229属于2014年，但是这一周的大多数日期是在2015年，因此返回结果为1，表示是2015年的第一周。
select weekofyear(to_date("20141229", "yyyymmdd"));
--返回1。
select weekofyear(to_date("20141231", "yyyymmdd"));
--返回53。
select weekofyear(to_date("20151229", "yyyymmdd"));
--返回48。
set odps.sql.type.system.odps2=false;
select weekofyear('2021-11-29 00:01:00');
--返回NULL。
select weekofyear('20141231');
--返回NULL。
select weekofyear(null);
```

- 表数据示例

  基于示例数据，计算datetime1列日期为所在周的第几天，命令示例如下。

```
select datetime1, weekofyear(datetime1) as datetime1_weekofyear from mf_date_fun_t;
```

  返回结果如下。

```
+---------------------+----------------------+
| datetime1           | datetime1_weekofyear |
+---------------------+----------------------+
| 2021-11-29 00:01:00 | 48                   |
| 2021-11-28 00:02:00 | 47                   |
| 2021-11-27 00:03:00 | 47                   |
| 2021-11-26 00:04:00 | 47                   |
| 2021-11-25 00:05:00 | 47                   |
| 2021-11-24 00:06:00 | 47                   |
| 2021-11-23 00:07:00 | 47                   |
| 2021-11-22 00:08:00 | 47                   |
| 2021-11-21 00:09:00 | 46                   |
| 2021-11-20 00:10:00 | 46                   |
+---------------------+----------------------+
```

ADD_MONTHS

- 命令格式

```
string add_months(date|datetime|timestamp|string <startdate>, int <num_months>)
```

- 命令说明

  返回开始日期startdate增加num_months个月后的日期。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - startdate：必填。DATE、DATETIME、TIMESTAMP或STRING类型，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。
  - num_months：必填。INT型数值。

- 返回值说明

  返回STRING类型的日期值，格式为 `yyyy-mm-dd` 。返回规则如下：
  - startdate非DATE、DATETIME、TIMESTAMP或STRING类型，或格式不符合要求时，返回NULL。
  - startdate值为NULL时，返回报错。
  - num_months值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2017-05-14。
select add_months('2017-02-14',3);
--返回0017-05-14。
select add_months('17-2-14',3);
--返回2017-05-14。
select add_months('2017-02-14 21:30:00',3);
--返回NULL。
select add_months('20170214',3);
--返回NULL。
select add_months('2017-02-14 21:30:00',null);
```

- 表数据示例

  基于示例数据，将date1、datetime1、timestamp1和date3列日期转换为整型的UNIX格式的日期值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, add_months(date1,1) as date1_add_months, datetime1, add_months(datetime1, 2) as datetime1_add_months, timestam
p1, add_months(timestamp1,3) as timestamp1_add_months, date3, add_months(date3,4) as date3_add_months from mf_date_fun_t;
```

  返回结果如下。

```
+------------+------------------+---------------------+---------------------+-----------------------------+--------------
---------+------------+------------------+
| date1      | date1_add_months | datetime1           | datetime1_add_months | timestamp1                  | timestamp1_ad
d_months | date3      | date3_add_months |
+------------+------------------+---------------------+---------------------+-----------------------------+--------------
---------+------------+------------------+
| 2021-11-29 | 2021-12-29       | 2021-11-29 00:01:00 | 2022-01-29          | 2021-01-11 00:00:00.123456789 | 2021-04-11
| 2021-11-20 | 2022-03-20       |
| 2021-11-28 | 2021-12-28       | 2021-11-28 00:02:00 | 2022-01-28          | 2021-02-11 00:00:00.123456789 | 2021-05-11
| 2021-11-21 | 2022-03-21       |
| 2021-11-27 | 2021-12-27       | 2021-11-27 00:03:00 | 2022-01-27          | 2021-03-11 00:00:00.123456789 | 2021-06-11
| 2021-11-22 | 2022-03-22       |
| 2021-11-26 | 2021-12-26       | 2021-11-26 00:04:00 | 2022-01-26          | 2021-04-11 00:00:00.123456789 | 2021-07-11
| 2021-11-23 | 2022-03-23       |
| 2021-11-25 | 2021-12-25       | 2021-11-25 00:05:00 | 2022-01-25          | 2021-05-11 00:00:00.123456789 | 2021-08-11
| 2021-11-24 | 2022-03-24       |
| 2021-11-24 | 2021-12-24       | 2021-11-24 00:06:00 | 2022-01-24          | 2021-06-11 00:00:00.123456789 | 2021-09-11
| 2021-11-25 | 2022-03-25       |
| 2021-11-23 | 2021-12-23       | 2021-11-23 00:07:00 | 2022-01-23          | 2021-07-11 00:00:00.123456789 | 2021-10-11
| 2021-11-26 | 2022-03-26       |
| 2021-11-22 | 2021-12-22       | 2021-11-22 00:08:00 | 2022-01-22          | 2021-08-11 00:00:00.123456789 | 2021-11-11
| 2021-11-27 | 2022-03-27       |
| 2021-11-21 | 2021-12-21       | 2021-11-21 00:09:00 | 2022-01-21          | 2021-09-11 00:00:00.123456789 | 2021-12-11
| 2021-11-28 | 2022-03-28       |
| 2021-11-20 | 2021-12-20       | 2021-11-20 00:10:00 | 2022-01-20          | 2021-10-11 00:00:00.123456789 | 2022-01-11
| 2021-11-29 | 2022-03-29       |
+------------+------------------+---------------------+---------------------+-----------------------------+--------------
---------+------------+------------------+
```

CURRENT_TIMESTAMP

- 命令格式

```
timestamp current_timestamp()
```

- 命令说明

  返回当前TIMESTAMP类型的时间戳，值不固定。此函数为MaxCompute 2.0扩展函数。

- 返回值说明

  返回TIMESTAMP类型。

- 示例

```
--返回'2017-08-03 11:50:30.661'。
set odps.sql.type.system.odps2=true;
select current_timestamp();
```

DAY

- 命令格式

```
int day(datetime|timestamp|date|string <date>)
```

- 命令说明

  返回一个日期的天。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME、TIMESTAMP、DATE或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。

- 返回值说明

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP、DATE或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回1。
select day('2014-09-01');
--返回NULL。
select day('20140901');
--返回NULL。
select day(null);
```

  - 表数据示例

    基于示例数据，返回date1、datetime1、timestamp1和date3列日期对应的天，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, day(date1) as date1_day, datetime1, day(datetime1) as datetime1_day, timestamp1, day(timestamp1) as timestamp1
_day, date3, day(date3) as date3_day from mf_date_fun_t;
```

    返回结果如下。

```
+------------+-----------+---------------------+---------------+-----------------------------+----------------+-----------
-+-----------+
| date1      | date1_day | datetime1           | datetime1_day | timestamp1                  | timestamp1_day | date3
| date3_day |
+------------+-----------+---------------------+---------------+-----------------------------+----------------+-----------
-+-----------+
| 2021-11-29 | 29        | 2021-11-29 00:01:00 | 29            | 2021-01-11 00:00:00.123456789 | 11             | 2021-11-20
| 20        |
| 2021-11-28 | 28        | 2021-11-28 00:02:00 | 28            | 2021-02-11 00:00:00.123456789 | 11             | 2021-11-21
| 21        |
| 2021-11-27 | 27        | 2021-11-27 00:03:00 | 27            | 2021-03-11 00:00:00.123456789 | 11             | 2021-11-22
| 22        |
| 2021-11-26 | 26        | 2021-11-26 00:04:00 | 26            | 2021-04-11 00:00:00.123456789 | 11             | 2021-11-23
| 23        |
| 2021-11-25 | 25        | 2021-11-25 00:05:00 | 25            | 2021-05-11 00:00:00.123456789 | 11             | 2021-11-24
| 24        |
| 2021-11-24 | 24        | 2021-11-24 00:06:00 | 24            | 2021-06-11 00:00:00.123456789 | 11             | 2021-11-25
| 25        |
| 2021-11-23 | 23        | 2021-11-23 00:07:00 | 23            | 2021-07-11 00:00:00.123456789 | 11             | 2021-11-26
| 26        |
| 2021-11-22 | 22        | 2021-11-22 00:08:00 | 22            | 2021-08-11 00:00:00.123456789 | 11             | 2021-11-27
| 27        |
| 2021-11-21 | 21        | 2021-11-21 00:09:00 | 21            | 2021-09-11 00:00:00.123456789 | 11             | 2021-11-28
| 28        |
| 2021-11-20 | 20        | 2021-11-20 00:10:00 | 20            | 2021-10-11 00:00:00.123456789 | 11             | 2021-11-29
| 29        |
+------------+-----------+---------------------+---------------+-----------------------------+----------------+-----------
-+-----------+
```

DAYOFMONTH

- 命令格式

```
int dayofmonth(datetime|timestamp|date|string <date>)
```

- 命令说明

  返回日期日部分的值。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME、TIMESTAMP、DATE或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。

- 返回值说明

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP、DATE或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--返回1。
select dayofmonth('2014-09-01');
--返回NULL。
select dayofmonth('20140901');
--返回NULL。
select dayofmonth(null);
```

- 表数据示例

基于示例数据，返回date1、datetime1、timestamp1和date3列日期日部分的值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, dayofmonth(date1) as date1_dayofmonth, datetime1, dayofmonth(datetime1) as datetime1_dayofmonth, timestamp1, d
ayofmonth(timestamp1) as timestamp1_dayofmonth, date3, dayofmonth(date3) as date3_dayofmonth from mf_date_fun_t;
```

返回结果如下。

```
+------------+------------------+---------------------+----------------------+-------------------------------+--------------
---------+------------+------------------+
| date1      | date1_dayofmonth | datetime1           | datetime1_dayofmonth | timestamp1                    | timestamp1_da
yofmonth | date3      | date3_dayofmonth |
+------------+------------------+---------------------+----------------------+-------------------------------+--------------
---------+------------+------------------+
| 2021-11-29 | 29               | 2021-11-29 00:01:00 | 29                   | 2021-01-11 00:00:00.123456789 | 11
| 2021-11-20 | 20               |
| 2021-11-28 | 28               | 2021-11-28 00:02:00 | 28                   | 2021-02-11 00:00:00.123456789 | 11
| 2021-11-21 | 21               |
| 2021-11-27 | 27               | 2021-11-27 00:03:00 | 27                   | 2021-03-11 00:00:00.123456789 | 11
| 2021-11-22 | 22               |
| 2021-11-26 | 26               | 2021-11-26 00:04:00 | 26                   | 2021-04-11 00:00:00.123456789 | 11
| 2021-11-23 | 23               |
| 2021-11-25 | 25               | 2021-11-25 00:05:00 | 25                   | 2021-05-11 00:00:00.123456789 | 11
| 2021-11-24 | 24               |
| 2021-11-24 | 24               | 2021-11-24 00:06:00 | 24                   | 2021-06-11 00:00:00.123456789 | 11
| 2021-11-25 | 25               |
| 2021-11-23 | 23               | 2021-11-23 00:07:00 | 23                   | 2021-07-11 00:00:00.123456789 | 11
| 2021-11-26 | 26               |
| 2021-11-22 | 22               | 2021-11-22 00:08:00 | 22                   | 2021-08-11 00:00:00.123456789 | 11
| 2021-11-27 | 27               |
| 2021-11-21 | 21               | 2021-11-21 00:09:00 | 21                   | 2021-09-11 00:00:00.123456789 | 11
| 2021-11-28 | 28               |
| 2021-11-20 | 20               | 2021-11-20 00:10:00 | 20                   | 2021-10-11 00:00:00.123456789 | 11
| 2021-11-29 | 29               |
+------------+------------------+---------------------+----------------------+-------------------------------+--------------
---------+------------+------------------+
```

EXTRACT

- 命令格式

```
int extract(<datepart> from date|datetime|timestamp <date>)
```

- 命令说明

提取日期date中指定单位datepart的部分。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - datepart：必填。支持YEAR、MONTH、DAY、HOUR或MINUTE等时间取值。
  - date：必填。DATE、DATETIME、TIMESTAMP或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。
- 返回值

返回INT类型。返回规则如下：
  - datepart非YEAR、MONTH、DAY、HOUR或MINUTE等时间取值时，返回报错。
  - datepart值为NULL时，返回报错。
  - date非DATE、DATETIME、TIMESTAMP或STRING类型或为NULL时，返回NULL。
- 示例

- 静态数据示例

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select  extract(year from '2019-05-01 11:21:00') year
        ,extract(month from '2019-05-01 11:21:00') month
        ,extract(day from '2019-05-01 11:21:00') day
        ,extract(hour from '2019-05-01 11:21:00') hour
        ,extract(minute from '2019-05-01 11:21:00') minute;
--返回值如下。
+------+-------+------+------+--------+
| year | month | day  | hour | minute |
+------+-------+------+------+--------+
| 2019 | 5     | 1    | 11   | 21     |
+------+-------+------+------+--------+
--返回NULL。
select  extract(year from null);
```

- 表数据示例

  基于示例数据，提取timestamp1列的指定部分，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select timestamp1, extract(year from timestamp1) year, timestamp2, extract(month from timestamp2) month from mf_date_fun_t;
```

  返回结果如下。

```
+-------------------------------+------+-------------------------------+-------+
| timestamp1                    | year | timestamp2                    | month |
+-------------------------------+------+-------------------------------+-------+
| 2021-01-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-02-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-03-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-04-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-05-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-06-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-07-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-08-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-09-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
| 2021-10-11 00:00:00.123456789 | 2021 | 2021-10-11 00:00:00.123456789 | 10    |
+-------------------------------+------+-------------------------------+-------+
```

FROM_UTC_TIMESTAMP

- 命令格式

```
datetime|timestamp from_utc_timestamp({any primitive type}*, string <timezone>)
```

- 命令说明

  将一个UTC时区的时间戳转换成一个指定时区的时间戳。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - {any primitive type}*：必填。时间戳，支持TIMESTAMP、DATETIME、TINYINT、SMALLINT、INT或BIGINT数据类型。如果该参数为TINYINT、SMALLINT、INT或BIGINT数据类型，则单位为毫秒。
  - timezone：必填。指定需要转换的目标时区。

    > **说明** 您可直接在搜索引擎中查找相关时区列表详情。

- 返回值说明

  返回DATETIME或TIMESTAMP类型，格式为 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。返回规则如下：
  - {any primitive type}*非TIMESTAMP、DATETIME、TINYINT、SMALLINT、INT或BIGINT时，返回报错。
  - {any primitive type}*值为NULL时，返回报错。
  - timezone值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--输入参数为毫秒（ms），返回2017-08-01 04:24:00.0。
select from_utc_timestamp(1501557840000, 'PST');
--返回1970-01-30 08:00:00.0。
select from_utc_timestamp('1970-01-30 16:00:00','PST');
--返回1970-01-29 16:00:00.0。
select from_utc_timestamp('1970-01-30','PST');
--开启2.0新类型。此命令需要与SQL语句一起提交。返回2011-12-25 17:00:00.123。
set odps.sql.type.system.odps2=true;
select from_utc_timestamp(timestamp '2011-12-25 09:00:00.123456', 'Asia/Shanghai');
--开启2.0新类型。此命令需要与SQL语句一起提交。返回2011-12-25 01:55:00.0。
set odps.sql.type.system.odps2=true;
select from_utc_timestamp(timestamp '2011-12-25 06:55:00', 'America/Toronto');
--返回NULL。
select from_utc_timestamp('1970-01-30',null);
```

- 表数据示例

  基于示例数据，将datetime1和timestamp1列转换为指定时区的时间戳，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select datetime1, from_utc_timestamp(datetime1,'PST') pst, timestamp1, from_utc_timestamp(timestamp1,'Asia/Shanghai') asia f
rom mf_date_fun_t;
```

  返回结果如下。

```
+---------------------+---------------------+-----------------------------+-----------------------------+
| datetime1           | pst                 | timestamp1                  | asia                        |
+---------------------+---------------------+-----------------------------+-----------------------------+
| 2021-11-29 00:01:00 | 2021-11-28 16:01:00 | 2021-01-11 00:00:00.123456789 | 2021-01-11 08:00:00.123456789 |
| 2021-11-28 00:02:00 | 2021-11-27 16:02:00 | 2021-02-11 00:00:00.123456789 | 2021-02-11 08:00:00.123456789 |
| 2021-11-27 00:03:00 | 2021-11-26 16:03:00 | 2021-03-11 00:00:00.123456789 | 2021-03-11 08:00:00.123456789 |
| 2021-11-26 00:04:00 | 2021-11-25 16:04:00 | 2021-04-11 00:00:00.123456789 | 2021-04-11 08:00:00.123456789 |
| 2021-11-25 00:05:00 | 2021-11-24 16:05:00 | 2021-05-11 00:00:00.123456789 | 2021-05-11 08:00:00.123456789 |
| 2021-11-24 00:06:00 | 2021-11-23 16:06:00 | 2021-06-11 00:00:00.123456789 | 2021-06-11 08:00:00.123456789 |
| 2021-11-23 00:07:00 | 2021-11-22 16:07:00 | 2021-07-11 00:00:00.123456789 | 2021-07-11 08:00:00.123456789 |
| 2021-11-22 00:08:00 | 2021-11-21 16:08:00 | 2021-08-11 00:00:00.123456789 | 2021-08-11 08:00:00.123456789 |
| 2021-11-21 00:09:00 | 2021-11-20 16:09:00 | 2021-09-11 00:00:00.123456789 | 2021-09-11 08:00:00.123456789 |
| 2021-11-20 00:10:00 | 2021-11-19 16:10:00 | 2021-10-11 00:00:00.123456789 | 2021-10-11 08:00:00.123456789 |
+---------------------+---------------------+-----------------------------+-----------------------------+
```

HOUR

- 命令格式

```
int hour(datetime|timestamp|string <date>)
```

- 命令说明

  返回日期小时部分的值。

- 参数说明

  date：必填。DATETIME、TIMESTAMP或STRING类型日期值，格式为 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。此函数为MaxCompute 2.0扩展函数。

- 返回值说明

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回12。
select hour('2014-09-01 12:00:00');
--返回12。
select hour('12:00:00');
--返回NULL。
select hour('20140901120000');
--返回NULL。
select hour(null);
```

- 表数据示例

  基于示例数据，返回datetime1和timestamp1小时部分的值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select datetime1, hour(datetime1) as datetime1_hour, timestamp1, hour(timestamp1) as timestamp1_hour from mf_date_fun_t;
```

  返回结果如下。

```
+---------------------+----------------+-------------------------------+-----------------+
| datetime1           | datetime1_hour | timestamp1                    | timestamp1_hour |
+---------------------+----------------+-------------------------------+-----------------+
| 2021-11-29 00:01:00 | 0              | 2021-01-11 00:00:00.123456789 | 0               |
| 2021-11-28 00:02:00 | 0              | 2021-02-11 00:00:00.123456789 | 0               |
| 2021-11-27 00:03:00 | 0              | 2021-03-11 00:00:00.123456789 | 0               |
| 2021-11-26 00:04:00 | 0              | 2021-04-11 00:00:00.123456789 | 0               |
| 2021-11-25 00:05:00 | 0              | 2021-05-11 00:00:00.123456789 | 0               |
| 2021-11-24 00:06:00 | 0              | 2021-06-11 00:00:00.123456789 | 0               |
| 2021-11-23 00:07:00 | 0              | 2021-07-11 00:00:00.123456789 | 0               |
| 2021-11-22 00:08:00 | 0              | 2021-08-11 00:00:00.123456789 | 0               |
| 2021-11-21 00:09:00 | 0              | 2021-09-11 00:00:00.123456789 | 0               |
| 2021-11-20 00:10:00 | 0              | 2021-10-11 00:00:00.123456789 | 0               |
+---------------------+----------------+-------------------------------+-----------------+
```

LAST_DAY

- 命令格式

```
string last_day(date|datetime|timestamp|string <date>)
```

- 命令说明

  返回该日期所在月份的最后一天日期。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATE、DATETIME、TIMESTAMP或STRING类型日期值。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。

- 返回值说明

  返回STRING类型的日期值，格式为 `yyyy-mm-dd` 。返回规则如下：

  - date非DATE、DATETIME、TIMESTAMP或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回报错。

- 示例

  - 静态数据示例

```
--返回2017-03-31。
select last_day('2017-03-04');
--返回2017-07-31。
select last_day('2017-07-04 11:40:00');
--返回NULL。
select last_day('20170304');
```

- 表数据示例

  基于示例数据，返回date1、datetime1、timestamp1和date3列日期所在月的最后一天，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, last_day(date1) as date1_lastday, datetime1, last_day(datetime1) as datetime1_lastday, timestamp1, last_day(ti
mestamp1) as timestamp1_lastday, date3, last_day(date3) as date3_lastday from mf_date_fun_t;
```

  返回结果如下。

```
+------------+---------------+---------------------+------------------+-----------------------------+--------------------
+------------+---------------+
| date1      | date1_lastday | datetime1           | datetime1_lastday | timestamp1                  | timestamp1_lastday
| date3      | date3_lastday |
+------------+---------------+---------------------+------------------+-----------------------------+--------------------
+------------+---------------+
| 2021-11-29 | 2021-11-30    | 2021-11-29 00:01:00 | 2021-11-30        | 2021-01-11 00:00:00.123456789 | 2021-01-31
| 2021-11-20 | 2021-11-30    |
| 2021-11-28 | 2021-11-30    | 2021-11-28 00:02:00 | 2021-11-30        | 2021-02-11 00:00:00.123456789 | 2021-02-28
| 2021-11-21 | 2021-11-30    |
| 2021-11-27 | 2021-11-30    | 2021-11-27 00:03:00 | 2021-11-30        | 2021-03-11 00:00:00.123456789 | 2021-03-31
| 2021-11-22 | 2021-11-30    |
| 2021-11-26 | 2021-11-30    | 2021-11-26 00:04:00 | 2021-11-30        | 2021-04-11 00:00:00.123456789 | 2021-04-30
| 2021-11-23 | 2021-11-30    |
| 2021-11-25 | 2021-11-30    | 2021-11-25 00:05:00 | 2021-11-30        | 2021-05-11 00:00:00.123456789 | 2021-05-31
| 2021-11-24 | 2021-11-30    |
| 2021-11-24 | 2021-11-30    | 2021-11-24 00:06:00 | 2021-11-30        | 2021-06-11 00:00:00.123456789 | 2021-06-30
| 2021-11-25 | 2021-11-30    |
| 2021-11-23 | 2021-11-30    | 2021-11-23 00:07:00 | 2021-11-30        | 2021-07-11 00:00:00.123456789 | 2021-07-31
| 2021-11-26 | 2021-11-30    |
| 2021-11-22 | 2021-11-30    | 2021-11-22 00:08:00 | 2021-11-30        | 2021-08-11 00:00:00.123456789 | 2021-08-31
| 2021-11-27 | 2021-11-30    |
| 2021-11-21 | 2021-11-30    | 2021-11-21 00:09:00 | 2021-11-30        | 2021-09-11 00:00:00.123456789 | 2021-09-30
| 2021-11-28 | 2021-11-30    |
| 2021-11-20 | 2021-11-30    | 2021-11-20 00:10:00 | 2021-11-30        | 2021-10-11 00:00:00.123456789 | 2021-10-31
| 2021-11-29 | 2021-11-30    |
+------------+---------------+---------------------+------------------+-----------------------------+--------------------
+------------+---------------+
```

MINUTE

- 命令格式

```
int minute(datetime|timestamp|string <date>)
```

- 命令说明

  返回日期分钟部分的值。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME、TIMESTAMP或STRING类型日期值，格式为 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。

- 返回值说明

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回30。
select minute('2014-09-01 12:30:00');
--返回30。
select minute('12:30:00');
--返回NULL。
select minute('20140901120000');
--返回NULL。
select minute(null);
```

- 表数据示例

  基于示例数据，返回datetime1和timestamp1分钟部分的值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select datetime1, minute(datetime1) as datetime1_minute, timestamp1, minute(timestamp1) as timestamp1_minute from mf_date_fu
n_t;
```

  返回结果如下。

```
+---------------------+-----------------+-----------------------------+-------------------+
| datetime1           | datetime1_minute | timestamp1                  | timestamp1_minute |
+---------------------+-----------------+-----------------------------+-------------------+
| 2021-11-29 00:01:00 | 1               | 2021-01-11 00:00:00.123456789 | 0                 |
| 2021-11-28 00:02:00 | 2               | 2021-02-11 00:00:00.123456789 | 0                 |
| 2021-11-27 00:03:00 | 3               | 2021-03-11 00:00:00.123456789 | 0                 |
| 2021-11-26 00:04:00 | 4               | 2021-04-11 00:00:00.123456789 | 0                 |
| 2021-11-25 00:05:00 | 5               | 2021-05-11 00:00:00.123456789 | 0                 |
| 2021-11-24 00:06:00 | 6               | 2021-06-11 00:00:00.123456789 | 0                 |
| 2021-11-23 00:07:00 | 7               | 2021-07-11 00:00:00.123456789 | 0                 |
| 2021-11-22 00:08:00 | 8               | 2021-08-11 00:00:00.123456789 | 0                 |
| 2021-11-21 00:09:00 | 9               | 2021-09-11 00:00:00.123456789 | 0                 |
| 2021-11-20 00:10:00 | 10              | 2021-10-11 00:00:00.123456789 | 0                 |
+---------------------+-----------------+-----------------------------+-------------------+
```

MONTH

- 命令格式

```
int month(datetime|timestamp|date|string <date>)
```

- 命令说明

  返回一个日期的月份。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME、TIMESTAMP、DATE或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。

- 返回值说明

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP、DATE或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回9。
select month('2014-09-01');
--返回NULL。
select month('20140901');
--返回NULL。
select month(null);
```

- 表数据示例

  基于示例数据，返回date1、datetime1、timestamp1和date3列日期所属月份，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, month(date1) as date1_month, datetime1, month(datetime1) as datetime1_month, timestamp1, month(timestamp1) as
timestamp1_month, date3, month(date3) as date3_month from mf_date_fun_t;
```

  返回结果如下。

```
+------------+-------------+---------------------+-----------------+-----------------------------+------------------+-----
-------+-------------+
| date1      | date1_month | datetime1           | datetime1_month | timestamp1                  | timestamp1_month | date
3      | date3_month |
+------------+-------------+---------------------+-----------------+-----------------------------+------------------+-----
-------+-------------+
| 2021-11-29 | 11          | 2021-11-29 00:01:00 | 11              | 2021-01-11 00:00:00.123456789 | 1                | 2021
-11-20 | 11          |
| 2021-11-28 | 11          | 2021-11-28 00:02:00 | 11              | 2021-02-11 00:00:00.123456789 | 2                | 2021
-11-21 | 11          |
| 2021-11-27 | 11          | 2021-11-27 00:03:00 | 11              | 2021-03-11 00:00:00.123456789 | 3                | 2021
-11-22 | 11          |
| 2021-11-26 | 11          | 2021-11-26 00:04:00 | 11              | 2021-04-11 00:00:00.123456789 | 4                | 2021
-11-23 | 11          |
| 2021-11-25 | 11          | 2021-11-25 00:05:00 | 11              | 2021-05-11 00:00:00.123456789 | 5                | 2021
-11-24 | 11          |
| 2021-11-24 | 11          | 2021-11-24 00:06:00 | 11              | 2021-06-11 00:00:00.123456789 | 6                | 2021
-11-25 | 11          |
| 2021-11-23 | 11          | 2021-11-23 00:07:00 | 11              | 2021-07-11 00:00:00.123456789 | 7                | 2021
-11-26 | 11          |
| 2021-11-22 | 11          | 2021-11-22 00:08:00 | 11              | 2021-08-11 00:00:00.123456789 | 8                | 2021
-11-27 | 11          |
| 2021-11-21 | 11          | 2021-11-21 00:09:00 | 11              | 2021-09-11 00:00:00.123456789 | 9                | 2021
-11-28 | 11          |
| 2021-11-20 | 11          | 2021-11-20 00:10:00 | 11              | 2021-10-11 00:00:00.123456789 | 10               | 2021
-11-29 | 11          |
+------------+-------------+---------------------+-----------------+-----------------------------+------------------+-----
-------+-------------+
```

MONTHS_BETWEEN

- 命令格式

```
double months_between(datetime|timestamp|date|string <date1>, datetime|timestamp|date|string <date2>)
```

- 命令说明

  返回日期date1和date2之间的月数。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date1、date2：必填。DATETIME、TIMESTAMP、DATE或STRING类型，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。

- 返回值说明

  返回DOUBLE类型。返回规则如下：

  - 当date1晚于date2时，返回值为正。当date2晚于date1时，返回值为负。
  - 当date1和date2分别对应两个月的最后一天，返回整数月；否则计算方式为date1减去date2的天数除以31天。
  - date1或date2值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回3.9495967741935485。
select months_between('1997-02-28 10:30:00', '1996-10-30');
--返回-3.9495967741935485。
select months_between('1996-10-30','1997-02-28 10:30:00' );
--返回-3.0。
select months_between('1996-09-30','1996-12-31');
--返回NULL。
select months_between('1996-09-30',null);
```

- 表数据示例

  基于示例数据，计算timestamp1和timestamp2列之间的月数，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select timestamp1, timestamp2, months_between(timestamp1, timestamp2) from mf_date_fun_t;
```

  返回结果如下。

```
+-------------------------------+-------------------------------+------------+
| timestamp1                    | timestamp2                    | _c2        |
+-------------------------------+-------------------------------+------------+
| 2021-01-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -9.0       |
| 2021-02-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -8.0       |
| 2021-03-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -7.0       |
| 2021-04-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -6.0       |
| 2021-05-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -5.0       |
| 2021-06-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -4.0       |
| 2021-07-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -3.0       |
| 2021-08-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -2.0       |
| 2021-09-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | -1.0       |
| 2021-10-11 00:00:00.123456789 | 2021-10-11 00:00:00.123456789 | 0.0        |
+------------+------------+------------+
```

NEXT_DAY

- 命令格式

```
string next_day(timestamp|date|datetime|string <startdate>, string <week>)
```

- 命令说明

  返回大于指定日期startdate并且与week相匹配的第一个日期，即下周几的具体日期。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - startdate：必填。TIMESTAMP、DATE、DATETIME或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。
  - week：必填。STRING类型，一个星期前2个或3个字母，或者一个星期的全名。例如MO、TUE或FRIDAY。

- 返回值说明

  返回STRING类型的日期值，格式为 `yyyy-mm-dd` 。返回规则如下：
  - date非TIMESTAMP、DATE、DATETIME或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回报错。
  - week值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2017-08-08。
select next_day('2017-08-01','TU');
--返回2017-08-08。
select next_day('2017-08-01 23:34:00','TU');
--返回NULL。
select next_day('20170801','TU');
--返回NULL。
select next_day('2017-08-01 23:34:00',null);
```

- 表数据示例

  基于示例数据，返回date1、datetime1、timestamp1和date3列对应下周的具体日期，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, next_day(date1,'MON') as date1_next_day, datetime1, next_day(datetime1,'TUE') as datetime1_next_day, timestamp
1, next_day(timestamp1,'WED') as timestamp1_next_day, date3, next_day(date3,'THU') as date3_next_day from mf_date_fun_t;
```

  返回结果如下。

```
+------------+----------------+---------------------+--------------------+-----------------------------+------------------
---+------------+----------------+
| date1      | date1_next_day | datetime1           | datetime1_next_day | timestamp1                  | timestamp1_next_d
ay | date3      | date3_next_day |
+------------+----------------+---------------------+--------------------+-----------------------------+------------------
---+------------+----------------+
| 2021-11-29 | 2021-12-06     | 2021-11-29 00:01:00 | 2021-11-30         | 2021-01-11 00:00:00.123456789 | 2021-01-13
| 2021-11-20 | 2021-11-25     |
| 2021-11-28 | 2021-11-29     | 2021-11-28 00:02:00 | 2021-11-30         | 2021-02-11 00:00:00.123456789 | 2021-02-17
| 2021-11-21 | 2021-11-25     |
| 2021-11-27 | 2021-11-29     | 2021-11-27 00:03:00 | 2021-11-30         | 2021-03-11 00:00:00.123456789 | 2021-03-17
| 2021-11-22 | 2021-11-25     |
| 2021-11-26 | 2021-11-29     | 2021-11-26 00:04:00 | 2021-11-30         | 2021-04-11 00:00:00.123456789 | 2021-04-14
| 2021-11-23 | 2021-11-25     |
| 2021-11-25 | 2021-11-29     | 2021-11-25 00:05:00 | 2021-11-30         | 2021-05-11 00:00:00.123456789 | 2021-05-12
| 2021-11-24 | 2021-11-25     |
| 2021-11-24 | 2021-11-29     | 2021-11-24 00:06:00 | 2021-11-30         | 2021-06-11 00:00:00.123456789 | 2021-06-16
| 2021-11-25 | 2021-12-02     |
| 2021-11-23 | 2021-11-29     | 2021-11-23 00:07:00 | 2021-11-30         | 2021-07-11 00:00:00.123456789 | 2021-07-14
| 2021-11-26 | 2021-12-02     |
| 2021-11-22 | 2021-11-29     | 2021-11-22 00:08:00 | 2021-11-23         | 2021-08-11 00:00:00.123456789 | 2021-08-18
| 2021-11-27 | 2021-12-02     |
| 2021-11-21 | 2021-11-22     | 2021-11-21 00:09:00 | 2021-11-23         | 2021-09-11 00:00:00.123456789 | 2021-09-15
| 2021-11-28 | 2021-12-02     |
| 2021-11-20 | 2021-11-22     | 2021-11-20 00:10:00 | 2021-11-23         | 2021-10-11 00:00:00.123456789 | 2021-10-13
| 2021-11-29 | 2021-12-02     |
+------------+----------------+---------------------+--------------------+-----------------------------+------------------
---+------------+----------------+
```

QUARTER

- 命令格式

```
int quarter (datetime|timestamp|date|string <date>)
```

- 命令说明

  返回一个日期的季度，范围是1~4。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME、TIMESTAMP、DATE或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。

- 返回值说明

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP、DATE或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回4。
select quarter('1970-11-12 10:00:00');
--返回4。
select quarter('1970-11-12');
--返回NULL。
select quarter(null);
```

- 表数据示例

  基于示例数据，返回date1、datetime1、timestamp1和date3列日期所属季度，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, quarter(date1) as date1_quarter, datetime1, quarter(datetime1) as datetime1_quarter, timestamp1, quarter(times
tamp1) as timestamp1_quarter, date3, quarter(date3) as date3_quarter from mf_date_fun_t;
```

  返回结果如下。

```
+------------+---------------+---------------------+------------------+-----------------------------+--------------------
+------------+---------------+
| date1      | date1_quarter | datetime1           | datetime1_quarter | timestamp1                  | timestamp1_quarter
| date3      | date3_quarter |
+------------+---------------+---------------------+------------------+-----------------------------+--------------------
+------------+---------------+
| 2021-11-29 | 4             | 2021-11-29 00:01:00 | 4                 | 2021-01-11 00:00:00.123456789 | 1
| 2021-11-20 | 4             |
| 2021-11-28 | 4             | 2021-11-28 00:02:00 | 4                 | 2021-02-11 00:00:00.123456789 | 1
| 2021-11-21 | 4             |
| 2021-11-27 | 4             | 2021-11-27 00:03:00 | 4                 | 2021-03-11 00:00:00.123456789 | 1
| 2021-11-22 | 4             |
| 2021-11-26 | 4             | 2021-11-26 00:04:00 | 4                 | 2021-04-11 00:00:00.123456789 | 2
| 2021-11-23 | 4             |
| 2021-11-25 | 4             | 2021-11-25 00:05:00 | 4                 | 2021-05-11 00:00:00.123456789 | 2
| 2021-11-24 | 4             |
| 2021-11-24 | 4             | 2021-11-24 00:06:00 | 4                 | 2021-06-11 00:00:00.123456789 | 2
| 2021-11-25 | 4             |
| 2021-11-23 | 4             | 2021-11-23 00:07:00 | 4                 | 2021-07-11 00:00:00.123456789 | 3
| 2021-11-26 | 4             |
| 2021-11-22 | 4             | 2021-11-22 00:08:00 | 4                 | 2021-08-11 00:00:00.123456789 | 3
| 2021-11-27 | 4             |
| 2021-11-21 | 4             | 2021-11-21 00:09:00 | 4                 | 2021-09-11 00:00:00.123456789 | 3
| 2021-11-28 | 4             |
| 2021-11-20 | 4             | 2021-11-20 00:10:00 | 4                 | 2021-10-11 00:00:00.123456789 | 4
| 2021-11-29 | 4             |
+------------+---------------+---------------------+------------------+-----------------------------+--------------------
+------------+---------------+
```

SECOND

- 命令格式

```
int second(datetime|timestamp|string <date>)
```

- 命令说明

  返回日期秒数部分的值。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME、TIMESTAMP或STRING类型日期值，格式为 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。

- 返回值说明

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回45。
select second('2014-09-01 12:30:45');
--返回45。
select second('12:30:45');
--返回NULL。
select second('20140901123045');
--返回NULL。
select second(null);
```

- 表数据示例

  基于示例数据，返回datetime1和timestamp1列秒数部分的值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
elect datetime1, second(datetime1) as datetime1_second, timestamp1, second(timestamp1) as timestamp1_second from mf_date_fun
_t;
```

  返回结果如下。

```
+---------------------+-----------------+-----------------------------+-------------------+
| datetime1           | datetime1_second | timestamp1                  | timestamp1_second |
+---------------------+-----------------+-----------------------------+-------------------+
| 2021-11-29 00:01:00 | 0               | 2021-01-11 00:00:00.123456789 | 0                 |
| 2021-11-28 00:02:00 | 0               | 2021-02-11 00:00:00.123456789 | 0                 |
| 2021-11-27 00:03:00 | 0               | 2021-03-11 00:00:00.123456789 | 0                 |
| 2021-11-26 00:04:00 | 0               | 2021-04-11 00:00:00.123456789 | 0                 |
| 2021-11-25 00:05:00 | 0               | 2021-05-11 00:00:00.123456789 | 0                 |
| 2021-11-24 00:06:00 | 0               | 2021-06-11 00:00:00.123456789 | 0                 |
| 2021-11-23 00:07:00 | 0               | 2021-07-11 00:00:00.123456789 | 0                 |
| 2021-11-22 00:08:00 | 0               | 2021-08-11 00:00:00.123456789 | 0                 |
| 2021-11-21 00:09:00 | 0               | 2021-09-11 00:00:00.123456789 | 0                 |
| 2021-11-20 00:10:00 | 0               | 2021-10-11 00:00:00.123456789 | 0                 |
+---------------------+-----------------+-----------------------------+-------------------+
```

TO_MILLIS

- 命令格式

```
bigint to_millis(datetime|timestamp <date>);
```

- 命令说明

  将给定日期date转换为以毫秒为单位的UNIX时间戳。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME或TIMESTAMP类型日期值，格式为 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - date非DATETIME或TIMESTAMP类型时，返回报错。
  - date值为NULL时，返回报错。

- 示例
  - 静态数据示例

```
--返回1617174900000。
select to_millis(datetime '2021-03-31 15:15:00');
--返回1617174900000。
set odps.sql.type.system.odps2=true;
select to_millis(timestamp '2021-03-31 15:15:00');
```

- 表数据示例

  基于示例数据，将datetime1和timestamp1转换为以毫秒为单位的UNIX时间戳，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select datetime1, to_millis(datetime1) as datetime1_to_millis, timestamp1, to_millis(timestamp1) as timestamp1_to_millis fro
m mf_date_fun_t;
```

  返回结果如下。

```
+---------------------+---------------------+-----------------------------+----------------------+
| datetime1           | datetime1_to_millis | timestamp1                  | timestamp1_to_millis |
+---------------------+---------------------+-----------------------------+----------------------+
| 2021-11-29 00:01:00 | 1638115260000       | 2021-01-11 00:00:00.123456789 | 1610294400123        |
| 2021-11-28 00:02:00 | 1638028920000       | 2021-02-11 00:00:00.123456789 | 1612972800123        |
| 2021-11-27 00:03:00 | 1637942580000       | 2021-03-11 00:00:00.123456789 | 1615392000123        |
| 2021-11-26 00:04:00 | 1637856240000       | 2021-04-11 00:00:00.123456789 | 1618070400123        |
| 2021-11-25 00:05:00 | 1637769900000       | 2021-05-11 00:00:00.123456789 | 1620662400123        |
| 2021-11-24 00:06:00 | 1637683560000       | 2021-06-11 00:00:00.123456789 | 1623340800123        |
| 2021-11-23 00:07:00 | 1637597220000       | 2021-07-11 00:00:00.123456789 | 1625932800123        |
| 2021-11-22 00:08:00 | 1637510880000       | 2021-08-11 00:00:00.123456789 | 1628611200123        |
| 2021-11-21 00:09:00 | 1637424540000       | 2021-09-11 00:00:00.123456789 | 1631289600123        |
| 2021-11-20 00:10:00 | 1637338200000       | 2021-10-11 00:00:00.123456789 | 1633881600123        |
+---------------------+---------------------+-----------------------------+----------------------+
```

YEAR

- 命令格式

```
int year(datetime|timestamp|date|string <date>)
```

- 命令说明

  返回日期date的年。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  date：必填。DATETIME、TIMESTAMP、DATE或STRING类型日期值，格式为 `yyyy-mm-dd` 、 `yyyy-mm-dd hh:mi:ss` 或 `yyyy-mm-dd hh:mi:ss.ff3` 。取值为STRING类型格式时，至少要包含 `yyyy-mm-dd` 且不含多余的字符串。

- 返回值

  返回INT类型。返回规则如下：

  - date非DATETIME、TIMESTAMP、DATE或STRING类型，或格式不符合要求时，返回NULL。
  - date值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回1970。
select year('1970-01-01 12:30:00');
--返回1970。
select year('1970-01-01');
--返回70。
select year('70-01-01');
--返回NULL。
select year('1970/03/09');
--返回NULL。
select year(null);
```

- 表数据示例

  基于示例数据，返回date1、datetime1、timestamp1和date3列日期所属年份，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select date1, year(date1) as date1_year, datetime1, year(datetime1) as datetime1_year, timestamp1, year(timestamp1) as times
tamp1_year, date3, year(date3) as date3_year from mf_date_fun_t;
```

  返回结果如下。

```
+------------+------------+---------------------+----------------+-----------------------------+-----------------+--------
----+------------+
| date1      | date1_year | datetime1           | datetime1_year | timestamp1                  | timestamp1_year | date3
| date3_year |
+------------+------------+---------------------+----------------+-----------------------------+-----------------+--------
----+------------+
| 2021-11-29 | 2021       | 2021-11-29 00:01:00 | 2021           | 2021-01-11 00:00:00.123456789 | 2021            | 2021-11
-20 | 2021       |
| 2021-11-28 | 2021       | 2021-11-28 00:02:00 | 2021           | 2021-02-11 00:00:00.123456789 | 2021            | 2021-11
-21 | 2021       |
| 2021-11-27 | 2021       | 2021-11-27 00:03:00 | 2021           | 2021-03-11 00:00:00.123456789 | 2021            | 2021-11
-22 | 2021       |
| 2021-11-26 | 2021       | 2021-11-26 00:04:00 | 2021           | 2021-04-11 00:00:00.123456789 | 2021            | 2021-11
-23 | 2021       |
| 2021-11-25 | 2021       | 2021-11-25 00:05:00 | 2021           | 2021-05-11 00:00:00.123456789 | 2021            | 2021-11
-24 | 2021       |
| 2021-11-24 | 2021       | 2021-11-24 00:06:00 | 2021           | 2021-06-11 00:00:00.123456789 | 2021            | 2021-11
-25 | 2021       |
| 2021-11-23 | 2021       | 2021-11-23 00:07:00 | 2021           | 2021-07-11 00:00:00.123456789 | 2021            | 2021-11
-26 | 2021       |
| 2021-11-22 | 2021       | 2021-11-22 00:08:00 | 2021           | 2021-08-11 00:00:00.123456789 | 2021            | 2021-11
-27 | 2021       |
| 2021-11-21 | 2021       | 2021-11-21 00:09:00 | 2021           | 2021-09-11 00:00:00.123456789 | 2021            | 2021-11
-28 | 2021       |
| 2021-11-20 | 2021       | 2021-11-20 00:10:00 | 2021           | 2021-10-11 00:00:00.123456789 | 2021            | 2021-11
-29 | 2021       |
+------------+------------+---------------------+----------------+-----------------------------+-----------------+--------
----+------------+
```

3.9.4. 数学函数

MaxCompute SQL提供了常见的数学函数供您在开发过程中使用，您可以根据实际需要选择合适的数学函数，进行数据计算、数据转换相关操作。本文为您提供MaxCompute SQL支持的数学函数的命令格式、参数说明及示例，指导您使用数学函数完成开发。

MaxCompute SQL支持的数学函数如下。

函数	功能
ABS	计算绝对值。
ACOS	计算反余弦值。
ASIN	计算反正弦值。
ATAN	计算反正切值。
CEIL	计算向上取整值。
CONV	计算进制转换值。
COS	计算余弦值。
COSH	计算双曲余弦值。
COT	计算余切值。
EXP	计算指数值。
FLOOR	计算向下取整值。
LN	计算自然对数。
LOG	计算log对数值。

函数	功能
POW	计算幂值。
RAND	返回随机数。
ROUND	返回四舍五入到指定小数点位置的值。
SIN	计算正弦值。
SINH	计算双曲正弦值。
SQRT	计算平方根。
TAN	计算正切值。
TANH	计算双曲正切值。
TRUNC	返回截取到指定小数点位置的值。
BIN	计算二进制代码值。
CBRT	计算立方根值。
CORR	计算皮尔逊系数。
DEGREES	将弧度转换为角度。
E	返回e的值。
FACTORIAL	计算阶乘值。
FORMAT_NUMBER	将数字转化为指定格式的字符串。
HEX	返回整数或字符串的十六进制格式。
LOG2	计算以2为底的对数。
LOG10	计算以10为底的对数。
PI	返回π的值。
RADIANS	将角度转换为弧度。
SIGN	返回输入参数的符号。
SHIFTLEFT	计算按位左移值。
SHIFTRIGHT	计算按位右移值。
SHIFTRIGHTUNSIGNED	计算无符号按位右移值。
UNHEX	返回十六进制字符串所代表的字符串。
WIDTH_BUCKET	返回指定字段值落入的分组编号。

注意事项

升级到MaxCompute 2.0后，产品扩展了部分函数。如果您用到的函数涉及新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP或BINARY），在使用扩展函数时，需要执行如下语句开启新数据类型开关：

- Session级别：如果使用新数据类型，您需要在SQL语句前加上语句 `set odps.sql.type.system.odps2=true;` ，并与SQL语句一起提交执行。
- Project级别：Project Owner可根据需要对Project进行设置，等待10~15分钟后才会生效。命令如下。

```
setproject odps.sql.type.system.odps2=true;
```

对 `setproject` 的详细说明请参见项目空间操作。关于开启Project级别数据类型的注意事项，请参见数据类型版本说明。

示例数据

为便于理解各函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表mf_math_fun_t，并添加数据，命令示例如下。

```
create table if not exists mf_math_fun_t(
     int_data     int,
     bigint_data  bigint,
     double_data  double,
     decimal_data decimal,
     float_data   float,
     string_data  string
    );
insert into mf_math_fun_t values
(null, -10, 0.525, 0.525BD, cast(0.525 as float), '10'),
(-20, null, -0.1, -0.1BD, cast(-0.1 as float), '-10'),
(0, -1, null, 20.45BD, cast(-1 as float), '30'),
(-40, 4, 0.89, null, cast(0.89 as float), '-30'),
(5, -50, -1, -1BD, null, '50'),
(-60, 6, 1.5, 1.5BD, cast(1.5 as float), '-50'),
(-1, -70, -7.5, -7.5BD, cast(-7.5 as float),null ),
(-80, 1, -10.2, -10.2BD, cast(-10.2 as float), '-1' ),
(9, -90, 2.58, 2.58BD, cast(2.58 as float), '0'),
(-100, 10, -5.8, -5.8BD, cast(-5.8 as float), '-90');
```

ABS

- 命令格式

```
bigint|double|decimal abs(<number>)
```

- 命令说明

  计算number的绝对值。

- 参数说明

  number：必填。DOUBLE、BIGINT或DECIMAL类型。输入为STRING时，会隐式转换为DOUBLE类型后参与运算。

  > **说明** 当输入BIGINT类型的值超过BIGINT的最大表示范围时，会返回DOUBLE类型，但可能会损失精度。

- 返回值说明

  返回值类型取决于输入参数的类型。返回规则如下：

  - number为DOUBLE、BIGINT或DECIMAL类型时会返回相应的类型。
  - number为STRING类型时，返回DOUBLE类型。
  - number为NULL，则返回NULL。

- 示例
  - 静态数据示例

```
--返回NULL。
select abs(null);
--返回1。
select abs(-1);
--返回1.2。
select abs(-1.2);
--返回2.0。
select abs("-2");
--返回1.2232083745629837E32。
select abs(1223208374562983765923874569237 48);
--取tbl1表内id字段的绝对值。下面是一个完整的ABS函数在SQL中使用的示例，其他内建函数（除窗口函数和聚合函数外）的使用方式与其类似。
select abs(id) from tbl1;
```

- 表数据示例

  基于示例数据，计算绝对值，命令示例如下。

```
select abs(bigint_data) as bigint_new, abs(double_data) as double_new, abs(decimal_data) as decimal_new, abs(string_data) as
string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------+------------+-------------+------------+
| bigint_new | double_new | decimal_new | string_new |
+------------+------------+-------------+------------+
| 10         | 0.525      | 0.525       | 10.0       |
| NULL       | 0.1        | 0.1         | 10.0       |
| 1          | NULL       | 20.45       | 30.0       |
| 4          | 0.89       | NULL        | 30.0       |
| 50         | 1.0        | 1           | 50.0       |
| 6          | 1.5        | 1.5         | 50.0       |
| 70         | 7.5        | 7.5         | NULL       |
| 1          | 10.2       | 10.2        | 1.0        |
| 90         | 2.58       | 2.58        | 0.0        |
| 10         | 5.8        | 5.8         | 90.0       |
+------------+------------+-------------+------------+
```

ACOS

- 命令格式

```
double|decimal acos(<number>)
```

- 命令说明

  计算number的反余弦函数。

- 参数说明

  number：必填。取值范围为[-1,1]。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回值类型取决于输入参数的类型。值域在0~π之间。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值不在[-1,1]范围内时，返回NULL。
  - number值为NULL时，返回NULL。

- 示例

  - 静态数据示例

```
--返回0.5155940062460905。
select acos("0.87");
--返回1.5707963267948966。
select acos(0);
--返回NULL。
select acos(null);
```

- 表数据示例

  基于示例数据，计算反余弦值，命令示例如下。

```
select acos(bigint_data) as bigint_new, acos(double_data) as double_new, acos(decimal_data) as decimal_new, acos(string_data
) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------------+------------------+------------------+--------------------+
| bigint_new       | double_new       | decimal_new      | string_new         |
+------------------+------------------+------------------+--------------------+
| NULL             | 1.0180812136981134 | 1.0180812136981134 | NULL               |
| NULL             | 1.6709637479564565 | 1.6709637479564565 | NULL               |
| 3.141592653589793 | NULL             | NULL             | NULL               |
| NULL             | 0.4734511572720662 | NULL             | NULL               |
| NULL             | 3.141592653589793  | 3.141592653589793  | NULL               |
| NULL             | NULL             | NULL             | NULL               |
| NULL             | NULL             | NULL             | NULL               |
| 0.0              | NULL             | NULL             | 3.141592653589793  |
| NULL             | NULL             | NULL             | 1.5707963267948966 |
| NULL             | NULL             | NULL             | NULL               |
+------------------+------------------+------------------+--------------------+
```

ASIN

- 命令格式

```
double|decimal asin(<number>)
```

- 命令说明

  计算number的反正弦函数。

- 参数说明

  number：必填。取值范围为[-1,1]。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回值类型取决于输入参数的类型。值域在-π/2~π/2之间。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值不在[-1,1]范围内时，返回NULL。
  - number值为NULL时，返回NULL。

- 示例

  - 静态数据示例

```
--返回1.5707963267948966。
select asin(1);
--返回-1.5707963267948966。
select asin(-1);
--返回NULL。
select asin(null);
```

- 表数据示例

  基于示例数据，计算反正弦值，命令示例如下。

```
select asin(bigint_data) as bigint_new, asin(double_data) as double_new, asin(decimal_data) as decimal_new, asin(string_data
) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+--------------------+--------------------+--------------------+--------------------+
| bigint_new         | double_new         | decimal_new        | string_new         |
+--------------------+--------------------+--------------------+--------------------+
| NULL               | 0.5527151130967832 | 0.5527151130967832 | NULL               |
| NULL               | -0.1001674211615598 | -0.1001674211615598 | NULL               |
| -1.5707963267948966| NULL               | NULL               | NULL               |
| NULL               | 1.0973451695228305 | NULL               | NULL               |
| NULL               | -1.5707963267948966 | -1.5707963267948966 | NULL               |
| NULL               | NULL               | NULL               | NULL               |
| NULL               | NULL               | NULL               | NULL               |
| 1.5707963267948966 | NULL               | NULL               | -1.5707963267948966 |
| NULL               | NULL               | NULL               | 0.0                |
| NULL               | NULL               | NULL               | NULL               |
+--------------------+--------------------+--------------------+--------------------+
```

ATAN

- 命令格式

```
double atan(<number>)
```

- 命令说明

  计算number的反正切函数。

- 参数说明

  number：必填。DOUBLE类型。输入为STRING、BIGINT、DECIMAL类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE类型。值域在-π/2~π/2之间。number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回0.7853981633974483。
select atan(1);
--返回-0.7853981633974483。
select atan(-1);
--返回NULL。
select atan(null);
```

  - 表数据示例

    基于示例数据，计算反正切值，命令示例如下。

```
select atan(bigint_data) as bigint_new, atan(double_data) as double_new, atan(decimal_data) as decimal_new, atan(string_data
) as string_new from mf_math_fun_t;
```

    返回结果如下。

```
+--------------------+---------------------+---------------------+--------------------+
| bigint_new         | double_new          | decimal_new         | string_new         |
+--------------------+---------------------+---------------------+--------------------+
| -1.4711276743037347 | 0.483447001567199   | 0.483447001567199   | 1.4711276743037347 |
| NULL               | -0.09966865249116204 | -0.09966865249116204 | -1.4711276743037347 |
| -0.7853981633974483 | NULL                | 1.521935491607842   | 1.5374753309166493 |
| 1.3258176636680326 | 0.7272626879966904  | NULL                | -1.5374753309166493 |
| -1.550798992821746 | -0.7853981633974483 | -0.7853981633974483 | 1.550798992821746  |
| 1.4056476493802699 | 0.982793723247329   | 0.982793723247329   | -1.550798992821746 |
| -1.5565115842075   | -1.4382447944982226 | -1.4382447944982226 | NULL               |
| 0.7853981633974483 | -1.473069419436178  | -1.473069419436178  | -0.7853981633974483 |
| -1.5596856728972892 | 1.2010277920014796  | 1.2010277920014796  | 0.0                |
| 1.4711276743037347 | -1.4000611153196139 | -1.4000611153196139 | -1.5596856728972892 |
+--------------------+---------------------+---------------------+--------------------+
```

CEIL

- 命令格式

```
bigint ceil(<value>)
```

- 命令说明

  向上取整，返回不小于输入值value的最小整数。

- 参数说明

  value：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回BIGINT类型。number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

    ```
    --返回2。
    select ceil(1.1);
    --返回-1。
    select ceil(-1.1);
    --返回NULL。
    select ceil(null);
    ```

  - 表数据示例

    基于示例数据，向上取整，命令示例如下。

    ```
    select ceil(bigint_data) as bigint_new, ceil(double_data) as double_new, ceil(decimal_data) as decimal_new, ceil(string_data
    ) as string_new from mf_math_fun_t;
    ```

    返回结果如下。

    ```
    +------------+------------+-------------+------------+
    | bigint_new | double_new | decimal_new | string_new |
    +------------+------------+-------------+------------+
    | -10        | 1          | 1           | 10         |
    | NULL       | 0          | 0           | -10        |
    | -1         | NULL       | 21          | 30         |
    | 4          | 1          | NULL        | -30        |
    | -50        | -1         | -1          | 50         |
    | 6          | 2          | 2           | -50        |
    | -70        | -7         | -7          | NULL       |
    | 1          | -10        | -10         | -1         |
    | -90        | 3          | 3           | 0          |
    | 10         | -5         | -5          | -90        |
    +------------+------------+-------------+------------+
    ```

CONV

- 命令格式

  ```
  string conv(<input>, bigint <from_base>, bigint <to_base>)
  ```

- 命令说明

  该函数为进制转换函数。

- 参数说明
  - input：必填。以STRING表示的要转换的整数值，支持BIGINT和DOUBLE的隐式转换。
  - from_base、to_base：必填。以十进制表示的进制值，支持的值为2、8、10和16。支持STRING及DOUBLE类型的隐式转换。

- 返回值说明

  返回STRING类型。返回规则如下：
  - input、from_base或to_base值为NULL时，返回NULL。
  - 转换过程以64位精度工作，溢出时返回NULL。
  - input如果是负值，即以短划线（-）开头，返回NULL。如果输入的是小数，会转为整数值后进行进制转换，小数部分会被舍弃。

- 示例

- 静态数据示例

```
--返回12。
select conv('1100', 2, 10);
--返回C。
select conv('1100', 2, 16);
--返回171。
select conv('ab', 16, 10);
--返回AB。
select conv('ab', 16, 16);
--返回NULL。
select conv('1100', null, 10);
```

- 表数据示例

  基于示例数据，转换为二进制形式，命令示例如下。

```
select conv(bigint_data,10,2) as bigint_new, conv(double_data,10,2) as double_new, conv(decimal_data,10,2) as decimal_new, c
onv(string_data,10,2) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------+------------+-------------+------------+
| bigint_new | double_new | decimal_new | string_new |
+------------+------------+-------------+------------+
| NULL       | 0          | 0           | 1010       |
| NULL       | NULL       | NULL        | NULL       |
| NULL       | NULL       | 10100       | 11110      |
| 100        | 0          | NULL        | NULL       |
| NULL       | NULL       | NULL        | 110010     |
| 110        | 1          | 1           | NULL       |
| NULL       | NULL       | NULL        | NULL       |
| 1          | NULL       | NULL        | NULL       |
| NULL       | 10         | 10          | 0          |
| 1010       | NULL       | NULL        | NULL       |
+------------+------------+-------------+------------+
```

COS

- 命令格式

```
double|decimal cos(<number>)
```

- 命令说明

  计算number的余弦函数，输入为弧度值。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2.6794896585028633e-8。
select cos(3.1415926/2);
--返回-0.9999999999999986。
select cos(3.1415926);
--返回NULL。
select cos(null);
```

- 表数据示例

  基于示例数据，计算余弦值，命令示例如下。

```
select cos(bigint_data) as bigint_new, cos(double_data) as double_new, cos(decimal_data) as decimal_new, cos(string_data) as
string_new from mf_math_fun_t;
```

  返回结果如下。

```
+---------------------+--------------------+----------------------+---------------------+
| bigint_new          | double_new         | decimal_new          | string_new          |
+---------------------+--------------------+----------------------+---------------------+
| -0.8390715290764524 | 0.8653239416229412 | 0.8653239416229412   | -0.8390715290764524 |
| NULL                | 0.9950041652780258 | 0.9950041652780258   | -0.8390715290764524 |
| 0.5403023058681398  | NULL               | -0.02964340851507803 | 0.15425144988758405 |
| -0.6536436208636119 | 0.6294120265736969 | NULL                 | 0.15425144988758405 |
| 0.9649660284921133  | 0.5403023058681398 | 0.5403023058681398   | 0.9649660284921133  |
| 0.960170286650366   | 0.0707372016677029 | 0.0707372016677029   | 0.9649660284921133  |
| 0.6333192030862999  | 0.3466353178350258 | 0.3466353178350258   | NULL                |
| 0.5403023058681398  | -0.7142656520272003| -0.7142656520272003  | 0.5403023058681398  |
| -0.4480736161291701 | -0.8464080412157756| -0.8464080412157756  | 1.0                 |
| -0.8390715290764524 | 0.8855195169413189 | 0.8855195169413189   | -0.4480736161291701 |
+---------------------+--------------------+----------------------+---------------------+
```

COSH

- 命令格式

```
double|decimal cosh(<number>)
```

- 命令说明

  计算number的双曲余弦函数。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例

  - 静态数据示例

```
--返回2.5091784169949913。
select cosh(3.1415926/2);
--返回NULL。
select cosh(null);
```

  - 表数据示例

    基于示例数据，计算双曲余弦值，命令示例如下。

```
select cosh(bigint_data) as bigint_new, cosh(double_data) as double_new, cosh(decimal_data) as decimal_new, cosh(string_data
) as string_new from mf_math_fun_t;
```

    返回结果如下。

```
+----------------------+--------------------+--------------------+----------------------+
| bigint_new           | double_new         | decimal_new        | string_new           |
+----------------------+--------------------+--------------------+----------------------+
| 11013.232920103324   | 1.1410071063729532 | 1.1410071063729532 | 11013.232920103324   |
| NULL                 | 1.0050041680558035 | 1.0050041680558035 | 11013.232920103324   |
| 1.5430806348152437   | NULL               | 380445243.96844625 | 5343237290762.231    |
| 27.308232836016487   | 1.42289270202111   | NULL               | 5343237290762.231    |
| 2.592352764293536e21 | 1.5430806348152437 | 1.5430806348152437 | 2.592352764293536e21 |
| 201.7156361224559    | 2.352409615243247  | 2.352409615243247  | 2.592352764293536e21 |
| 1.2577193354595834e30| 904.0214837702166  | 904.0214837702166  | NULL                 |
| 1.5430806348152437   | 13451.593055733929 | 13451.593055733929 | 1.5430806348152437   |
| 6.102016471589204e38 | 6.636456081840602  | 6.636456081840602  | 1.0                  |
| 11013.232920103324   | 165.151293732197   | 165.151293732197   | 6.102016471589204e38 |
+----------------------+--------------------+--------------------+----------------------+
```

COT

- 命令格式

```
double|decimal cot(<number>)
```

- 命令说明

  计算number的余切函数，输入为弧度值。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2.6794896585028643E-8。
select cot(3.1415926/2);
--返回NULL。
select cot(null);
```

  - 表数据示例

    基于示例数据，计算余切值，命令示例如下。

```
select cosh(bigint_data) as bigint_new, cosh(double_data) as double_new, cosh(decimal_data) as decimal_new, cosh(string_data
) as string_new from mf_math_fun_t;
```

    返回结果如下。

```
+----------------------+--------------------+--------------------+----------------------+
| bigint_new           | double_new         | decimal_new        | string_new           |
+----------------------+--------------------+--------------------+----------------------+
| 11013.232920103324   | 1.1410071063729532 | 1.1410071063729532 | 11013.232920103324   |
| NULL                 | 1.0050041680558035 | 1.0050041680558035 | 11013.232920103324   |
| 1.5430806348152437   | NULL               | 380445243.96844625 | 5343237290762.231    |
| 27.308232836016487   | 1.42289270202111   | NULL               | 5343237290762.231    |
| 2.592352764293536e21 | 1.5430806348152437 | 1.5430806348152437 | 2.592352764293536e21 |
| 201.7156361224559    | 2.352409615243247  | 2.352409615243247  | 2.592352764293536e21 |
| 1.2577193354595834e30 | 904.0214837702166 | 904.0214837702166  | NULL                 |
| 1.5430806348152437   | 13451.593055733929 | 13451.593055733929 | 1.5430806348152437   |
| 6.102016471589204e38 | 6.636456081840602  | 6.636456081840602  | 1.0                  |
| 11013.232920103324   | 165.151293732197   | 165.151293732197   | 6.102016471589204e38 |
+----------------------+--------------------+--------------------+----------------------+
```

EXP

- 命令格式

```
double|decimal exp(<number>)
```

- 命令说明

  计算number的指数函数。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回4.810477252069109。
select exp(3.1415926/2);
--返回NULL。
select exp(null);
```

- 表数据示例

  基于示例数据，计算指数值，命令示例如下。

```
select exp(bigint_data) as bigint_new, exp(double_data) as double_new, exp(decimal_data) as decimal_new, exp(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+-------------------------+-------------------------+-------------------------+-------------------------+
| bigint_new              | double_new              | decimal_new             | string_new              |
+-------------------------+-------------------------+-------------------------+-------------------------+
| 0.000045399929762484854 | 1.6904588483790914      | 1.6904588483790914      | 22026.465794806718      |
| NULL                    | 0.9048374180359595      | 0.9048374180359595      | 0.000045399929762484854 |
| 0.36787944117144233     | NULL                    | 760890487.9368925       | 10686474581524.463      |
| 54.598150033144236      | 2.4351296512898744      | NULL                    | 9.357622968840175e-14   |
| 1.9287498479639178e-22  | 0.36787944117144233     | 0.36787944117144233     | 5.184705528587072e21    |
| 403.4287934927351       | 4.4816890703380645      | 4.4816890703380645      | 1.9287498479639178e-22  |
| 3.975449735908647e-31   | 0.0005530843701478336   | 0.0005530843701478336   | NULL                    |
| 2.718281828459045       | 0.000037170318684126734 | 0.000037170318684126734 | 0.36787944117144233     |
| 8.194012623990515e-40   | 13.197138159658358      | 13.197138159658358      | 1.0                     |
| 22026.465794806718      | 0.0030275547453758153   | 0.0030275547453758153   | 8.194012623990515e-40   |
+-------------------------+-------------------------+-------------------------+-------------------------+
```

FLOOR

- 命令格式

```
bigint floor(<number>)
```

- 命令说明

  向下取整，返回不大于number的最大整数值。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回BIGINT类型。number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回1。
select floor(1.2);
--返回0。
select floor(0.1);
--返回-2。
select floor(-1.2);
--返回-1。
select floor(-0.1);
--返回0。
select floor(0.0);
--返回0。
select floor(-0.0);
--返回NULL。
select floor(null);
```

- 表数据示例

  基于示例数据，向下取整，命令示例如下。

```
select floor(bigint_data) as bigint_new, floor(double_data) as double_new, floor(decimal_data) as decimal_new, floor(string_
data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------+------------+-------------+------------+
| bigint_new | double_new | decimal_new | string_new |
+------------+------------+-------------+------------+
| -10        | 0          | 0           | 10         |
| NULL       | -1         | -1          | -10        |
| -1         | NULL       | 20          | 30         |
| 4          | 0          | NULL        | -30        |
| -50        | -1         | -1          | 50         |
| 6          | 1          | 1           | -50        |
| -70        | -8         | -8          | NULL       |
| 1          | -11        | -11         | -1         |
| -90        | 2          | 2           | 0          |
| 10         | -6         | -6          | -90        |
+------------+------------+-------------+------------+
```

LN

- 命令格式

```
double|decimal ln(<number>)
```

- 命令说明

  计算number的自然对数。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为负数或0时，返回NULL。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回1.144729868791239。
select ln(3.1415926);
--返回NULL。
select ln(null);
```

  - 表数据示例

    基于示例数据，计算自然对数值，命令示例如下。

```
select ln(bigint_data) as bigint_new, ln(double_data) as double_new, ln(decimal_data) as decimal_new, ln(string_data) as str
ing_new from mf_math_fun_t;
```

    返回结果如下。

```
+--------------------+---------------------+---------------------+---------------------+
| bigint_new         | double_new          | decimal_new         | string_new          |
+--------------------+---------------------+---------------------+---------------------+
| NULL               | -0.6443570163905132 | -0.6443570163905132 | 2.302585092994046   |
| NULL               | NULL                | NULL                | NULL                |
| NULL               | NULL                | 3.017982882488811   | 3.4011973816621555  |
| 1.3862943611198906 | -0.11653381625595151 | NULL                | NULL                |
| NULL               | NULL                | NULL                | 3.912023005428146   |
| 1.791759469228055  | 0.4054651081081644  | 0.4054651081081644  | NULL                |
| NULL               | NULL                | NULL                | NULL                |
| 0.0                | NULL                | NULL                | NULL                |
| NULL               | 0.9477893989335261  | 0.9477893989335261  | NULL                |
| 2.302585092994046  | NULL                | NULL                | NULL                |
+--------------------+---------------------+---------------------+---------------------+
```

LOG

- 命令格式

```
double log(<base>, <x>)
```

- 命令说明

  计算以base为底的x的对数。

- 参数说明
  - base：必填。底数。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。
  - x：必填。待计算对数的值。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE类型。 返回规则如下：
  - base或x为NULL时，返回NULL。
  - base或x为负数或0时，返回NULL。
  - 如果base为1（会引发一个除零行为），会返回NULL。

- 示例
  - 静态数据示例

```
--返回4.0。
select log(2, 16);
--返回NULL。
select log(2, null);
```

  - 表数据示例

    基于示例数据，计算列以2为底的对数值，命令示例如下。

```
select log(2,bigint_data) as bigint_new, log(2,double_data) as double_new, log(2,decimal_data) as decimal_new, log(2,string_
data) as string_new from mf_math_fun_t;
```

    返回结果如下。

```
+--------------------+----------------------+--------------------+--------------------+
| bigint_new         | double_new           | decimal_new        | string_new         |
+--------------------+----------------------+--------------------+--------------------+
| NULL               | -0.929610672108602   | -0.929610672108602 | 3.3219280948873626 |
| NULL               | NULL                 | NULL               | NULL               |
| NULL               | NULL                 | 4.354028938054387  | 4.906890595608519  |
| 2.0                | -0.16812275880832692 | NULL               | NULL               |
| NULL               | NULL                 | NULL               | 5.643856189774724  |
| 2.584962500721156  | 0.5849625007211562   | 0.5849625007211562 | NULL               |
| NULL               | NULL                 | NULL               | NULL               |
| 0.0                | NULL                 | NULL               | NULL               |
| NULL               | 1.3673710656485296   | 1.3673710656485296 | NULL               |
| 3.3219280948873626 | NULL                 | NULL               | NULL               |
+--------------------+----------------------+--------------------+--------------------+
```

POW

- 命令格式

```
double|decimal pow(<x>, <y>)
```

- 命令说明

  计算x的y次方，即 `x^y` 。

- 参数说明
  - x：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。
  - y：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：
  - x或y为DOUBLE、DECIMAL类型时会返回相应的类型。
  - x或y为STRING、BIGINT类型时，返回DOUBLE类型。
  - x或y值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--返回65536.0。
select pow(2, 16);
--返回NULL。
select pow(2, null);
```

- 表数据示例

基于示例数据，计算列的2次方，命令示例如下。

```
select pow(bigint_data, 2) as bigint_new, pow(double_data, 2) as double_new, pow(decimal_data, 2) as decimal_new, pow(string_data, 2) as string_new from mf_math_fun_t;
```

返回结果如下。

```
+------------+----------------------+----------------------+------------+
| bigint_new | double_new           | decimal_new          | string_new |
+------------+----------------------+----------------------+------------+
| 100.0      | 0.275625             | 0.275625             | 100.0      |
| NULL       | 0.010000000000000002 | 0.010000000000000002 | 100.0      |
| 1.0        | NULL                 | 418.2025             | 900.0      |
| 16.0       | 0.7921               | NULL                 | 900.0      |
| 2500.0     | 1.0                  | 1.0                  | 2500.0     |
| 36.0       | 2.25                 | 2.25                 | 2500.0     |
| 4900.0     | 56.25                | 56.25                | NULL       |
| 1.0        | 104.03999999999999   | 104.03999999999999   | 1.0        |
| 8100.0     | 6.6564000000000005   | 6.6564000000000005   | 0.0        |
| 100.0      | 33.64                | 33.64                | 8100.0     |
+------------+----------------------+----------------------+------------+
```

RAND

- 命令格式

```
double rand(bigint <seed>)
```

- 命令说明

返回DOUBLE类型的随机数，返回值区间是0～1。

- 参数说明

seed：可选。BIGINT类型。随机数种子，决定随机数序列的起始值。

> **说明** 您可以通过seed确定随机数序列，seed确定后，即可固定该函数的返回结果。执行环境是一样的情况下，seed取值不变的话，返回值是一样的，如果需要返回不一样的结果，需要修改seed取值。

- 返回值说明

返回DOUBLE类型。

- 示例

```
--返回4.7147460303803655E-4。
select rand();
--返回0.7308781907032909。
select rand(1);
```

ROUND

- 命令格式

```
double|decimal round(<number>[, bigint <decimal_places>])
```

- 命令说明

四舍五入到指定小数点位置。

- 参数说明
  - number：必填。DOUBLE类型或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。
  - decimal_places：可选。BIGINT类型常量，四舍五入计算到小数点后的位置。如果省略表示四舍五入到个位数，默认值为0。

> **说明** decimal_places可以是负数。负数会从小数点向左开始计数，并且不保留小数部分。如果decimal_places超过了整数部分长度，返回0。

- 返回值说明

返回DOUBLE类型或DECIMAL类型。返回规则如下：

- number为DOUBLE、DECIMAL类型时会返回相应的类型。
- number为STRING、BIGINT类型时，返回DOUBLE类型。
- decimal_places非BIGINT类型时，返回报错。
- number或decimal_places值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回125.0。
select round(125.315);
--返回125.3。
select round(125.315, 1);
--返回125.32。
select round(125.315, 2);
--返回125.315。
select round(125.315, 3);
--返回-125.32。
select round(-125.315, 2);
--返回100.0。
select round(123.345, -2);
--返回NULL。
select round(null);
--返回123.345。
select round(123.345, 4);
--返回0.0。
select round(123.345, -4);
```

  - 表数据示例

    基于示例数据，对列四舍五入到小数点指定位置，命令示例如下。

```
select round(bigint_data, 1) as bigint_new, round(double_data, 2) as double_new, round(decimal_data, 1) as decimal_new, roun
d(string_data) as string_new from mf_math_fun_t;
```

    返回结果如下。

```
+------------+------------+-------------+------------+
| bigint_new | double_new | decimal_new | string_new |
+------------+------------+-------------+------------+
| -10.0      | 0.53       | 0.5         | 10.0       |
| NULL       | -0.1       | -0.1        | -10.0      |
| -1.0       | NULL       | 20.5        | 30.0       |
| 4.0        | 0.89       | NULL        | -30.0      |
| -50.0      | -1.0       | -1          | 50.0       |
| 6.0        | 1.5        | 1.5         | -50.0      |
| -70.0      | -7.5       | -7.5        | NULL       |
| 1.0        | -10.2      | -10.2       | -1.0       |
| -90.0      | 2.58       | 2.6         | 0.0        |
| 10.0       | -5.8       | -5.8        | -90.0      |
+------------+------------+-------------+------------+
```

SIN

- 命令格式

```
double|decimal sin(<number>)
```

- 命令说明

  计算number的正弦函数，输入为弧度值。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：
  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--返回-0.3048106211022167。
select sin(60);
--返回NULL。
select sin(null);
```

- 表数据示例

  基于示例数据，计算列的正弦值，命令示例如下。

```
select sin(bigint_data) as bigint_new, sin(double_data) as double_new, sin(decimal_data) as decimal_new, sin(string_data) as
string_new from mf_math_fun_t;
```

  返回结果如下。

```
+----------------------+----------------------+----------------------+----------------------+
| bigint_new           | double_new           | decimal_new          | string_new           |
+----------------------+----------------------+----------------------+----------------------+
| 0.5440211108893698   | 0.5012130046737979   | 0.5012130046737979   | -0.5440211108893698  |
| NULL                 | -0.09983341664682815 | -0.09983341664682815 | 0.5440211108893698   |
| -0.8414709848078965  | NULL                 | 0.9995605376022045   | -0.9880316240928618  |
| -0.7568024953079282  | 0.7770717475268238   | NULL                 | 0.9880316240928618   |
| 0.26237485370392877  | -0.8414709848078965  | -0.8414709848078965  | -0.26237485370392877 |
| -0.27941549819892586 | 0.9974949866040544   | 0.9974949866040544   | 0.26237485370392877  |
| -0.7738906815578891  | -0.9379999767747389  | -0.9379999767747389  | NULL                 |
| 0.8414709848078965   | 0.6998746875935423   | 0.6998746875935423   | -0.8414709848078965  |
| -0.8939966636005579  | 0.5325349075556212   | 0.5325349075556212   | 0.0                  |
| -0.5440211108893698  | 0.46460217941375737  | 0.46460217941375737  | -0.8939966636005579  |
+----------------------+----------------------+----------------------+----------------------+
```

SINH

- 命令格式

```
double|decimal sinh(<number>)
```

- 命令说明

  计算number的双曲正弦函数。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回5.343237290762231E12。
select sinh(30);
--返回NULL。
select sinh(null);
```

- 表数据示例

  基于示例数据，计算列的双曲正弦值，命令示例如下。

```
select sinh(bigint_data) as bigint_new, sinh(double_data) as double_new, sinh(decimal_data) as decimal_new, sinh(string_data
) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------------------+----------------------+----------------------+-----------------------+
| bigint_new             | double_new           | decimal_new          | string_new            |
+------------------------+----------------------+----------------------+-----------------------+
| -11013.232874703393    | 0.5494517420061382   | 0.5494517420061382   | 11013.232874703393    |
| NULL                   | -0.10016675001984403 | -0.10016675001984403 | -11013.232874703393   |
| -1.1752011936438014    | NULL                 | 380445243.96844625   | 5343237290762.231     |
| 27.28991719712775      | 1.0122369492687646   | NULL                 | -5343237290762.231    |
| -2.592352764293536e21  | -1.1752011936438014  | -1.1752011936438014  | 2.592352764293536e21  |
| 201.71315737027922     | 2.1292794550948173   | 2.1292794550948173   | -2.592352764293536e21 |
| -1.2577193354595834e30 | -904.0209306858466   | -904.0209306858466   | NULL                  |
| 1.1752011936438014     | -13451.593018563612  | -13451.593018563612  | -1.1752011936438014   |
| -6.102016471589204e38  | 6.560682077817757    | 6.560682077817757    | 0.0                   |
| 11013.232874703393     | -165.1482661774516   | -165.1482661774516   | -6.102016471589204e38 |
+------------------------+----------------------+----------------------+-----------------------+
```

SQRT

- 命令格式

```
double|decimal sqrt(<number>)
```

- 命令说明

  计算number的平方根。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型，必须大于0，小于0时返回NULL。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2.0。
select sqrt(4);
--返回NULL。
select sqrt(null);
```

  - 表数据示例

    基于示例数据，计算列的平方根，命令示例如下。

```
select sqrt(bigint_data) as bigint_new, sqrt(double_data) as double_new, sqrt(decimal_data) as decimal_new, sqrt(string_data
) as string_new from mf_math_fun_t;
```

    返回结果如下。

```
+--------------------+--------------------+--------------------+--------------------+
| bigint_new         | double_new         | decimal_new        | string_new         |
+--------------------+--------------------+--------------------+--------------------+
| NULL               | 0.724568837309472  | 0.724568837309472  | 3.1622776601683795 |
| NULL               | NULL               | NULL               | NULL               |
| NULL               | NULL               | 4.522167621838006  | 5.477225575051661  |
| 2.0                | 0.9433981132056604 | NULL               | NULL               |
| NULL               | NULL               | NULL               | 7.0710678118654755 |
| 2.449489742783178  | 1.224744871391589  | 1.224744871391589  | NULL               |
| NULL               | NULL               | NULL               | NULL               |
| 1.0                | NULL               | NULL               | NULL               |
| NULL               | 1.606237840420901  | 1.606237840420901  | 0.0                |
| 3.1622776601683795 | NULL               | NULL               | NULL               |
+--------------------+--------------------+--------------------+--------------------+
```

TAN

- 命令格式

```
double|decimal tan(<number>)
```

- 命令说明

  计算number的正切函数，输入为弧度值。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回-6.405331196646276。
select tan(30);
--返回NULL。
select tan(null);
```

  - 表数据示例

    基于示例数据，计算列的正切值，命令示例如下。

```
select tan(bigint_data) as bigint_new, tan(double_data) as double_new, tan(decimal_data) as decimal_new, tan(string_data) as
string_new from mf_math_fun_t;
```

    返回结果如下。

```
+----------------------+----------------------+----------------------+----------------------+
| bigint_new           | double_new           | decimal_new          | string_new           |
+----------------------+----------------------+----------------------+----------------------+
| -0.6483608274590866  | 0.5792200822893652   | 0.5792200822893652   | 0.6483608274590866   |
| NULL                 | -0.10033467208545055 | -0.10033467208545055 | -0.6483608274590866  |
| -1.5574077246549023  | NULL                 | -33.71948732190433   | -6.405331196646276   |
| 1.1578212823495775   | 1.2345994590490046   | NULL                 | 6.405331196646276    |
| 0.27190061199763077  | -1.5574077246549023  | -1.5574077246549023  | -0.27190061199763077 |
| -0.29100619138474915 | 14.101419947171719   | 14.101419947171719   | 0.27190061199763077  |
| -1.2219599181369434  | -2.706013866772691   | -2.706013866772691   | NULL                 |
| 1.5574077246549023   | -0.979852083895097   | -0.979852083895097   | -1.5574077246549023  |
| 1.995200412208242    | -0.6291704256385503  | -0.6291704256385503  | 0.0                  |
| 0.6483608274590866   | 0.5246662219468002   | 0.5246662219468002   | 1.995200412208242    |
+----------------------+----------------------+----------------------+----------------------+
```

TANH

- 命令格式

```
double|decimal tanh(<number>)
```

- 命令说明

  计算number的双曲正切函数。

- 参数说明

  number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：

  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回1.0。
select tanh(30);
--返回NULL。
select tanh(null);
```

- 表数据示例

  基于示例数据，计算列的双曲正切值，命令示例如下。

```
select tanh(bigint_data) as bigint_new, tanh(double_data) as double_new, tanh(decimal_data) as decimal_new, tanh(string_data
) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+---------------------+---------------------+---------------------+---------------------+
| bigint_new          | double_new          | decimal_new         | string_new          |
+---------------------+---------------------+---------------------+---------------------+
| -0.9999999958776927 | 0.48154979836430806 | 0.48154979836430806 | 0.9999999958776927  |
| NULL                | -0.09966799462495582 | -0.09966799462495582 | -0.9999999958776927 |
| -0.7615941559557649 | NULL                | 1.0                 | 1.0                 |
| 0.999329299739067   | 0.7113937318189625  | NULL                | -1.0                |
| -1.0                | -0.7615941559557649 | -0.7615941559557649 | 1.0                 |
| 0.9999877116507956  | 0.9051482536448664  | 0.9051482536448664  | -1.0                |
| -1.0                | -0.9999993881955461 | -0.9999993881955461 | NULL                |
| 0.7615941559557649  | -0.9999999972367348 | -0.9999999972367348 | -0.7615941559557649 |
| -1.0                | 0.9885821584459533  | 0.9885821584459533  | 0.0                 |
| 0.9999999958776927  | -0.9999816679925603 | -0.9999816679925603 | -1.0                |
+---------------------+---------------------+---------------------+---------------------+
```

TRUNC

- 命令格式

```
double|decimal trunc(<number>[, bigint <decimal_places>])
```

- 命令说明

  将输入值number截取到指定小数点位置。

  > **说明** Hive兼容模式下，该函数不再是数学函数，而用于实现日期转换，对应日期函数请参见DATETRUNC。请根据实际情况设置MaxCompute项目的数据类型版本，更多数据类型版本信息，请参见数据类型版本说明。

- 参数说明
  - number：必填。DOUBLE或DECIMAL类型。输入为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算。
  - decimal_places：可选。BIGINT类型常量，要截取到的小数点位置。省略此参数时默认截取到个位数。decimal_places可以是负数，负数会从小数点向左开始截取，并且不保留小数部分。如果decimal_places超过了整数部分长度，则返回0。
- 返回值说明

  返回DOUBLE或DECIMAL类型。返回规则如下：
  - number为DOUBLE、DECIMAL类型时会返回相应的类型。
  - number为STRING、BIGINT类型时，返回DOUBLE类型。
  - decimal_places非BIGINT类型时，返回报错。
  - number或decimal_places值为NULL时，返回NULL。

  > **说明**
  > - 返回DOUBLE类型时，返回的结果显示可能不符合预期，例如 `trunc(125.815,1)` ，这个DOUBLE类型显示问题任何系统都存在。
  > - 截取掉的部分补0。

- 示例

- 静态数据示例

```
--返回125.0。
select trunc(125.815,0);
--返回125.80000000000001。
select trunc(125.815,1);
--返回125.81。
select trunc(125.815,2);
--返回125.815。
select trunc(125.815,3);
--返回-125.81。
select trunc(-125.815,2);
--返回120.0。
select trunc(125.815,-1);
--返回100.0。
select trunc(125.815,-2);
--返回0.0。
select trunc(125.815,-3);
--返回123.345。
select trunc(123.345,4);
--返回0.0。
select trunc(123.345,-4);
--返回NULL。
select trunc(123.345,null);
```

- 表数据示例

  基于示例数据，将列截取到指定小数点位置，命令示例如下。

```
select trunc(bigint_data, 1) as bigint_new, trunc(double_data,1) as double_new, trunc(decimal_data, 1) as decimal_new, trunc
(string_data, 1) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------+---------------------+-------------+------------+
| bigint_new | double_new          | decimal_new | string_new |
+------------+---------------------+-------------+------------+
| -10.0      | 0.5                 | 0.5         | 10.0       |
| NULL       | -0.1                | -0.1        | -10.0      |
| -1.0       | NULL                | 20.4        | 30.0       |
| 4.0        | 0.8                 | NULL        | -30.0      |
| -50.0      | -1.0                | -1          | 50.0       |
| 6.0        | 1.5                 | 1.5         | -50.0      |
| -70.0      | -7.5                | -7.5        | NULL       |
| 1.0        | -10.200000000000001 | -10.2       | -1.0       |
| -90.0      | 2.5                 | 2.5         | 0.0        |
| 10.0       | -5.800000000000001  | -5.8        | -90.0      |
+------------+---------------------+-------------+------------+
```

BIN

- 命令格式

```
string bin(<number>)
```

- 命令说明

  返回number的二进制代码表示。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。BIGINT、INT、SMALLINT、TINYINT类型。

- 返回值说明

  返回STRING类型。返回规则如下：

  - number非BIGINT、INT、SMALLINT、TINYINT类型时，返回报错。
  - number值为0时，返回0。
  - number值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--返回0。
select bin(0);
--返回NULL。
select bin(null);
--返回1100。
select bin(12);
```

- 表数据示例

  基于示例数据，获取int_data和bigint_data两列的二进制表示形式，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select bin(int_data) as int_new, bin(bigint_data) as bigint_new from mf_math_fun_t;
```

  返回结果如下。

```
+-----------------------------------------------------------------------------+-----------------------------------------------
--------------------+
| int_new                                                                     | bigint_new
|
+-----------------------------------------------------------------------------+-----------------------------------------------
--------------------+
| NULL                                                                        | 1111111111111111111111111111111111111111111111
111111111111110110 |
| 1111111111111111111111111111111111111111111111111111111101100           | NULL
|
| 0                                                                           | 1111111111111111111111111111111111111111111111
111111111111111111 |
| 1111111111111111111111111111111111111111111111111111111011000           | 100
|
| 101                                                                         | 1111111111111111111111111111111111111111111111
111111111111001110 |
| 1111111111111111111111111111111111111111111111111111111000100           | 110
|
| 1111111111111111111111111111111111111111111111111111111111111111           | 1111111111111111111111111111111111111111111111
111111111110111010 |
| 1111111111111111111111111111111111111111111111111111110110000           | 1
|
| 1001                                                                        | 1111111111111111111111111111111111111111111111
111111111110100110 |
| 1111111111111111111111111111111111111111111111111111110011100           | 1010
|
+-----------------------------------------------------------------------------+-----------------------------------------------
--------------------+
```

CBRT

- 命令格式

```
double cbrt(<number>)
```

- 命令说明

  计算number的立方根。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。BIGINT、INT、SMALLINT、TINYINT、DOUBLE、FLOAT、STRING类型数据。

- 返回值说明

  返回DOUBLE类型。返回规则如下：

  - number非BIGINT、INT、SMALLINT、TINYINT、DOUBLE、FLOAT、STRING类型时，返回报错。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回2.0。
select cbrt(8);
--返回NULL。
select cbrt(null);
```

- 表数据示例

  基于示例数据，获取除decimal_data之外其他列的立方根，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select cbrt(int_data) as int_new, cbrt(bigint_data) as bigint_new, cbrt(double_data) as double_new, cbrt(float_data) as floa
t_new, cbrt(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+--------------------+--------------------+---------------------+--------------------+--------------------+
| int_new            | bigint_new         | double_new          | float_new          | string_new         |
+--------------------+--------------------+---------------------+--------------------+--------------------+
| NULL               | -2.1544346900318834 | 0.806714323012272   | 0.8067143108004823 | 2.1544346900318834 |
| -2.7144176165949063 | NULL               | -0.46415888336127786 | -0.46415888566678  | -2.1544346900318834 |
| 0.0                | -1.0               | NULL                | -1.0               | 3.107232505953859  |
| -3.4199518933533937 | 1.5874010519681996 | 0.9619001716077046  | 0.961900166454112  | -3.107232505953859 |
| 1.7099759466766968 | -3.6840314986403864 | -1.0                | NULL               | 3.6840314986403864 |
| -3.9148676411688634 | 1.8171205928321394 | 1.1447142425533317  | 1.1447142425533317 | -3.6840314986403864 |
| -1.0               | -4.121285299808557 | -1.9574338205844317 | -1.9574338205844317 | NULL               |
| -4.308869380063767 | 1.0                | -2.168702885250197  | -2.1687028717323127 | -1.0               |
| 2.080083823051904  | -4.481404746557165 | 1.3715339700741747  | 1.3715339565548288 | 0.0                |
| -4.641588833612778 | 2.1544346900318834 | -1.7967017791430528 | -1.7967017988380907 | -4.481404746557165 |
+--------------------+--------------------+---------------------+--------------------+--------------------+
```

CORR

- 命令格式

```
double corr(<col1>, <col2>)
```

- 命令说明

  计算两列数据的皮尔逊系数（Pearson Correlation Coefficien）。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  col1、col2：必填。待计算皮尔逊系数的表的两个列名称。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL类型。col1和col2的数据类型可以不相同。

- 返回值说明

  返回DOUBLE类型。如果某一输入列的某一行存在NULL值，该行不参与计算。

- 示例

  基于示例数据，计算double_data和float_data两列的皮尔逊系数，命令示例如下。

```
select corr(double_data,float_data) from mf_math_fun_t;
```

  返回结果为1.0。

DEGREES

- 命令格式

```
double degrees(<number>)
```

- 命令说明

  将弧度转换为角度。

- 参数说明

  number：必填。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL、STRING类型。此函数为MaxCompute 2.0扩展函数。

- 返回值说明

  返回DOUBLE类型。number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回90.0。
select degrees(1.5707963267948966);
--返回0.0。
select degrees(0);
--返回NULL。
select degrees(null);
```

- 表数据示例

  基于示例数据，获取所有列对应的角度，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select degrees(int_data) as int_new, degrees(bigint_data) as bigint_new, degrees(double_data) as double_new, degrees(decimal
_data) as decimal_new, degrees(float_data) as float_new, degrees(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+---------------------+---------------------+---------------------+---------------------+---------------------+-------------
--------+
| int_new             | bigint_new          | double_new          | decimal_new         | float_new           | string_new
|
+---------------------+---------------------+---------------------+---------------------+---------------------+-------------
--------+
| NULL                | -572.9577951308232  | 30.08028424436822   | 30.08028424436822   | 30.080282878330387  | 572.95779513
08232   |
| -1145.9155902616465 | NULL                | -5.729577951308232  | -5.729577951308232  | -5.729578036685597  | -572.9577951
308232  |
| 0.0                 | -57.29577951308232  | NULL                | 1171.6986910425335  | -57.29577951308232  | 1718.8733853
924698  |
| -2291.831180523293  | 229.1831180523293   | 50.99324376664326   | NULL                | 50.99324294702057   | -1718.873385
3924698 |
| 286.4788975654116   | -2864.7889756541163 | -57.29577951308232  | -57.29577951308232  | NULL                | 2864.7889756
541163  |
| -3437.7467707849396 | 343.77467707849394  | 85.94366926962348   | 85.94366926962348   | 85.94366926962348   | -2864.788975
6541163 |
| -57.29577951308232  | -4010.7045659157625 | -429.71834634811745 | -429.71834634811745 | -429.71834634811745 | NULL
|
| -4583.662361046586  | 57.29577951308232   | -584.4169510334397  | -584.4169510334397  | -584.416940105137   | -57.29577951
308232  |
| 515.662015617741    | -5156.620156177409  | 147.8231111437524   | 147.8231111437524   | 147.82310677243132  | 0.0
|
| -5729.5779513082325 | 572.9577951308232   | -332.31552117587745 | -332.31552117587745 | -332.31553210418014 | -5156.620156
177409  |
+---------------------+---------------------+---------------------+---------------------+---------------------+-------------
--------+
```

E

- 命令格式

```
double e()
```

- 命令说明

  返回 `e` 的值。此函数为MaxCompute 2.0扩展函数。

- 返回值说明

  返回DOUBLE类型。

- 示例

```
--返回2.718281828459045。
select e();
```

FACTORIAL

- 命令格式

```
bigint factorial(<number>)
```

- 命令说明

  返回number的阶乘。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。BIGINT、INT、SMALLINT、TINYINT类型，且在[0,20]之间。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - number值为0时，返回1。
  - number值为NULL或[0,20]之外的值，返回NULL。

- 示例

- 静态数据示例

```
--返回120。5!=5*4*3*2*1=120
select factorial(5);
--返回1。
select factorial(0);
--返回NULL。
select factorial(null);
```

- 表数据示例

  基于示例数据，计算int_data和bigint_data两列的阶乘，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select factorial(int_data) as int_new, factorial(bigint_data) as bigint_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------+------------+
| int_new    | bigint_new |
+------------+------------+
| NULL       | NULL       |
| NULL       | NULL       |
| 1          | NULL       |
| NULL       | 24         |
| 120        | NULL       |
| NULL       | 720        |
| NULL       | NULL       |
| NULL       | 1          |
| 362880     | NULL       |
| NULL       | 3628800    |
+------------+------------+
```

FORMAT_NUMBER

- 命令格式

```
string format_number(float|double|decimal <expr1>, <expr2>)
```

- 命令说明

  将数字转化为指定格式的字符串。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - expr1：必填。需要格式化的数据。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL、STRING类型。
  - expr2：必填。需要转化的目标格式。指定需要保留小数位数。也可以为类似 `#,###,###.##` 格式描述。

- 返回值说明

  返回STRING类型数据。返回规则如下：
  - 如果expr2>0，则四舍五入到小数点后指定位数。
  - 如果expr2=0，则没有小数点或小数部分。
  - 如果expr2<0，或者expr2>340，则返回报错。
  - expr1或expr2值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回5.230。
select format_number(5.230134523424545456,3);
--返回12,332.123。
select format_number(12332.123456, '#,###,###,###.###');
--返回NULL。
select format_number(null,3);
```

- 表数据示例

  基于示例数据，将所有列保留指定小数位数，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select format_number(int_data, 1) as int_new, format_number(bigint_data, 1) as bigint_new, format_number(double_data, 2) as
double_new, format_number(decimal_data, 1) as decimal_new, format_number(float_data, 0) as float_new, format_number(string_d
ata, 1) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+---------+------------+------------+-------------+-----------+------------+
| int_new | bigint_new | double_new | decimal_new | float_new | string_new |
+---------+------------+------------+-------------+-----------+------------+
| NULL    | -10.0      | 0.53       | 0.5         | 1         | 10.0       |
| -20.0   | NULL       | -0.10      | -0.1        | -0        | -10.0      |
| 0.0     | -1.0       | NULL       | 20.5        | -1        | 30.0       |
| -40.0   | 4.0        | 0.89       | NULL        | 1         | -30.0      |
| 5.0     | -50.0      | -1.00      | -1.0        | NULL      | 50.0       |
| -60.0   | 6.0        | 1.50       | 1.5         | 2         | -50.0      |
| -1.0    | -70.0      | -7.50      | -7.5        | -8        | NULL       |
| -80.0   | 1.0        | -10.20     | -10.2       | -10       | -1.0       |
| 9.0     | -90.0      | 2.58       | 2.6         | 3         | 0.0        |
| -100.0  | 10.0       | -5.80      | -5.8        | -6        | -90.0      |
+---------+------------+------------+-------------+-----------+------------+
```

HEX

- 命令格式

```
string hex(<number>)
```

- 命令说明

  将数值或字符串转换为十六进制格式。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL、STRING类型。

- 返回值说明

  返回STRING类型。返回规则如下：

  - number值非0或非NULL时，返回STRING类型。
  - number值为0时，返回0。
  - number值为NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回0。
select hex(0);
--返回616263。
select hex('abc');
--返回11。
select hex(17);
--返回3137。
select hex('17');
--返回报错。
select hex(null);
```

- 表数据示例

  基于示例数据，将所有列转换为十六进制表示形式，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select hex(int_data) as int_new, hex(bigint_data) as bigint_new, hex(double_data) as double_new, hex(decimal_data) as decima
l_new, hex(float_data) as float_new, hex(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------------+------------------+------------+-------------+------------+------------+
| int_new          | bigint_new       | double_new | decimal_new | float_new  | string_new |
+------------------+------------------+------------+-------------+------------+------------+
| NULL             | FFFFFFFFFFFFFF6 | 302E353235 | 302E353235  | 302E353235 | 3130       |
| FFFFFFFFFFFFFEC | NULL             | 2D302E31   | 2D302E31    | 2D302E31   | 2D3130     |
| 0                | FFFFFFFFFFFFFFF | NULL       | 32302E3435  | 2D31       | 3330       |
| FFFFFFFFFFFFFD8 | 4                | 302E3839   | NULL        | 302E3839   | 2D3330     |
| 5                | FFFFFFFFFFFFFCE | 2D312E30   | 2D31        | NULL       | 3530       |
| FFFFFFFFFFFFFC4 | 6                | 312E35     | 312E35      | 312E35     | 2D3530     |
| FFFFFFFFFFFFFFF | FFFFFFFFFFFFFBA | 2D372E35   | 2D372E35    | 2D372E35   | NULL       |
| FFFFFFFFFFFFFB0 | 1                | 2D31302E32 | 2D31302E32  | 2D31302E32 | 2D31       |
| 9                | FFFFFFFFFFFFFA6 | 322E3538   | 322E3538    | 322E3538   | 30         |
| FFFFFFFFFFFFF9C | A                | 2D352E38   | 2D352E38    | 2D352E38   | 2D3930     |
+------------------+------------------+------------+-------------+------------+------------+
```

LOG2

- 命令格式

```
double log2(<number>)
```

- 命令说明

  返回以2为底，number的对数。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL、STRING类型。

- 返回值说明

  返回DOUBLE类型。number值为0、负数或NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回NULL。
select log2(null);
--返回NULL。
select log2(0);
--返回3.0。
select log2(8);
```

- 表数据示例

  基于示例数据，计算所有列以2为底的对数，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select log2(int_data) as int_new, log2(bigint_data) as bigint_new, log2(double_data) as double_new, log2(decimal_data) as de
cimal_new, log2(float_data) as float_new, log2(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+--------------------+--------------------+---------------------+--------------------+---------------------+--------------
------+
| int_new            | bigint_new         | double_new          | decimal_new        | float_new           | string_new
|
+--------------------+--------------------+---------------------+--------------------+---------------------+--------------
------+
| NULL               | NULL               | -0.929610672108602  | -0.929610672108602 | -0.9296107376258038 | 3.32192809488
73626 |
| NULL               | NULL               | NULL                | NULL               | NULL                | NULL
|
| NULL               | NULL               | NULL                | 4.354028938054387  | NULL                | 4.90689059560
8519  |
| NULL               | 2.0                | -0.16812275880832692 | NULL               | -0.16812278199699915 | NULL
|
| 2.321928094887362  | NULL               | NULL                | NULL               | NULL                | 5.64385618977
4724  |
| NULL               | 2.584962500721156  | 0.5849625007211562  | 0.5849625007211562 | 0.5849625007211562  | NULL
|
| NULL               | NULL               | NULL                | NULL               | NULL                | NULL
|
| NULL               | 0.0                | NULL                | NULL               | NULL                | NULL
|
| 3.1699250014423126 | NULL               | 1.3673710656485296  | 1.3673710656485296 | 1.367371022986166   | NULL
|
| NULL               | 3.3219280948873626 | NULL                | NULL               | NULL                | NULL
|
+--------------------+--------------------+---------------------+--------------------+---------------------+--------------
------+
```

LOG10

- 命令格式

```
double log10(<number>)
```

- 命令说明

  返回以10为底，number的对数。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL、STRING类型。

- 返回值说明

  返回DOUBLE类型。number值为0、负数或NULL时，返回NULL。

- 示例
  - 静态数据示例

```
--返回NULL。
select log10(null);
--返回NULL。
select log10(0);
--返回0.9030899869919435。
select log10(8);
```

- 表数据示例

  基于示例数据，计算所有列以10为底的对数，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select log10(int_data) as int_new, log10(bigint_data) as bigint_new, log10(double_data) as double_new, log10(decimal_data) a
s decimal_new, log10(float_data) as float_new, log10(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+--------------------+--------------------+--------------------+--------------------+----------------------+-------------
-------+
| int_new            | bigint_new         | double_new         | decimal_new        | float_new            | string_new
|
+--------------------+--------------------+--------------------+--------------------+----------------------+-------------
-------+
| NULL               | NULL               | -0.2798406965940431 | -0.2798406965940431 | -0.27984071631668606  | 1.0
|
| NULL               | NULL               | NULL               | NULL               | NULL                 | NULL
|
| NULL               | NULL               | NULL               | 1.3106933123433606  | NULL                 | 1.4771212547
196624 |
| NULL               | 0.6020599913279623 | -0.0506099933550872 | NULL               | -0.050610000335573106 | NULL
|
| 0.6989700043360187 | NULL               | NULL               | NULL               | NULL                 | 1.6989700043
360185 |
| NULL               | 0.7781512503836435 | 0.17609125905568124 | 0.17609125905568124 | 0.17609125905568124  | NULL
|
| NULL               | NULL               | NULL               | NULL               | NULL                 | NULL
|
| NULL               | 0.0                | NULL               | NULL               | NULL                 | NULL
|
| 0.9542425094393249 | NULL               | 0.4116197059632301  | 0.4116197059632301  | 0.411619693120579    | NULL
|
| NULL               | 1.0                | NULL               | NULL               | NULL                 | NULL
|
+--------------------+--------------------+--------------------+--------------------+----------------------+-------------
-------+
```

PI

- 命令格式

```
double pi()
```

- 命令说明

  返回π的值。此函数为MaxCompute 2.0扩展函数。

- 返回值说明

  返回DOUBLE类型。

- 示例

```
--返回3.141592653589793。
select pi();
```

RADIANS

- 命令格式

```
double radians(<number>)
```

- 命令说明

  将角度转换为弧度。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL、STRING类型。

- 返回值说明

  返回DOUBLE类型。number值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--返回1.5707963267948966。
select radians(90);
--返回0.0。
select radians(0);
--返回NULL。
select radians(null);
```

- 表数据示例

  基于示例数据，计算所有列转换为弧度，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select radians(int_data) as int_new, radians(bigint_data) as bigint_new, radians(double_data) as double_new, radians(decimal
_data) as decimal_new, radians(float_data) as float_new, radians(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------+
-----------------------+
| int_new               | bigint_new            | double_new            | decimal_new           | float_new             |
string_new            |
+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------+
-----------------------+
| NULL                  | -0.17453292519943295  | 0.00916297857297023   | 0.00916297857297023   | 0.009162978156851308  |
0.17453292519943295   |
| -0.3490658503988659   | NULL                  | -0.0017453292519943296 | -0.0017453292519943296 | -0.0017453292780017621 |
-0.17453292519943295  |
| 0.0                   | -0.017453292519943295 | NULL                  | 0.3569198320328404    | -0.017453292519943295 |
0.5235987755982988    |
| -0.6981317007977318   | 0.06981317007977318   | 0.015533430342749534  | NULL                  | 0.015533430093078181  |
-0.5235987755982988   |
| 0.08726646259971647   | -0.8726646259971648   | -0.017453292519943295 | -0.017453292519943295 | NULL                  |
0.8726646259971648    |
| -1.0471975511965976   | 0.10471975511965977   | 0.02617993877991494   | 0.02617993877991494   | 0.02617993877991494   |
-0.8726646259971648   |
| -0.017453292519943295 | -1.2217304763960306   | -0.1308996938995747   | -0.1308996938995747   | -0.1308996938995747   |
NULL                  |
| -1.3962634015954636   | 0.017453292519943295  | -0.17802358370342158  | -0.17802358370342158  | -0.17802358037447025  |
-0.017453292519943295 |
| 0.15707963267948966   | -1.5707963267948966   | 0.045029494701453704  | 0.045029494701453704  | 0.04502949336987316   |
0.0                   |
| -1.7453292519943295   | 0.17453292519943295   | -0.10122909661567112  | -0.10122909661567112  | -0.10122909994462247  |
-1.5707963267948966   |
+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------+
-----------------------+
```

SIGN

- 命令格式

```
double sign(<number>)
```

- 命令说明

  获取输入参数的符号。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL、STRING类型。

- 返回值说明

  返回DOUBLE类型。返回规则如下：

  - number值为正数时，返回1.0。
  - number值为负数时，返回-1.0。
  - number值为0时，返回0.0。
  - number值为NULL时，返回NULL。

- 示例

- 静态数据示例

```
--返回-1.0。
select sign(-2.5);
--返回1.0。
select sign(2.5);
--返回0.0。
select sign(0);
--返回NULL。
select sign(null);
```

- 表数据示例

  基于示例数据，获取所有列数据的符号，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select sign(int_data) as int_new, sign(bigint_data) as bigint_new, sign(double_data) as double_new, sign(decimal_data) as de
cimal_new, sign(float_data) as float_new, sign(string_data) as string_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------+------------+------------+-------------+------------+------------+
| int_new    | bigint_new | double_new | decimal_new | float_new  | string_new |
+------------+------------+------------+-------------+------------+------------+
| NULL       | -1.0       | 1.0        | 1           | 1.0        | 1.0        |
| -1.0       | NULL       | -1.0       | -1          | -1.0       | -1.0       |
| 0.0        | -1.0       | NULL       | 1           | -1.0       | 1.0        |
| -1.0       | 1.0        | 1.0        | NULL        | 1.0        | -1.0       |
| 1.0        | -1.0       | -1.0       | -1          | NULL       | 1.0        |
| -1.0       | 1.0        | 1.0        | 1           | 1.0        | -1.0       |
| -1.0       | -1.0       | -1.0       | -1          | -1.0       | NULL       |
| -1.0       | 1.0        | -1.0       | -1          | -1.0       | -1.0       |
| 1.0        | -1.0       | 1.0        | 1           | 1.0        | 0.0        |
| -1.0       | 1.0        | -1.0       | -1          | -1.0       | -1.0       |
+------------+------------+------------+-------------+------------+------------+
```

SHIFTLEFT

- 命令格式

```
int shiftleft(tinyint|smallint|int <number1>, int <number2>)
bigint shiftleft(bigint <number1>, int <number2>)
```

- 命令说明

  按位左移（<<）。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - number1：必填。TINYINT、SMALLINT、INT或BIGINT类型。
  - number2：必填。INT类型。
- 返回值说明

  返回INT或BIGINT类型。返回规则如下
  - number1非TINYINT、SMALLINT、INT或BIGINT类型时，返回报错。
  - number2非INT类型时，返回报错。
  - number1或number2值为NULL时，返回NULL。
- 示例
  - 静态数据示例

```
--返回4。1的二进制左移2位（1<<2,0001左移两位是0100）。
select shiftleft(1,2);
--返回32。4的二进制左移3位（4<<3,0100左移3位是100000）。
select shiftleft(4,3);
--返回NULL。
select shiftleft(null,2);
```

- 表数据示例

  基于示例数据，将int_data和bigint_data列按位左移，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select shiftleft(int_data, 1) as int_new, shiftleft(bigint_data, 1) as bigint_new from mf_math_fun_t;
```

  返回结果如下。

```
+---------+------------+
| int_new | bigint_new |
+---------+------------+
| NULL    | -20        |
| -40     | NULL       |
| 0       | -2         |
| -80     | 8          |
| 10      | -100       |
| -120    | 12         |
| -2      | -140       |
| -160    | 2          |
| 18      | -180       |
| -200    | 20         |
+---------+------------+
```

SHIFTRIGHT

- 命令格式

```
int shiftright(tinyint|smallint|int <number1>, int <number2>)
bigint shiftright(bigint <number1>, int <number2>)
```

- 命令说明

  按位右移（>>）。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - number1：必填。TINYINT、SMALLINT、INT或BIGINT类型。
  - number2：必填。INT类型。
- 返回值说明

  返回INT或BIGINT类型。返回规则如下：
  - number1非TINYINT、SMALLINT、INT或BIGINT类型时，返回报错。
  - number2非INT类型时，返回报错。
  - number1或number2值为NULL时，返回NULL。
- 示例
  - 静态数据示例

```
--返回1。4的二进制右移2位（4>>2,0100右移两位是0001）。
select shiftright(4,2);
--返回4。32的二进制右移3位（32>>3,100000右移3位是0100）。
select shiftright(32,3);
--返回NULL。
select shiftright(null,3);
```

- 表数据示例

  基于示例数据，将int_data和bigint_data列按位右移，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select shiftright(int_data, 1) as int_new, shiftright(bigint_data, 1) as bigint_new from mf_math_fun_t;
```

  返回结果如下。

```
+---------+------------+
| int_new | bigint_new |
+---------+------------+
| NULL    | -5         |
| -10     | NULL       |
| 0       | -1         |
| -20     | 2          |
| 2       | -25        |
| -30     | 3          |
| -1      | -35        |
| -40     | 0          |
| 4       | -45        |
| -50     | 5          |
+---------+------------+
```

SHIFTRIGHTUNSIGNED

- 命令格式

```
int shiftrightunsigned(tinyint|smallint|int <number1>, int <number2>)
bigint shiftrightunsigned(bigint <number1>, int <number2>)
```

- 命令说明

  无符号按位右移（>>>）。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - number1：必填。TINYINT、SMALLINT、INT或BIGINT整型数据。
  - number2：必填。INT整型数据。
- 返回值说明

  返回INT或BIGINT类型。返回规则如下：
  - number1非TINYINT、SMALLINT、INT或BIGINT类型时，返回报错。
  - number2非INT类型时，返回报错。
  - number1或number2值为NULL时，返回NULL。
- 示例
  - 静态数据示例

```
--返回2。8的二进制无符号右移2位(8>>>2,1000右移两位是0010)。
select shiftrightunsigned(8,2);
--返回1073741820。-14的二进制右移2位(-14>>>2, 11111111 11111111 11111111 11110010右移2位是00111111 11111111 11111111 11111100)。
select shiftrightunsigned(-14,2);
--返回NULL。
select shiftrightunsigned(-14,null);
```

- 表数据示例

  基于示例数据，返回int_data和bigint_data列的无符号按位右移值，命令示例如下。

```
--开启2.0新类型。此命令需要与SQL语句一起提交。
set odps.sql.type.system.odps2=true;
select shiftrightunsigned(int_data, 1) as int_new, shiftrightunsigned(bigint_data, 1) as bigint_new from mf_math_fun_t;
```

  返回结果如下。

```
+------------+---------------------+
| int_new    | bigint_new          |
+------------+---------------------+
| NULL       | 9223372036854775803 |
| 2147483638 | NULL                |
| 0          | 9223372036854775807 |
| 2147483628 | 2                   |
| 2          | 9223372036854775783 |
| 2147483618 | 3                   |
| 2147483647 | 9223372036854775773 |
| 2147483608 | 0                   |
| 4          | 9223372036854775763 |
| 2147483598 | 5                   |
+------------+---------------------+
```

UNHEX

- 命令格式

```
binary unhex(string <number>)
```

- 命令说明

  返回十六进制字符串所代表的字符串。此函数为MaxCompute 2.0扩展函数。

- 参数说明

  number：必填。为十六进制字符串。

- 返回值说明

  返回BINARY类型。返回规则如下：

  - number值为0时，返回报错。
  - number值为NULL时，返回NULL。

- 示例

```
--返回abc。
select unhex('616263');
--返回abc。
select unhex(616263);
--返回NULL。
select unhex(null);
```

WIDTH_BUCKET

- 命令格式

```
width_bucket(numeric <expr>, numeric <min_value>, numeric <max_value>, int <num_buckets>)
```

- 命令说明

  设定分组范围的最小值、最大值和分组个数，构建指定个数的大小相同的分组，返回指定字段值落入的分组编号。支持的数据类型为BIGINT、INT、FLOAT、DOUBLE、DECIMAL以及数据类型2.0的DECIMAL(precision,scale)，详情请参见2.0数据类型版本。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - expr：必填。需要判断分组编号的字段。
  - min_value：必填。分组范围最小值。
  - max_value：必填。分组范围最大值，最大值必须比最小值大。
  - num_buckets：必填。分组个数，必须大于0。

- 返回值说明

  返回值为BIGINT类型，范围为0到设定的分组个数+1。返回规则如下：

  - expr值小于min_value时，返回0。
  - expr大于max_value时，返回设定的分组个数+1。

- expr值为NULL时，分组编号返回值也为NULL。其他情况返回字段值落入的分组编号。字段值对应的分组编号取值规则为 `floor( num_buckets * (expr - min_value)/(max_value - min_value) + 1)` 。
- min_value、max_value或num_buckets值为NULL时，返回NULL。

- 示例
  - 示例1：入参无NULL。命令示例如下。

```
select key,value,width_bucket(value,100,500,5) as value_group
from values
    (1,99),
    (2,100),
    (3,199),
    (4,200),
    (5,499),
    (6,500),
    (7,501),
    (8,NULL)
as t(key,value);
```

返回结果如下。

```
+-------+--------+-------------+
| key   | value  | value_group |
+-------+--------+-------------+
| 1     | 99     | 0           |
| 2     | 100    | 1           |
| 3     | 199    | 2           |
| 4     | 200    | 2           |
| 5     | 499    | 5           |
| 6     | 500    | 6           |
| 7     | 501    | 6           |
| 8     | \N     | \N          |
+-------+--------+-------------+
```

  - 示例2：入参存在NULL。命令示例如下。

```
select key,value,width_bucket(value,100,500,null) as value_group
from values
    (1,99),
    (2,100),
    (3,199),
    (4,200),
    (5,499),
    (6,500),
    (7,501),
    (8,NULL)
as t(key,value);
```

返回结果如下。

```
+------+-------+-------------+
| key  | value | value_group |
+------+-------+-------------+
| 1    | 99    | NULL        |
| 2    | 100   | NULL        |
| 3    | 199   | NULL        |
| 4    | 200   | NULL        |
| 5    | 499   | NULL        |
| 6    | 500   | NULL        |
| 7    | 501   | NULL        |
| 8    | NULL  | NULL        |
+------+-------+-------------+
```

3.9.5. 窗口函数

您可以在MaxCompute SQL中使用窗口函数对指定开窗列的数据灵活地进行分析处理工作。本文为您提供MaxCompute SQL支持的窗口函数的命令格式、参数说明及示例，指导您使用窗口函数完成开发。

MaxCompute SQL支持的窗口函数如下。

函数	功能
ROW_NUMBER	计算行号。从1开始递增。
RANK	计算排名。排名可能不连续。

函数	功能
DENSE_RANK	计算排名。排名是连续的。
PERCENT_RANK	计算排名。输出百分比格式。
CUME_DIST	计算累计分布。
NTILE	将数据顺序切分成N等份，返回数据所在等份的编号（从1到N）。
LAG	取当前行往前（朝分区头部方向）第N行数据的值。
LEAD	取当前行往后（朝分区尾部方向）第N行数据的值。
FIRST_VALUE	取当前行所对应窗口的第一条数据的值。
LAST_VALUE	取当前行所对应窗口的最后一条数据的值。
NTH_VALUE	取当前行所对应窗口的第N条数据的值。
CLUSTER_SAMPLE	用户随机抽样。返回True表示该行数据被抽中。
COUNT	计算窗口中的记录数。
MIN	计算窗口中的最小值。
MAX	计算窗口中的最大值。
AVG	对窗口中的数据求平均值。
SUM	对窗口中的数据求和。
MEDIAN	计算窗口中的中位数。
STDDEV	计算总体标准差。是STDDEV_POP的别名。
STDDEV_SAMP	计算样本标准差。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在 `select` 语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

窗口函数语法

窗口函数的语法声明如下。

```
<function_name>([distinct][<expression> [, ...]]) over (<window_definition>)
<function_name>([distinct][<expression> [, ...]]) over <window_name>
```

- function_name：内建窗口函数、聚合函数或用户自定义聚合函数UDAF。
- expression：函数格式，具体格式以实际函数语法为准。
- windowing_definition：窗口定义。详细语法格式请参见windowing_definition。
- window_name：窗口名称。您可以使用 `window` 关键字自定义窗口，为windowing_definition定义名称。自定义语句（named_window_def）如下：

  ```
  window <window_name> as (<window_definition>)
  ```

  自定义语句在SQL中的位置如下：

  ```
  select ... from ... [where ...] [group by ...] [having ...] named_window_def [order by ...] [limit ...]
  ```

windowing_definition

windowing_definition的语法声明如下。

```
--partition_clause:
[partition by <expression> [, ...]]
--orderby_clause:
[order by <expression> [asc|desc][nulls {first|last}] [, ...]]
[<frame_clause>]
```

在SELECT语句中加入窗口函数，计算窗口函数的结果时，数据会按照窗口定义中的 `partition by` 和 `order by` 语句进行分区和排序。如果没有 `partition by` 语句，则仅有一个分区，包含全部数据。如果没有 `order by` 语句，则分区内的数据会按照任意顺序排布，最终生成一个确定的数据流。之后对于每一行数据（当前行），会按照窗口定义中的 `frame_clause` 从数据流中截取一段数据，构成当前行的窗口。窗口函数会根据窗口中包含的数据，计算得到窗口函数针对当前行对应的输出结果。

- partition by <expression> [, ...]：可选。指定分区。分区列的值相同的行被视为在同一个窗口内。详细格式请参见表操作。
- order by <expression> [asc|desc][nulls {first|last}] [, ...]：可选。指定数据在一个窗口内如何排序。

> **说明** 当遇到相同的 `order by` 值时，排序结果不稳定。为减少随机性，应当尽可能保持 `order by` 值的唯一性。

- frame_clause：可选。用于确定数据边界，更多frame_clause信息，请参见frame_clause。

frame_clause

frame_clause的语法声明如下。

```
--格式一。
{ROWS|RANGE|GROUPS} <frame_start> [<frame_exclusion>]
--格式二。
{ROWS|RANGE|GROUPS} between <frame_start> and <frame_end> [<frame_exclusion>]
```

frame_clause是一个闭区间，用于确定数据边界，包含frame_start和frame_end位置的数据行。

- ROWS|RANGE|GROUPS：必填。frame_clause的类型，各类型的frame_start和frame_end实现规则不相同。其中：
  - ROWS类型：通过数据行数确定数据边界。
  - RANGE类型：通过比较 `order by` 列值的大小关系来确定数据边界。一般在窗口定义中会指定 `order by` ，未指定 `order by` 时，一个分区中的所有数据行具有相同的 `order by` 列值。NULL与NULL被认为是相等的。
  - GROUPS：一个分区中所有具有相同 `order by` 列值的数据组成一个GROUP。未指定 `order by` 时，分区中的所有数据组成一个GROUP。NULL与NULL被认为是相等的。
- frame_start、frame_end：表示窗口的起始和终止边界。frame_start必填。frame_end可选，省略时默认值为CURRENT ROW。

  frame_start确定的位置必须在frame_end确定的位置的前面，或者等于frame_end的位置，即frame_start相比frame_end更靠近分区头部。分区头部是指数据按窗口定义中的 `order by` 语句排序之后第1行数据的位置。ROWS、RANGE、GROUPS类型对应的取值范围及实现逻辑如下。

frame_clause类型	frame_start/frame_end取值	说明
ROWS、RANGE、GROUPS	UNBOUNDED PRECEDING	表示分区的第一行，从1开始计数。
	UNBOUNDED FOLLOWING	表示分区的最后一行。
ROWS	CURRENT ROW	指当前行的位置。每一行数据都会对应一个窗口函数的结果值，当前行是指在给哪一行数据计算窗口函数的结果。
	offset PRECEDING	指从当前行位置，向分区头部位置移动 `offset` 行的位置。例如 `0 PRECEDING` 指当前行， `1 PRECEDING` 指前一行。 `offset` 必须为非负整数。
	offset FOLLOWING	指从当前行位置，向分区尾部移动 `offset` 行的位置。例如 `0 FOLLOWING` 指当前行， `1 FOLLOWING` 指下一行。 `offset` 必须为非负整数。
	CURRENT ROW	◦ 作为frame_start时，指第一条与当前行具有相同 `order by` 列值的数据的位置。 ◦ 作为frame_end时，指最后一条与当前行具有相同 `order by` 列值的数据的位置。

frame_clause类型	frame_start/frame_end取值	说明
RANGE	offset PRECEDING	frame_start和frame_end的位置与 `order by` 的顺序相关。假设窗口按照X进行排序，Xi表示第i行数据对应的X值，Xc表示当前行数据对应X值。位置说明如下： ○ 当 `order by` 为升序时： ■ frame_start：指第一条满足 `Xc - Xi <= offset` 数据的位置。 ■ frame_end：指最后一条满足 `Xc - Xi >= offset` 数据的位置。 ○ 当 `order by` 为降序时： ■ frame_start：指第一条满足 `Xi - Xc <= offset` 数据的位置。 ■ frame_end：指最后一条满足 `Xi - Xc >= offset` 数据的位置。 `order by` 的列支持的数据类型为：TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、DATETIME、DATE、TIMESTAMP。 日期类型数据的 `offset` 语法如下： ○ `N` ：表示N天或N秒。非负整数。对于DATETIME和TIMESTAMP，表示N秒；对于DATE，表示N天。 ○ `interval 'N' {YEAR\MONTH\DAY\HOUR\MINUTE\SECOND}` ：表示N年/月/日/小时/分钟/秒。例如 `INTERVAL '3' YEAR` 表示3年。 ○ `INTERVAL 'N-M' YEAR TO MONTH` ：表示N年M月。例如 `INTERVAL '1-3' YEAR TO MONTH` 表示1年3个月。 ○ `INTERVAL 'D[ H[:M[:S[:N]]]]' DAY TO SECOND` ：表示D天H小时M分钟S秒N纳秒。例如 `INTERVAL '1 2:3:4:5' DAY TO SECOND` 表示1天2小时3分钟4秒5纳秒。
	offset FOLLOWING	frame_start和frame_end的位置与 `order by` 的顺序相关。假设窗口按照X进行排序，Xi表示第i行数据对应的X值，Xc表示当前行数据对应X值。位置说明如下： ○ 当 `order by` 为升序时： ■ frame_start：指第一条满足 `Xi - Xc >= offset` 数据的位置。 ■ frame_end：指最后一条满足 `Xi - Xc <= offset` 数据的位置。 ○ 当 `order by` 为降序时： ■ frame_start：指第一条满足 `Xc - Xi >= offset` 数据的位置。 ■ frame_end：指最后一条满足 `Xc - Xi <= offset` 数据的位置。
GROUPS	CURRENT ROW	○ 作为frame_start时，指当前行所属GROUP的第一条数据。 ○ 作为frame_end时，指当前行所属GROUP的最后一行数据。
	offset PRECEDING	○ 作为frame_start时，指从当前行所属GROUP开始，朝分区头部移动 `offset` 个GROUP之后，所在GROUP的第一条数据的位置。 ○ 作为frame_end时，指从当前行所属GROUP开始，朝分区头部移动 `offset` 个GROUP之后，所在GROUP的最后一条数据的位置。 **说明** frame_start不能设置为UNBOUNDED FOLLOWING，frame_end不能设置为UNBOUNED PRECEDING。
	offset FOLLOWING	○ 作为frame_start时，指从当前行所属GROUP开始，朝分区尾部移动 `offset` 个GROUP之后，所在GROUP的第一条数据的位置。 ○ 作为frame_end时，指从当前行所属GROUP开始，朝分区尾部移动 `offset` 个GROUP之后，所在GROUP的最后一条数据的位置。 **说明** frame_start不能设置为UNBOUNDED FOLLOWING，frame_end不能设置为UNBOUNED PRECEDING。

- frame_exclusion：可选。用于从窗口中剔除一部分数据。取值范围如下：
  - EXCLUDE NO OTHERS：表示不剔除任何数据。
  - EXCLUDE CURRENT ROW：表示剔除当前行。
  - EXCLUDE GROUP：表示剔除整个GROUP，即分区中与当前行具有相同 `order by` 值的所有数据。
  - EXCLUDE TIES：表示剔除整个GROUP，但保留当前行。

默认frame_clause

未显示设置frame_clause时，MaxCompute会使用默认的frame_clause来决定窗口所包含数据的边界。默认的frame_clause为：

- 当开启Hive兼容模式（`set odps.sql.hive.compatible=true;`）时，默认的frame_clause如下，与大部分SQL系统相同。

```
RANGE between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE NO OTHERS
```

- 当关闭Hive兼容模式（`set odps.sql.hive.compatible=false;`），同时窗口定义中指定了`order by`语句，且窗口函数为AVG、COUNT、MAX、MIN、STDDEV、STEDEV_POP、STDDEV_SAMP或SUM时，会使用ROWS类型的默认frame_clause。

```
ROWS between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE NO OTHERS
```

窗口边界示例

假设表tbl结构为`pid: bigint, oid: bigint, rid: bigint`，表中包含如下数据：

```
+------------+------------+------------+
| pid        | oid        | rid        |
+------------+------------+------------+
| 1          | NULL       | 1          |
| 1          | NULL       | 2          |
| 1          | 1          | 3          |
| 1          | 1          | 4          |
| 1          | 2          | 5          |
| 1          | 4          | 6          |
| 1          | 7          | 7          |
| 1          | 11         | 8          |
| 2          | NULL       | 9          |
| 2          | NULL       | 10         |
+------------+------------+------------+
```

您可以将如下SQL语句中的`...`替换为窗口定义语句windowing_definition来展示每一条数据所对应的窗口包含的数据列表：

> **说明** Window列为NULL时，不包含任何数据。

- ROW类型窗口
  - 窗口定义1

```
partition by pid order by oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [1]    |
| 1          | NULL       | 2          | [1, 2] |
| 1          | 1          | 3          | [1, 2, 3] |
| 1          | 1          | 4          | [1, 2, 3, 4] |
| 1          | 2          | 5          | [1, 2, 3, 4, 5] |
| 1          | 4          | 6          | [1, 2, 3, 4, 5, 6] |
| 1          | 7          | 7          | [1, 2, 3, 4, 5, 6, 7] |
| 1          | 11         | 8          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 2          | NULL       | 9          | [9]    |
| 2          | NULL       | 10         | [9, 10] |
+------------+------------+------------+--------+
```

- 窗口定义2

```
partition by pid order by oid ROWS between UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid ROWS between UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | NULL       | 2          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | 1          | 3          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | 1          | 4          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | 2          | 5          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | 4          | 6          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | 7          | 7          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | 11         | 8          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 2          | NULL       | 9          | [9, 10] |
| 2          | NULL       | 10         | [9, 10] |
+------------+------------+------------+--------+
```

- 窗口定义3

```
partition by pid order by oid ROWS between 1 FOLLOWING and 3 FOLLOWING
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid ROWS between 1 FOLLOWING and 3 FOLLOWING) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [2, 3, 4] |
| 1          | NULL       | 2          | [3, 4, 5] |
| 1          | 1          | 3          | [4, 5, 6] |
| 1          | 1          | 4          | [5, 6, 7] |
| 1          | 2          | 5          | [6, 7, 8] |
| 1          | 4          | 6          | [7, 8] |
| 1          | 7          | 7          | [8]    |
| 1          | 11         | 8          | NULL   |
| 2          | NULL       | 9          | [10]   |
| 2          | NULL       | 10         | NULL   |
+------------+------------+------------+--------+
```

- 窗口定义4

```
partition by pid order by oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE CURRENT ROW
--SQL语句如下。
select pid,
oid,
rid,
collect_list(rid) over(partition by pid order by
oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE CURRENT ROW) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | NULL   |
| 1          | NULL       | 2          | [1]    |
| 1          | 1          | 3          | [1, 2] |
| 1          | 1          | 4          | [1, 2, 3] |
| 1          | 2          | 5          | [1, 2, 3, 4] |
| 1          | 4          | 6          | [1, 2, 3, 4, 5] |
| 1          | 7          | 7          | [1, 2, 3, 4, 5, 6] |
| 1          | 11         | 8          | [1, 2, 3, 4, 5, 6, 7] |
| 2          | NULL       | 9          | NULL   |
| 2          | NULL       | 10         | [9]    |
+------------+------------+------------+--------+
```

- 窗口定义5

```
partition by pid order by oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE GROUP
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE GROUP) as wind
ow from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | NULL   |
| 1          | NULL       | 2          | NULL   |
| 1          | 1          | 3          | [1, 2] |
| 1          | 1          | 4          | [1, 2] |
| 1          | 2          | 5          | [1, 2, 3, 4] |
| 1          | 4          | 6          | [1, 2, 3, 4, 5] |
| 1          | 7          | 7          | [1, 2, 3, 4, 5, 6] |
| 1          | 11         | 8          | [1, 2, 3, 4, 5, 6, 7] |
| 2          | NULL       | 9          | NULL   |
| 2          | NULL       | 10         | NULL   |
+------------+------------+------------+--------+
```

- 窗口定义6

```
partition by pid order by oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE TIES
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid ROWS between UNBOUNDED PRECEDING and CURRENT ROW EXCLUDE TIES) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [1]    |
| 1          | NULL       | 2          | [2]    |
| 1          | 1          | 3          | [1, 2, 3] |
| 1          | 1          | 4          | [1, 2, 4] |
| 1          | 2          | 5          | [1, 2, 3, 4, 5] |
| 1          | 4          | 6          | [1, 2, 3, 4, 5, 6] |
| 1          | 7          | 7          | [1, 2, 3, 4, 5, 6, 7] |
| 1          | 11         | 8          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 2          | NULL       | 9          | [9]    |
| 2          | NULL       | 10         | [10]   |
+------------+------------+------------+--------+
```

对比本示例与前一个示例中 `rid` 为2、4、10的 `window` 结果，可以观察到EXCLUDE CURRENT ROW与EXCLUDE GROUP的差异，即对于EXCLUDE GROUP，同一个分区中（ `pid` 相等），与当前行为相同 `oid` 的数据都被剔除了。

- RANGE类型窗口
  - 窗口定义1

```
partition by pid order by oid RANGE between UNBOUNDED PRECEDING and CURRENT ROW
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid RANGE between UNBOUNDED PRECEDING and CURRENT ROW) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [1, 2] |
| 1          | NULL       | 2          | [1, 2] |
| 1          | 1          | 3          | [1, 2, 3, 4] |
| 1          | 1          | 4          | [1, 2, 3, 4] |
| 1          | 2          | 5          | [1, 2, 3, 4, 5] |
| 1          | 4          | 6          | [1, 2, 3, 4, 5, 6] |
| 1          | 7          | 7          | [1, 2, 3, 4, 5, 6, 7] |
| 1          | 11         | 8          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 2          | NULL       | 9          | [9, 10] |
| 2          | NULL       | 10         | [9, 10] |
+------------+------------+------------+--------+
```

CURRENT ROW作为frame_end时，取与当前行具有相同 `order by` 值 `oid` 的最后一条数据，因此 `rid` 为1的记录的 `window` 结果为[1, 2]。

- 窗口定义2

```
partition by pid order by oid RANGE between CURRENT ROW and UNBOUNDED FOLLOWING
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid RANGE between CURRENT ROW and UNBOUNDED FOLLOWING) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | NULL       | 2          | [1, 2, 3, 4, 5, 6, 7, 8] |
| 1          | 1          | 3          | [3, 4, 5, 6, 7, 8] |
| 1          | 1          | 4          | [3, 4, 5, 6, 7, 8] |
| 1          | 2          | 5          | [5, 6, 7, 8] |
| 1          | 4          | 6          | [6, 7, 8] |
| 1          | 7          | 7          | [7, 8] |
| 1          | 11         | 8          | [8]    |
| 2          | NULL       | 9          | [9, 10] |
| 2          | NULL       | 10         | [9, 10] |
+------------+------------+------------+--------+
```

- 窗口定义3

```
partition by pid order by oid RANGE between 3 PRECEDING and 1 PRECEDING
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid RANGE between 3 PRECEDING and 1 PRECEDING) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [1, 2] |
| 1          | NULL       | 2          | [1, 2] |
| 1          | 1          | 3          | NULL   |
| 1          | 1          | 4          | NULL   |
| 1          | 2          | 5          | [3, 4] |
| 1          | 4          | 6          | [3, 4, 5] |
| 1          | 7          | 7          | [6]    |
| 1          | 11         | 8          | NULL   |
| 2          | NULL       | 9          | [9, 10] |
| 2          | NULL       | 10         | [9, 10] |
+------------+------------+------------+--------+
```

`order by` 值 `oid` 为NULL的行，对于 `offset {PRECEDING|FOLLOWING}` ，只要 `offset` 不为UNBOUNDED，则作为frame_start，指向分区中第一条 `order by` 值为NULL的数据；作为frame_end，指向最后一条 `order by` 值为NULL的数据。

- GROUPS类型窗口

窗口定义如下：

```
partition by pid order by oid GROUPS between 2 PRECEDING and CURRENT ROW
--SQL语句如下。
select pid,
       oid,
       rid,
collect_list(rid) over(partition by pid order by
oid GROUPS between 2 PRECEDING and CURRENT ROW) as window from tbl;
```

返回结果如下：

```
+------------+------------+------------+--------+
| pid        | oid        | rid        | window |
+------------+------------+------------+--------+
| 1          | NULL       | 1          | [1, 2] |
| 1          | NULL       | 2          | [1, 2] |
| 1          | 1          | 3          | [1, 2, 3, 4] |
| 1          | 1          | 4          | [1, 2, 3, 4] |
| 1          | 2          | 5          | [1, 2, 3, 4, 5] |
| 1          | 4          | 6          | [3, 4, 5, 6] |
| 1          | 7          | 7          | [5, 6, 7] |
| 1          | 11         | 8          | [6, 7, 8] |
| 2          | NULL       | 9          | [9, 10] |
| 2          | NULL       | 10         | [9, 10] |
+------------+------------+------------+--------+
```

示例数据

为便于理解各函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表emp，并添加数据，命令示例如下：

```
create table if not exists emp
   (empno bigint,
    ename string,
    job string,
    mgr bigint,
    hiredate datetime,
    sal bigint,
    comm bigint,
    deptno bigint);
tunnel upload emp.txt emp;
```

emp.txt中的数据如下：

```
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

ROW_NUMBER

- 命令格式

  ```
  row_number() over([partition_clause] [orderby_clause])
  ```

- 命令说明

  计算当前行在分区中的行号，从1开始递增。

- 参数说明

  请参见windowing_definition。不允许包含frame_clause。

- 返回值说明

  返回BIGINT类型。

- 示例

  将所有职工根据部门（deptno）分组（作为开窗列），每个组内根据薪水（sal）做降序排序，获得职工在自己组内的序号。命令示例如下：

  ```
  select deptno, ename, sal, row_number() over (partition by deptno order by sal desc) as nums from emp;
  ```

  返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | nums       |
+------------+------------+------------+------------+
| 10         | JACCKA     | 5000       | 1          |
| 10         | KING       | 5000       | 2          |
| 10         | CLARK      | 2450       | 3          |
| 10         | WELAN      | 2450       | 4          |
| 10         | TEBAGE     | 1300       | 5          |
| 10         | MILLER     | 1300       | 6          |
| 20         | SCOTT      | 3000       | 1          |
| 20         | FORD       | 3000       | 2          |
| 20         | JONES      | 2975       | 3          |
| 20         | ADAMS      | 1100       | 4          |
| 20         | SMITH      | 800        | 5          |
| 30         | BLAKE      | 2850       | 1          |
| 30         | ALLEN      | 1600       | 2          |
| 30         | TURNER     | 1500       | 3          |
| 30         | MARTIN     | 1250       | 4          |
| 30         | WARD       | 1250       | 5          |
| 30         | JAMES      | 950        | 6          |
+------------+------------+------------+------------+
```

RANK

- 命令格式

```
bigint rank() over ([partition_clause] [orderby_clause])
```

- 命令说明

  计算当前行在分区中按照orderby_clause排序后所处的排名。从1开始计数。

- 参数说明

  partition_clause及orderby_clause：详情请参见windowing_definition。

- 返回值说明

  返回BIGINT类型。返回值可能重复、且不连续。具体的返回值为该行数据所在GROUP的第一条数据的 `ROW_NUMBER()` 值。未指定orderby_clause时，返回结果全为1。

- 示例

  将所有职工根据部门（deptno）分组（作为开窗列），每个组内根据薪水（sal）做降序排序，获得职工自己组内的序号。命令示例如下：

```
select deptno, ename, sal, rank() over (partition by deptno order by sal desc) as nums from emp;
```

  返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | nums       |
+------------+------------+------------+------------+
| 10         | JACCKA     | 5000       | 1          |
| 10         | KING       | 5000       | 1          |
| 10         | CLARK      | 2450       | 3          |
| 10         | WELAN      | 2450       | 3          |
| 10         | TEBAGE     | 1300       | 5          |
| 10         | MILLER     | 1300       | 5          |
| 20         | SCOTT      | 3000       | 1          |
| 20         | FORD       | 3000       | 1          |
| 20         | JONES      | 2975       | 3          |
| 20         | ADAMS      | 1100       | 4          |
| 20         | SMITH      | 800        | 5          |
| 30         | BLAKE      | 2850       | 1          |
| 30         | ALLEN      | 1600       | 2          |
| 30         | TURNER     | 1500       | 3          |
| 30         | MARTIN     | 1250       | 4          |
| 30         | WARD       | 1250       | 4          |
| 30         | JAMES      | 950        | 6          |
+------------+------------+------------+------------+
```

DENSE_RANK

- 命令格式

```
bigint dense_rank() over ([partition_clause] [orderby_clause])
```

- 命令说明

计算当前行在分区中按照orderby_clause排序后所处的排名。从1开始计数。分区中具有相同 `order by` 值的行的排名相等。每当 `order by` 值发生变化时，排名加1。

- 参数说明

  partition_clause及orderby_clause：详情请参见windowing_definition。

- 返回值说明

  返回BIGINT类型。未指定orderby_clause时，返回结果全为1。

- 示例

  将所有职工根据部门（deptno）分组（作为开窗列），每个组内根据薪水（sal）做降序排序，获得职工自己组内的序号。命令示例如下：

```
select deptno, ename, sal, dense_rank() over (partition by deptno order by sal desc) as nums from emp;
```

返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | nums       |
+------------+------------+------------+------------+
| 10         | JACCKA     | 5000       | 1          |
| 10         | KING       | 5000       | 1          |
| 10         | CLARK      | 2450       | 2          |
| 10         | WELAN      | 2450       | 2          |
| 10         | TEBAGE     | 1300       | 3          |
| 10         | MILLER     | 1300       | 3          |
| 20         | SCOTT      | 3000       | 1          |
| 20         | FORD       | 3000       | 1          |
| 20         | JONES      | 2975       | 2          |
| 20         | ADAMS      | 1100       | 3          |
| 20         | SMITH      | 800        | 4          |
| 30         | BLAKE      | 2850       | 1          |
| 30         | ALLEN      | 1600       | 2          |
| 30         | TURNER     | 1500       | 3          |
| 30         | MARTIN     | 1250       | 4          |
| 30         | WARD       | 1250       | 4          |
| 30         | JAMES      | 950        | 5          |
+------------+------------+------------+------------+
```

PERCENT_RANK

- 命令格式

```
double percent_rank() over([partition_clause] [orderby_clause])
```

- 命令说明

  计算当前行在分区中按照orderby_clause排序后的百分比排名。

- 参数说明

  partition_clause及orderby_clause：详情请参见windowing_definition。

- 返回值说明

  返回DOUBLE类型，值域为[0.0, 1.0]。具体的返回值等于 `“(rank - 1) / (partition_row_count - 1)”` ，其中： `rank` 为该行数据的RANK窗口函数的返回结果， `partition_row_count` 为该行数据所属分区的数据行数。当分区中只有一行数据时，输出结果为0.0。

- 示例

  计算员工薪水在组内的百分比排名。命令示例如下：

```
select deptno, ename, sal, percent_rank() over (partition by deptno order by sal desc) as sal_new from emp;
```

返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | sal_new    |
+------------+------------+------------+------------+
| 10         | JACCKA     | 5000       | 0.0        |
| 10         | KING       | 5000       | 0.0        |
| 10         | CLARK      | 2450       | 0.4        |
| 10         | WELAN      | 2450       | 0.4        |
| 10         | TEBAGE     | 1300       | 0.8        |
| 10         | MILLER     | 1300       | 0.8        |
| 20         | SCOTT      | 3000       | 0.0        |
| 20         | FORD       | 3000       | 0.0        |
| 20         | JONES      | 2975       | 0.5        |
| 20         | ADAMS      | 1100       | 0.75       |
| 20         | SMITH      | 800        | 1.0        |
| 30         | BLAKE      | 2850       | 0.0        |
| 30         | ALLEN      | 1600       | 0.2        |
| 30         | TURNER     | 1500       | 0.4        |
| 30         | MARTIN     | 1250       | 0.6        |
| 30         | WARD       | 1250       | 0.6        |
| 30         | JAMES      | 950        | 1.0        |
+------------+------------+------------+------------+
```

CUME_DIST

- 命令格式

```
double cume_dist() over([partition_clause] [orderby_clause])
```

- 命令说明

  求累计分布，相当于求分区中大于等于当前行的数据在分区中的占比。大小关系由orderby_clause判定。

- 参数说明

  partition_clause及orderby_clause：详情请参见windowing_definition。

- 返回值说明

  返回DOUBLE类型。具体的返回值等于 `row_number_of_last_peer / partition_row_count` ，其中： `row_number_of_last_peer` 指当前行所属GROUP的最后一行数据的ROW_NUMBER窗口函数返回值， `partition_row_count` 为该行数据所属分区的数据行数。

- 示例

  将所有职工根据部门（deptno）分组（作为开窗列），计算薪水（sal）在同一组内的前百分之几。命令示例如下：

```
select deptno, ename, sal, concat(round(cume_dist() over (partition by deptno order by sal desc)*100,2),'%') as cume_dist from
emp;
```

  返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | cume_dist  |
+------------+------------+------------+------------+
| 10         | JACCKA     | 5000       | 33.33%     |
| 10         | KING       | 5000       | 33.33%     |
| 10         | CLARK      | 2450       | 66.67%     |
| 10         | WELAN      | 2450       | 66.67%     |
| 10         | TEBAGE     | 1300       | 100.0%     |
| 10         | MILLER     | 1300       | 100.0%     |
| 20         | SCOTT      | 3000       | 40.0%      |
| 20         | FORD       | 3000       | 40.0%      |
| 20         | JONES      | 2975       | 60.0%      |
| 20         | ADAMS      | 1100       | 80.0%      |
| 20         | SMITH      | 800        | 100.0%     |
| 30         | BLAKE      | 2850       | 16.67%     |
| 30         | ALLEN      | 1600       | 33.33%     |
| 30         | TURNER     | 1500       | 50.0%      |
| 30         | MARTIN     | 1250       | 83.33%     |
| 30         | WARD       | 1250       | 83.33%     |
| 30         | JAMES      | 950        | 100.0%     |
+------------+------------+------------+------------+
```

NTILE

- 命令格式

```
bigint ntile(bigint <N>) over ([partition_clause] [orderby_clause])
```

- 命令说明

用于将分区中的数据按照顺序切分成N等份，并返回数据所在等份的编号。如果分区中的数据不能被均匀地切分成N等份时，最前面的等份（编号较小的）会优先多分配1条数据。

- 参数说明
  - N：必填。切片数量。BIGINT类型。
  - partition_clause及orderby_clause：详情请参见windowing_definition。
- 返回值说明

  返回BIGINT类型。
- 示例

  将所有职工根据部门按薪水（sal）从高到低切分为3组，并获得职工自己所在组的序号。命令示例如下：

```
select deptno, ename, sal, ntile(3) over (partition by deptno order by sal desc) as nt3 from emp;
```

返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | nt3        |
+------------+------------+------------+------------+
| 10         | JACCKA     | 5000       | 1          |
| 10         | KING       | 5000       | 1          |
| 10         | CLARK      | 2450       | 2          |
| 10         | WELAN      | 2450       | 2          |
| 10         | TEBAGE     | 1300       | 3          |
| 10         | MILLER     | 1300       | 3          |
| 20         | SCOTT      | 3000       | 1          |
| 20         | FORD       | 3000       | 1          |
| 20         | JONES      | 2975       | 2          |
| 20         | ADAMS      | 1100       | 2          |
| 20         | SMITH      | 800        | 3          |
| 30         | BLAKE      | 2850       | 1          |
| 30         | ALLEN      | 1600       | 1          |
| 30         | TURNER     | 1500       | 2          |
| 30         | MARTIN     | 1250       | 2          |
| 30         | WARD       | 1250       | 3          |
| 30         | JAMES      | 950        | 3          |
+------------+------------+------------+------------+
```

LAG

- 命令格式

```
lag(<expr>[, bigint <offset>[, <default>]]) over([partition_clause] orderby_clause)
```

- 命令说明

  返回当前行往前（朝分区头部方向）第offset行数据对应的表达式expr的值。表达式expr可以是列、列运算或者函数运算等。
- 参数说明
  - expr：必填。待计算返回结果的表达式。
  - offset：可选。偏移量，BIGINT类型常量，取值大于等于0。值为0时表示当前行，为1时表示前一行，以此类推。默认值为1。输入值为STRING类型、DOUBLE类型则隐式转换为BIGINT类型后进行运算。
  - default：可选。当offset指定的范围越界时的缺省值，常量，默认值为NULL。需要与expr对应的数据类型相同。如果expr非常量，则基于当前行进行求值。
  - partition_clause及orderby_clause：详情请参见windowing_definition。
- 返回值说明

  返回值类型同expr类型。
- 示例

  将所有职工根据部门（deptno）分组（作为开窗列），每位员工的薪水（sal）做偏移。命令示例如下：

```
select deptno, ename, sal, lag(sal, 1) over (partition by deptno order by sal) as sal_new from emp;
```

返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | sal_new    |
+------------+------------+------------+------------+
| 10         | TEBAGE     | 1300       | NULL       |
| 10         | MILLER     | 1300       | 1300       |
| 10         | CLARK      | 2450       | 1300       |
| 10         | WELAN      | 2450       | 2450       |
| 10         | KING       | 5000       | 2450       |
| 10         | JACCKA     | 5000       | 5000       |
| 20         | SMITH      | 800        | NULL       |
| 20         | ADAMS      | 1100       | 800        |
| 20         | JONES      | 2975       | 1100       |
| 20         | SCOTT      | 3000       | 2975       |
| 20         | FORD       | 3000       | 3000       |
| 30         | JAMES      | 950        | NULL       |
| 30         | MARTIN     | 1250       | 950        |
| 30         | WARD       | 1250       | 1250       |
| 30         | TURNER     | 1500       | 1250       |
| 30         | ALLEN      | 1600       | 1500       |
| 30         | BLAKE      | 2850       | 1600       |
+------------+------------+------------+------------+
```

LEAD

- 命令格式

```
lead(<expr>[, bigint <offset>[, <default>]]) over([partition_clause] orderby_clause)
```

- 命令说明

  返回当前行往后（朝分区尾部方向）第offset行数据对应的表达式expr的值。表达式expr可以是列、列运算或者函数运算等。

- 参数说明
  - expr：必填。待计算返回结果的表达式。
  - offset：可选。偏移量，BIGINT类型常量，取值大于等于0。值为0时表示当前行，为1时表示后一行，以此类推。默认值为1。输入值为STRING类型、DOUBLE类型则隐式转换为BIGINT类型后进行运算。
  - default：可选。当offset指定的范围越界时的缺省值，常量，默认值为NULL。需要与expr对应的数据类型相同。如果expr非常量，则基于当前行进行求值。
  - partition_clause及orderby_clause：详情请参见windowing_definition。

- 返回值说明

  返回值类型同expr类型。

- 示例

  将所有职工根据部门（deptno）分组（作为开窗列），每位员工的薪水（sal）做偏移。命令示例如下：

```
select deptno, ename, sal, lead(sal, 1) over (partition by deptno order by sal) as sal_new from emp;
```

  返回结果如下：

```
+------------+------------+------------+------------+
| deptno     | ename      | sal        | sal_new    |
+------------+------------+------------+------------+
| 10         | TEBAGE     | 1300       | 1300       |
| 10         | MILLER     | 1300       | 2450       |
| 10         | CLARK      | 2450       | 2450       |
| 10         | WELAN      | 2450       | 5000       |
| 10         | KING       | 5000       | 5000       |
| 10         | JACCKA     | 5000       | NULL       |
| 20         | SMITH      | 800        | 1100       |
| 20         | ADAMS      | 1100       | 2975       |
| 20         | JONES      | 2975       | 3000       |
| 20         | SCOTT      | 3000       | 3000       |
| 20         | FORD       | 3000       | NULL       |
| 30         | JAMES      | 950        | 1250       |
| 30         | MARTIN     | 1250       | 1250       |
| 30         | WARD       | 1250       | 1500       |
| 30         | TURNER     | 1500       | 1600       |
| 30         | ALLEN      | 1600       | 2850       |
| 30         | BLAKE      | 2850       | NULL       |
+------------+------------+------------+------------+
```

FIRST_VALUE

- 命令格式

```
first_value(<expr>[, <ignore_nulls>]) over ([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  返回表达式expr在窗口的第一条数据上进行运算的结果。

- 参数说明
  - expr：必填。待计算返回结果的表达式。
  - ignore_nulls：可选。BOOLEAN类型。表示是否忽略NULL值。默认值为False。当参数的值为True时，返回窗口中第一条非NULL的expr值。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。

- 返回值说明

  返回值类型同expr类型。

- 示例

  将所有职工根据部门分组，返回每组中的第一行数据。命令示例如下：
  - 不指定order by：

```
select deptno, ename, sal, first_value(sal) over (partition by deptno) as first_value from emp;
```

    返回结果如下：

```
+------------+------------+------------+-------------+
| deptno     | ename      | sal        | first_value |
+------------+------------+------------+-------------+
| 10         | TEBAGE     | 1300       | 1300        |   --当前窗口的开始行。
| 10         | CLARK      | 2450       | 1300        |
| 10         | KING       | 5000       | 1300        |
| 10         | MILLER     | 1300       | 1300        |
| 10         | JACCKA     | 5000       | 1300        |
| 10         | WELAN      | 2450       | 1300        |
| 20         | FORD       | 3000       | 3000        |   --当前窗口的开始行。
| 20         | SCOTT      | 3000       | 3000        |
| 20         | SMITH      | 800        | 3000        |
| 20         | ADAMS      | 1100       | 3000        |
| 20         | JONES      | 2975       | 3000        |
| 30         | TURNER     | 1500       | 1500        |   --当前窗口的开始行。
| 30         | JAMES      | 950        | 1500        |
| 30         | ALLEN      | 1600       | 1500        |
| 30         | WARD       | 1250       | 1500        |
| 30         | MARTIN     | 1250       | 1500        |
| 30         | BLAKE      | 2850       | 1500        |
+------------+------------+------------+-------------+
```

  - 指定order by：

```
select deptno, ename, sal, first_value(sal) over (partition by deptno order by sal desc) as first_value from emp;
```

    返回结果如下：

```
+------------+------------+------------+-------------+
| deptno     | ename      | sal        | first_value |
+------------+------------+------------+-------------+
| 10         | JACCKA     | 5000       | 5000        |   --当前窗口的开始行。
| 10         | KING       | 5000       | 5000        |
| 10         | CLARK      | 2450       | 5000        |
| 10         | WELAN      | 2450       | 5000        |
| 10         | TEBAGE     | 1300       | 5000        |
| 10         | MILLER     | 1300       | 5000        |
| 20         | SCOTT      | 3000       | 3000        |   --当前窗口的开始行。
| 20         | FORD       | 3000       | 3000        |
| 20         | JONES      | 2975       | 3000        |
| 20         | ADAMS      | 1100       | 3000        |
| 20         | SMITH      | 800        | 3000        |
| 30         | BLAKE      | 2850       | 2850        |   --当前窗口的开始行。
| 30         | ALLEN      | 1600       | 2850        |
| 30         | TURNER     | 1500       | 2850        |
| 30         | MARTIN     | 1250       | 2850        |
| 30         | WARD       | 1250       | 2850        |
| 30         | JAMES      | 950        | 2850        |
+------------+------------+------------+-------------+
```

LAST_VALUE

- 命令格式

```
last_value(<expr>[, <ignore_nulls>]) over([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  返回表达式expr在窗口的最后一条数据上进行运算的结果。

- 参数说明
  - expr：必填。待计算返回结果的表达式。
  - ignore_nulls：可选。BOOLEAN类型。表示是否忽略NULL值。默认值为False。当参数的值为True时，返回窗口中最后一条非NULL的expr值。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。

- 返回值说明

  返回值类型同expr类型。

- 示例

  将所有职工根据部门分组，返回每组中的最后一行数据。命令示例如下：
  - 不指定order by，当前窗口为第一行到最后一行的范围，返回当前窗口的最后一行的值。

```
select deptno, ename, sal, last_value(sal) over (partition by deptno) as last_value from emp;
```

    返回结果如下：

```
+------------+------------+------------+-------------+
| deptno     | ename      | sal        | last_value |
+------------+------------+------------+-------------+
| 10         | TEBAGE     | 1300       | 2450        |
| 10         | CLARK      | 2450       | 2450        |
| 10         | KING       | 5000       | 2450        |
| 10         | MILLER     | 1300       | 2450        |
| 10         | JACCKA     | 5000       | 2450        |
| 10         | WELAN      | 2450       | 2450        |   --当前窗口的最后一行。
| 20         | FORD       | 3000       | 2975        |
| 20         | SCOTT      | 3000       | 2975        |
| 20         | SMITH      | 800        | 2975        |
| 20         | ADAMS      | 1100       | 2975        |
| 20         | JONES      | 2975       | 2975        |   --当前窗口的最后一行。
| 30         | TURNER     | 1500       | 2850        |
| 30         | JAMES      | 950        | 2850        |
| 30         | ALLEN      | 1600       | 2850        |
| 30         | WARD       | 1250       | 2850        |
| 30         | MARTIN     | 1250       | 2850        |
| 30         | BLAKE      | 2850       | 2850        |   --当前窗口的最后一行。
+------------+------------+------------+-------------+
```

  - 指定order by，当前窗口为第一行到当前行的范围。返回当前窗口的当前行的值。

```
select deptno, ename, sal, last_value(sal) over (partition by deptno order by sal desc) as last_value from emp;
```

    返回结果如下：

```
+------------+------------+------------+-------------+
| deptno     | ename      | sal        | last_value |
+------------+------------+------------+-------------+
| 10         | JACCKA     | 5000       | 5000        |   --当前窗口的当前行。
| 10         | KING       | 5000       | 5000        |   --当前窗口的当前行。
| 10         | CLARK      | 2450       | 2450        |   --当前窗口的当前行。
| 10         | WELAN      | 2450       | 2450        |   --当前窗口的当前行。
| 10         | TEBAGE     | 1300       | 1300        |   --当前窗口的当前行。
| 10         | MILLER     | 1300       | 1300        |   --当前窗口的当前行。
| 20         | SCOTT      | 3000       | 3000        |   --当前窗口的当前行。
| 20         | FORD       | 3000       | 3000        |   --当前窗口的当前行。
| 20         | JONES      | 2975       | 2975        |   --当前窗口的当前行。
| 20         | ADAMS      | 1100       | 1100        |   --当前窗口的当前行。
| 20         | SMITH      | 800        | 800         |   --当前窗口的当前行。
| 30         | BLAKE      | 2850       | 2850        |   --当前窗口的当前行。
| 30         | ALLEN      | 1600       | 1600        |   --当前窗口的当前行。
| 30         | TURNER     | 1500       | 1500        |   --当前窗口的当前行。
| 30         | MARTIN     | 1250       | 1250        |   --当前窗口的当前行。
| 30         | WARD       | 1250       | 1250        |   --当前窗口的当前行。
| 30         | JAMES      | 950        | 950         |   --当前窗口的当前行。
+------------+------------+------------+-------------+
```

NTH_VALUE

- 命令格式

```
nth_value(<expr>, <number> [, <ignore_nulls>]) over ([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  返回表达式expr在窗口的第N条数据进行运算的结果。

- 参数说明
  - expr：必填。待计算返回结果的表达式。
  - number：必填。BIGINT类型。大于等于1的整数。值为1时与FIRST_VALUE等价。
  - ignore_nulls：可选。BOOLEAN类型。表示是否忽略NULL值。默认值为False。当参数的值为True时，返回窗口中第N条非NULL的expr值。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。

- 返回值说明

  返回值类型同expr类型。

- 示例

  将所有职工根据部门分组，返回每组中的第6行数据。命令示例如下：
  - 不指定order by，当前窗口为第一行到最后一行的范围，返回当前窗口第6行的值。

    ```
    select deptno, ename, sal, nth_value(sal,6) over (partition by deptno) as nth_value from emp;
    ```

    返回结果如下：

    ```
    +------------+------------+------------+------------+
    | deptno     | ename      | sal        | nth_value  |
    +------------+------------+------------+------------+
    | 10         | TEBAGE     | 1300       | 2450       |
    | 10         | CLARK      | 2450       | 2450       |
    | 10         | KING       | 5000       | 2450       |
    | 10         | MILLER     | 1300       | 2450       |
    | 10         | JACCKA     | 5000       | 2450       |
    | 10         | WELAN      | 2450       | 2450       |   --当前窗口的第6行。
    | 20         | FORD       | 3000       | NULL       |
    | 20         | SCOTT      | 3000       | NULL       |
    | 20         | SMITH      | 800        | NULL       |
    | 20         | ADAMS      | 1100       | NULL       |
    | 20         | JONES      | 2975       | NULL       |   --当前窗口的没有第6行，返回NULL。
    | 30         | TURNER     | 1500       | 2850       |
    | 30         | JAMES      | 950        | 2850       |
    | 30         | ALLEN      | 1600       | 2850       |
    | 30         | WARD       | 1250       | 2850       |
    | 30         | MARTIN     | 1250       | 2850       |
    | 30         | BLAKE      | 2850       | 2850       |   --当前窗口的第6行。
    +------------+------------+------------+------------+
    ```

  - 指定order by，当前窗口为第一行到当前行的范围，返回当前窗口第6行的值。

    ```
    select deptno, ename, sal, nth_value(sal,6) over (partition by deptno order by sal) as nth_value from emp;
    ```

    返回结果如下：

    ```
    +------------+------------+------------+------------+
    | deptno     | ename      | sal        | nth_value  |
    +------------+------------+------------+------------+
    | 10         | TEBAGE     | 1300       | NULL       |
    | 10         | MILLER     | 1300       | NULL       |   --当前窗口只有2行，第6行超过了窗口长度。
    | 10         | CLARK      | 2450       | NULL       |
    | 10         | WELAN      | 2450       | NULL       |
    | 10         | KING       | 5000       | 5000       |
    | 10         | JACCKA     | 5000       | 5000       |
    | 20         | SMITH      | 800        | NULL       |
    | 20         | ADAMS      | 1100       | NULL       |
    | 20         | JONES      | 2975       | NULL       |
    | 20         | SCOTT      | 3000       | NULL       |
    | 20         | FORD       | 3000       | NULL       |
    | 30         | JAMES      | 950        | NULL       |
    | 30         | MARTIN     | 1250       | NULL       |
    | 30         | WARD       | 1250       | NULL       |
    | 30         | TURNER     | 1500       | NULL       |
    | 30         | ALLEN      | 1600       | NULL       |
    | 30         | BLAKE      | 2850       | 2850       |
    +------------+------------+------------+------------+
    ```

CLUSTER_SAMPLE

- 命令格式

```
boolean cluster_sample(bigint <N>) OVER ([partition_clause])
boolean cluster_sample(bigint <N>, bigint <M>) OVER ([partition_clause])
```

- 命令说明
  - `cluster_sample(bigint <N>)`：表示随机抽取N条数据。
  - `cluster_sample(bigint <N>, bigint <M>)`：表示按比例（M/N）随机抽取。即抽取`partition_row_count×M / N`条数据。`partition_row_count`指分区中的数据行数。
- 参数说明
  - N：必填。BIGINT类型常量。N为NULL时，返回值为NULL。
  - M：必填。BIGINT类型常量。M为NULL时，返回值为NULL。
  - partition_clause：可选。详情请参见windowing_definition。
- 返回值说明

  返回BOOLEAN类型。
- 示例

  如果您需要从每组中抽取约20%的值，命令示例如下：

```
select deptno, sal
    from (
        select deptno, sal, cluster_sample(5, 1) over (partition by deptno) as flag
        from emp
        ) sub
    where flag = true;
```

  返回结果如下：

```
+------------+------------+
| deptno     | sal        |
+------------+------------+
| 10         | 1300       |
| 20         | 3000       |
| 30         | 950        |
+------------+------------+
```

COUNT

- 命令格式

```
bigint count(*) over ([partition_clause] [orderby_clause] [frame_clause])
bigint count([distinct] <expr>[,...]) over ([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明
  - `count(*)`：返回总行数。
  - `count([distinct] <expr>[,...])`：计算行数时会忽略expr值为NULL的行，如果有多个expr，则任意expr值为NULL都被忽略。此外如果指定了distinct关键字，则计算去重之后的数据行数，任意expr值为NULL的行同样会被忽略。
- 参数说明
  - expr：必填。待计算计数值的列。可以为任意类型。当值为NULL时，该行不参与计算。当指定DISTINCT关键字时，表示取唯一值的计数值。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。
- 返回值说明

  返回BIGINT类型。
- 示例

- 示例1：指定薪水（sal）为开窗列，不排序，返回当前窗口（相同sal）的从开始行到最后一行的累计计数值。命令示例如下：

```
select sal, count(sal) over (partition by sal) as count from emp;
```

返回结果如下：

```
+------------+------------+
| sal        | count      |
+------------+------------+
| 800        | 1          |
| 950        | 1          |
| 1100       | 1          |
| 1250       | 2          |   --窗口开始行。第1行和第2行的sal一致，则第1行的count为第2行的累计计数值。
| 1250       | 2          |   --当前窗口从第1行到第2行的累计计数值。
| 1300       | 2          |
| 1300       | 2          |
| 1500       | 1          |
| 1600       | 1          |
| 2450       | 2          |
| 2450       | 2          |
| 2850       | 1          |
| 2975       | 1          |
| 3000       | 2          |
| 3000       | 2          |
| 5000       | 2          |
| 5000       | 2          |
+------------+------------+
```

- 示例2：非Hive兼容模式下，指定薪水（sal）为开窗列，并排序，返回当前窗口（相同sal）从开始行到当前行的累计计数值。命令示例如下：

```
--关闭Hive兼容模式。
set odps.sql.hive.compatible=false;
--执行如下SQL命令。
select sal, count(sal) over (partition by sal order by sal) as count from emp;
```

返回结果如下：

```
+------------+------------+
| sal        | count      |
+------------+------------+
| 800        | 1          |
| 950        | 1          |
| 1100       | 1          |
| 1250       | 1          |   --窗口开始行。第1行的累计计数值是1。
| 1250       | 2          |   --第2行的累计计数值是2。
| 1300       | 1          |
| 1300       | 2          |
| 1500       | 1          |
| 1600       | 1          |
| 2450       | 1          |
| 2450       | 2          |
| 2850       | 1          |
| 2975       | 1          |
| 3000       | 1          |
| 3000       | 2          |
| 5000       | 1          |
| 5000       | 2          |
+------------+------------+
```

- 示例3：Hive兼容模式下，指定薪水（sal）为开窗列，并排序，返回当前窗口（相同sal）从开始行至最后一行的累计计数值。命令示例如下：

```
--开启Hive兼容模式。
set odps.sql.hive.compatible=true;
--执行如下SQL命令。
select sal, count(sal) over (partition by sal order by sal) as count from emp;
```

返回结果如下：

```
+------------+------------+
| sal        | count      |
+------------+------------+
| 800        | 1          |
| 950        | 1          |
| 1100       | 1          |
| 1250       | 2          |   --窗口开始行。第1行和第2行的sal一致，则第1行的count为第2行的累计计数值。
| 1250       | 2          |   --当前窗口从第1行到第2行的累计计数值。
| 1300       | 2          |
| 1300       | 2          |
| 1500       | 1          |
| 1600       | 1          |
| 2450       | 2          |
| 2450       | 2          |
| 2850       | 1          |
| 2975       | 1          |
| 3000       | 2          |
| 3000       | 2          |
| 5000       | 2          |
| 5000       | 2          |
+------------+------------+
```

MIN

- 命令格式

```
min(<expr>) over([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  返回窗口中expr的最小值。

- 参数说明
  - expr：必填。用于计算最小值的表达式。除BOOLEAN外的任意类型，当值为NULL时，该行不参与计算。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。

- 返回值说明

  返回值类型同expr类型。

- 示例
  - 示例1：指定部门（deptno）为开窗列，计算薪水（sal）最小值，不排序，返回当前窗口（相同deptno）的最小值。执行命令如下：

```
select deptno, sal, min(sal) over (partition by deptno) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 1300       |   --窗口开始行，从第1行到第6行的最小值。
| 10         | 2450       | 1300       |   --从第1行到第6行的最小值。
| 10         | 5000       | 1300       |   --从第1行到第6行的最小值。
| 10         | 1300       | 1300       |
| 10         | 5000       | 1300       |
| 10         | 2450       | 1300       |
| 20         | 3000       | 800        |
| 20         | 3000       | 800        |
| 20         | 800        | 800        |
| 20         | 1100       | 800        |
| 20         | 2975       | 800        |
| 30         | 1500       | 950        |
| 30         | 950        | 950        |
| 30         | 1600       | 950        |
| 30         | 1250       | 950        |
| 30         | 1250       | 950        |
| 30         | 2850       | 950        |
+------------+------------+------------+
```

- 示例2：指定部门（deptno）为开窗列，计算薪水（sal）最小值，并排序，返回当前窗口（相同deptno）从开始行到当前行的最小值。执行命令如下：

```
select deptno, sal, min(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 1300       |   --窗口开始行。
| 10         | 1300       | 1300       |   --从第1行到第2行的最小值。
| 10         | 2450       | 1300       |   --从第1行到第3行的最小值。
| 10         | 2450       | 1300       |
| 10         | 5000       | 1300       |
| 10         | 5000       | 1300       |
| 20         | 800        | 800        |
| 20         | 1100       | 800        |
| 20         | 2975       | 800        |
| 20         | 3000       | 800        |
| 20         | 3000       | 800        |
| 30         | 950        | 950        |
| 30         | 1250       | 950        |
| 30         | 1250       | 950        |
| 30         | 1500       | 950        |
| 30         | 1600       | 950        |
| 30         | 2850       | 950        |
+------------+------------+------------+
```

MAX

- 命令格式

```
max(<expr>) over([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  返回窗口中expr的最大值。

- 参数说明
  - expr：必填。用于计算最大值的表达式。除BOOLEAN外的任意类型，当值为NULL时，该行不参与计算。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。

- 返回值说明

  返回值的类型同expr类型。

- 示例
  - 示例1：指定部门（deptno）为开窗列，计算薪水（sal）最大值，不排序，返回当前窗口（相同deptno）的最大值。执行命令如下：

```
select deptno, sal, max(sal) over (partition by deptno) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 5000       |   --窗口开始行，从第1行到第6行的最大值。
| 10         | 2450       | 5000       |   --从第1行到第6行的最大值。
| 10         | 5000       | 5000       |   --从第1行到第6行的最大值。
| 10         | 1300       | 5000       |
| 10         | 5000       | 5000       |
| 10         | 2450       | 5000       |
| 20         | 3000       | 3000       |
| 20         | 3000       | 3000       |
| 20         | 800        | 3000       |
| 20         | 1100       | 3000       |
| 20         | 2975       | 3000       |
| 30         | 1500       | 2850       |
| 30         | 950        | 2850       |
| 30         | 1600       | 2850       |
| 30         | 1250       | 2850       |
| 30         | 1250       | 2850       |
| 30         | 2850       | 2850       |
+------------+------------+------------+
```

- 示例2：指定部门（deptno）为开窗列，计算薪水（sal）最大值，并排序，返回当前窗口（相同deptno）从开始行到当前行的最大值。执行命令如下：

```
select deptno, sal, max(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 1300       |   --窗口开始行。
| 10         | 1300       | 1300       |   --从第1行到第2行的最大值。
| 10         | 2450       | 2450       |   --从第1行到第3行的最大值。
| 10         | 2450       | 2450       |   --从第1行到第4行的最大值。
| 10         | 5000       | 5000       |
| 10         | 5000       | 5000       |
| 20         | 800        | 800        |
| 20         | 1100       | 1100       |
| 20         | 2975       | 2975       |
| 20         | 3000       | 3000       |
| 20         | 3000       | 3000       |
| 30         | 950        | 950        |
| 30         | 1250       | 1250       |
| 30         | 1250       | 1250       |
| 30         | 1500       | 1500       |
| 30         | 1600       | 1600       |
| 30         | 2850       | 2850       |
+------------+------------+------------+
```

SUM

- 命令格式

```
sum([distinct] <expr>) over ([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  返回窗口中expr之和。

- 参数说明
  - expr：必填。待计算汇总值的列。DOUBLE类型、DECIMAL类型或BIGINT类型。
    - 当输入值为STRING类型时，会隐式转换为DOUBLE类型后参与运算，其他类型返回报错。
    - 当输入值为NULL时，该行不参与计算。
    - 当指定distinct关键字时，表示计算唯一值的汇总值。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。
- 返回值说明
  - 输入值为BIGINT类型时，返回BIGINT类型。
  - 输入值为DECIMAL类型时，返回DECIMAL类型。
  - 输入值为DOUBLE类型或STRING类型时，返回DOUBLE类型。
  - 输入值都为NULL时，返回NULL。
- 示例

- 示例1：指定部门（deptno）为开窗列，计算薪水（sal）汇总值，不排序，返回当前窗口（相同deptno）的累计汇总值。命令示例如下：

```
select deptno, sal, sum(sal) over (partition by deptno) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 17500      |   --窗口开始行。第1行到第6行的累计汇总值。
| 10         | 2450       | 17500      |   --从第1行到第6行的累计汇总值。
| 10         | 5000       | 17500      |   --从第1行到第6行的累计汇总值。
| 10         | 1300       | 17500      |
| 10         | 5000       | 17500      |
| 10         | 2450       | 17500      |
| 20         | 3000       | 10875      |
| 20         | 3000       | 10875      |
| 20         | 800        | 10875      |
| 20         | 1100       | 10875      |
| 20         | 2975       | 10875      |
| 30         | 1500       | 9400       |
| 30         | 950        | 9400       |
| 30         | 1600       | 9400       |
| 30         | 1250       | 9400       |
| 30         | 1250       | 9400       |
| 30         | 2850       | 9400       |
+------------+------------+------------+
```

- 示例2：非Hive兼容模式下，指定部门（deptno）为开窗列，计算薪水（sal）汇总值，并排序，返回当前窗口（相同deptno）从开始行到当前行的累计汇总值。命令示例如下：

```
--关闭Hive兼容模式。
set odps.sql.hive.compatible=false;
--执行如下SQL命令。
select deptno, sal, sum(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 1300       |   --窗口开始行。
| 10         | 1300       | 2600       |   --从第1行到第2行的累计汇总值。
| 10         | 2450       | 5050       |   --从第1行到第3行的累计汇总值。
| 10         | 2450       | 7500       |
| 10         | 5000       | 12500      |
| 10         | 5000       | 17500      |
| 20         | 800        | 800        |
| 20         | 1100       | 1900       |
| 20         | 2975       | 4875       |
| 20         | 3000       | 7875       |
| 20         | 3000       | 10875      |
| 30         | 950        | 950        |
| 30         | 1250       | 2200       |
| 30         | 1250       | 3450       |
| 30         | 1500       | 4950       |
| 30         | 1600       | 6550       |
| 30         | 2850       | 9400       |
+------------+------------+------------+
```

- 示例3：Hive兼容模式下，指定部门（deptno）为开窗列，计算薪水（sal）汇总值，并排序，返回当前窗口（相同deptno）从开始行到与当前行相同值所在行（相同sal值的汇总值相同）的累计汇总值。命令示例如下：

```
--开启Hive兼容模式。
set odps.sql.hive.compatible=true;
--执行如下SQL命令。
select deptno, sal, sum(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 2600       |   --窗口开始行。第1行和第2行的sal一致，则第1行的汇总值为第1行到第2行的累计汇总值。
| 10         | 1300       | 2600       |   --从第1行到第2行的累计汇总值。
| 10         | 2450       | 7500       |   --第3行和第4行的sal一致，则第3行的汇总值为从第1行到第4行的累计汇总值。
| 10         | 2450       | 7500       |   --从第1行到第4行的累计汇总值。
| 10         | 5000       | 17500      |
| 10         | 5000       | 17500      |
| 20         | 800        | 800        |
| 20         | 1100       | 1900       |
| 20         | 2975       | 4875       |
| 20         | 3000       | 10875      |
| 20         | 3000       | 10875      |
| 30         | 950        | 950        |
| 30         | 1250       | 3450       |
| 30         | 1250       | 3450       |
| 30         | 1500       | 4950       |
| 30         | 1600       | 6550       |
| 30         | 2850       | 9400       |
+------------+------------+------------+
```

AVG

- 命令格式

```
double avg([distinct] double <expr>) over ([partition_clause] [orderby_clause] [frame_clause])
decimal avg([distinct] decimal <expr>) over ([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  返回窗口中expr的平均值。

- 参数说明
  - expr：必填。计算返回结果的表达式。DOUBLE类型或DECIMAL类型。
    - 当输入值为STRING、BIGINT类型时，会隐式转换为DOUBLE类型后参与运算，输入其他数据类型返回报错。
    - 当输入值为NULL时，该行不参与计算。
    - 当指定distinct关键字时，表示取唯一值的平均值。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。
- 返回值说明

  expr为DECIMAL类型时，返回DECIMAL类型。其他情况下返回DOUBLE类型。expr的值都为NULL时，返回结果为NULL。

- 示例

- 示例1：指定部门（deptno）为开窗列，计算薪水（sal）平均值，不排序，返回当前窗口（相同deptno）从开始行到最后一行的累计平均值。命令示例如下：

```
select deptno, sal, avg(sal) over (partition by deptno) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 2916.6666666666665 |   --窗口开始行，从第1行到第6行的累计平均值。
| 10         | 2450       | 2916.6666666666665 |   --从第1行到第6行的累计平均值。
| 10         | 5000       | 2916.6666666666665 |   --从第1行到第6行的累计平均值。
| 10         | 1300       | 2916.6666666666665 |
| 10         | 5000       | 2916.6666666666665 |
| 10         | 2450       | 2916.6666666666665 |
| 20         | 3000       | 2175.0     |
| 20         | 3000       | 2175.0     |
| 20         | 800        | 2175.0     |
| 20         | 1100       | 2175.0     |
| 20         | 2975       | 2175.0     |
| 30         | 1500       | 1566.6666666666667 |
| 30         | 950        | 1566.6666666666667 |
| 30         | 1600       | 1566.6666666666667 |
| 30         | 1250       | 1566.6666666666667 |
| 30         | 1250       | 1566.6666666666667 |
| 30         | 2850       | 1566.6666666666667 |
+------------+------------+------------+
```

- 示例2：非Hive兼容模式下，指定部门（deptno）为开窗列，计算薪水（sal）平均值，并排序，返回当前窗口（相同deptno）从开始行到当前行的累计平均值。命令示例如下：

```
--关闭Hive兼容模式。
set odps.sql.hive.compatible=false;
--执行如下SQL命令。
select deptno, sal, avg(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 1300.0     |           --窗口开始行。
| 10         | 1300       | 1300.0     |           --从第1行到第2行的累计平均值。
| 10         | 2450       | 1683.3333333333333 |   --从第1行到第3行的累计平均值。
| 10         | 2450       | 1875.0     |           --从第1行到第4行的累计平均值。
| 10         | 5000       | 2500.0     |           --从第1行到第5行的累计平均值。
| 10         | 5000       | 2916.6666666666665 |   --从第1行到第6行的累计平均值。
| 20         | 800        | 800.0      |
| 20         | 1100       | 950.0      |
| 20         | 2975       | 1625.0     |
| 20         | 3000       | 1968.75    |
| 20         | 3000       | 2175.0     |
| 30         | 950        | 950.0      |
| 30         | 1250       | 1100.0     |
| 30         | 1250       | 1150.0     |
| 30         | 1500       | 1237.5     |
| 30         | 1600       | 1310.0     |
| 30         | 2850       | 1566.6666666666667 |
+------------+------------+------------+
```

- 示例3：Hive兼容模式下，指定部门（deptno）为开窗列，计算薪水（sal）平均值，并排序，返回当前窗口（相同deptno）从开始行到与当前行相同值所在行（相同sal值的平均值相同）的累计平均值。命令示例如下：

```
--开启Hive兼容模式。
set odps.sql.hive.compatible=true;
--执行如下SQL命令。
select deptno, sal, avg(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 1300.0     |          --窗口开始行。第1行和第2行的sal一致，则第1行的平均值为第1行到第2行的累计平均值。
| 10         | 1300       | 1300.0     |          --从第1行到第2行的累计平均值。
| 10         | 2450       | 1875.0     |          --第3行和第4行的sal一致，则第3行的平均值为从第1行到第4行的累计平均值。
| 10         | 2450       | 1875.0     |          --从第1行到第4行的累计平均值。
| 10         | 5000       | 2916.6666666666665 |
| 10         | 5000       | 2916.6666666666665 |
| 20         | 800        | 800.0      |
| 20         | 1100       | 950.0      |
| 20         | 2975       | 1625.0     |
| 20         | 3000       | 2175.0     |
| 20         | 3000       | 2175.0     |
| 30         | 950        | 950.0      |
| 30         | 1250       | 1150.0     |
| 30         | 1250       | 1150.0     |
| 30         | 1500       | 1237.5     |
| 30         | 1600       | 1310.0     |
| 30         | 2850       | 1566.6666666666667 |
+------------+------------+------------+
```

MEDIAN

- 命令格式

```
median(<expr>) over ([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  计算窗口中expr的中位数。

- 参数说明
  - expr：必填。待计算中位数的表达式。DOUBLE类型或DECIMAL类型。最多支持输入255个数字，至少要输入1个数字。
    - 当输入值为STRING类型或BIGINT类型，会隐式转换为DOUBLE类型后参与运算，输入为其他数据类型则返回报错。
    - 当输入值为NULL时，返回NULL。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。所有expr为NULL时，返回结果为NULL。

- 示例

  指定部门（deptno）为开窗列，计算薪水（sal）中位数，返回当前窗口（相同deptno）的中位数。执行命令如下：

```
select deptno, sal, median(sal) over (partition by deptno) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 2450.0     |   --窗口开始行，从第1行到第6行的中位数。
| 10         | 2450       | 2450.0     |
| 10         | 5000       | 2450.0     |
| 10         | 1300       | 2450.0     |
| 10         | 5000       | 2450.0     |
| 10         | 2450       | 2450.0     |
| 20         | 3000       | 2975.0     |
| 20         | 3000       | 2975.0     |
| 20         | 800        | 2975.0     |
| 20         | 1100       | 2975.0     |
| 20         | 2975       | 2975.0     |
| 30         | 1500       | 1375.0     |
| 30         | 950        | 1375.0     |
| 30         | 1600       | 1375.0     |
| 30         | 1250       | 1375.0     |
| 30         | 1250       | 1375.0     |
| 30         | 2850       | 1375.0     |
+------------+------------+------------+
```

STDDEV

- 命令格式

```
double stddev|stddev_pop([distinct] <expr>) over ([partition_clause] [orderby_clause] [frame_clause])
decimal stddev|stddev_pop([distinct] <expr>) over ([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  计算总体标准差，STDDEV_POP函数的别名。

- 参数说明
  - expr：必填。待计算总体标准差的表达式。DOUBLE类型或DECIMAL类型。
    - 当输入值为STRING类型或BIGINT类型时，会隐式转换为DOUBLE类型后参与运算，其他类型返回报错。
    - 当输入值为NULL时，该行不参与计算。
    - 当指定distinct关键字时，表示计算唯一值的总体标准差。
  - partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。
- 返回值说明

  返回值类型同expr类型。所有expr为NULL时，返回结果为NULL。

- 示例
  - 示例1：指定部门（deptno）为开窗列，计算薪水（sal）总体标准差，不排序，返回当前窗口（相同deptno）的累计总体标准差。命令示例如下：

    ```
    select deptno, sal, stddev(sal) over (partition by deptno) from emp;
    ```

    返回结果如下：

    ```
    +------------+------------+------------+
    | deptno     | sal        | _c2        |
    +------------+------------+------------+
    | 10         | 1300       | 1546.1421524412158 |   --窗口开始行，从第1行到第6行的累计总体标准差。
    | 10         | 2450       | 1546.1421524412158 |   --从第1行到第6行的累计总体标准差。
    | 10         | 5000       | 1546.1421524412158 |
    | 10         | 1300       | 1546.1421524412158 |
    | 10         | 5000       | 1546.1421524412158 |
    | 10         | 2450       | 1546.1421524412158 |
    | 20         | 3000       | 1004.7387720198718 |
    | 20         | 3000       | 1004.7387720198718 |
    | 20         | 800        | 1004.7387720198718 |
    | 20         | 1100       | 1004.7387720198718 |
    | 20         | 2975       | 1004.7387720198718 |
    | 30         | 1500       | 610.1001739241042 |
    | 30         | 950        | 610.1001739241042 |
    | 30         | 1600       | 610.1001739241042 |
    | 30         | 1250       | 610.1001739241042 |
    | 30         | 1250       | 610.1001739241042 |
    | 30         | 2850       | 610.1001739241042 |
    +------------+------------+------------+
    ```

- 示例2：非Hive兼容模式下，指定部门（deptno）为开窗列，计算薪水（sal）总体标准差，并排序，返回当前窗口（相同deptno）从开始行到当前行的累计总体标准差。命令示例如下：

```
--关闭Hive兼容模式。
set odps.sql.hive.compatible=false;
--执行如下SQL命令。
select deptno, sal, stddev(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 0.0        |           --窗口开始行。
| 10         | 1300       | 0.0        |           --从第1行到第2行的累计总体标准差。
| 10         | 2450       | 542.1151989096865 |    --从第1行到第3行的累计总体标准差。
| 10         | 2450       | 575.0      |           --从第1行到第4行的累计总体标准差。
| 10         | 5000       | 1351.6656391282572 |
| 10         | 5000       | 1546.1421524412158 |
| 20         | 800        | 0.0        |
| 20         | 1100       | 150.0      |
| 20         | 2975       | 962.4188277460079 |
| 20         | 3000       | 1024.2947268730811 |
| 20         | 3000       | 1004.7387720198718 |
| 30         | 950        | 0.0        |
| 30         | 1250       | 150.0      |
| 30         | 1250       | 141.4213562373095 |
| 30         | 1500       | 194.8557158514987 |
| 30         | 1600       | 226.71568097509268 |
| 30         | 2850       | 610.1001739241042 |
+------------+------------+------------+
```

- 示例3：Hive兼容模式下，指定部门（deptno）为开窗列，计算薪水（sal）总体标准差，并排序，返回当前窗口（相同deptno）从开始行到与当前行相同值所在行（相同sal值的总体标准差相同）的累计总体标准差。命令示例如下：

```
--开启Hive兼容模式。
set odps.sql.hive.compatible=true;
--执行如下SQL命令。
select deptno, sal, stddev(sal) over (partition by deptno order by sal) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| deptno     | sal        | _c2        |
+------------+------------+------------+
| 10         | 1300       | 0.0        |           --窗口开始行。第1行和第2行的sal一致，则第1行的总体标准差为第1行到第2行的累计总体标准差。
| 10         | 1300       | 0.0        |           --从第1行到第2行的累计总体标准差。
| 10         | 2450       | 575.0      |           --第3行和第4行的sal一致，则第3行的总体标准差为从第1行到第4行的累计总体标准差。
| 10         | 2450       | 575.0      |           --从第1行到第4行的累计总体标准差。
| 10         | 5000       | 1546.1421524412158 |
| 10         | 5000       | 1546.1421524412158 |
| 20         | 800        | 0.0        |
| 20         | 1100       | 150.0      |
| 20         | 2975       | 962.4188277460079 |
| 20         | 3000       | 1004.7387720198718 |
| 20         | 3000       | 1004.7387720198718 |
| 30         | 950        | 0.0        |
| 30         | 1250       | 141.4213562373095 |
| 30         | 1250       | 141.4213562373095 |
| 30         | 1500       | 194.8557158514987 |
| 30         | 1600       | 226.71568097509268 |
| 30         | 2850       | 610.1001739241042 |
+------------+------------+------------+
```

STDDEV_SAMP

- 命令格式

```
double stddev_samp([distinct] <expr>) over([partition_clause] [orderby_clause] [frame_clause])
decimal stddev_samp([distinct] <expr>) over([partition_clause] [orderby_clause] [frame_clause])
```

- 命令说明

  计算样本标准差。

- 参数说明

- expr：必填。待计算样本标准差的表达式。DOUBLE类型或DECIMAL类型。
  - 当输入值为STRING类型或BIGINT类型时，会隐式转换为DOUBLE类型后参与运算，其他类型返回报错。
  - 当输入值为NULL时，该行不参与计算。
  - 当指定distinct关键字时，表示计算唯一值的样本标准差。
- partition_clause、orderby_clause及frame_clause：详情请参见windowing_definition。

- 返回值说明

  返回值类型同expr类型。所有expr为NULL时，返回结果为NULL。窗口仅包含1条expr值非NULL的数据时，结果为0。

- 示例
  - 示例1：指定部门（deptno）为开窗列，计算薪水（sal）样本标准差，不排序，返回当前窗口（相同deptno）的累计样本标准差。命令示例如下：

    ```
    select deptno, sal, stddev_samp(sal) over (partition by deptno) from emp;
    ```

    返回结果如下：

    ```
    +------------+------------+------------+
    | deptno     | sal        | _c2        |
    +------------+------------+------------+
    | 10         | 1300       | 1693.7138680032904 |   --窗口开始行，第1行到第6行的累计样本标准差。
    | 10         | 2450       | 1693.7138680032904 |   --从第1行到第6行的累计样本标准差。
    | 10         | 5000       | 1693.7138680032904 |   --从第1行到第6行的累计样本标准差。
    | 10         | 1300       | 1693.7138680032904 |
    | 10         | 5000       | 1693.7138680032904 |
    | 10         | 2450       | 1693.7138680032904 |
    | 20         | 3000       | 1123.3320969330487 |
    | 20         | 3000       | 1123.3320969330487 |
    | 20         | 800        | 1123.3320969330487 |
    | 20         | 1100       | 1123.3320969330487 |
    | 20         | 2975       | 1123.3320969330487 |
    | 30         | 1500       | 668.331255192114 |
    | 30         | 950        | 668.331255192114 |
    | 30         | 1600       | 668.331255192114 |
    | 30         | 1250       | 668.331255192114 |
    | 30         | 1250       | 668.331255192114 |
    | 30         | 2850       | 668.331255192114 |
    +------------+------------+------------+
    ```

  - 示例2：指定部门（deptno）为开窗列，计算薪水（sal）样本标准差，并排序，返回当前窗口（相同deptno）从开始行到当前行的累计样本标准差。命令示例如下：

    ```
    select deptno, sal, stddev_samp(sal) over (partition by deptno order by sal) from emp;
    ```

    返回结果如下：

    ```
    +------------+------------+------------+
    | deptno     | sal        | _c2        |
    +------------+------------+------------+
    | 10         | 1300       | 0.0        |          --窗口开始行。
    | 10         | 1300       | 0.0        |          --从第1行到第2行的累计样本标准差。
    | 10         | 2450       | 663.9528095680697 |   --从第1行到第3行的累计样本标准差。
    | 10         | 2450       | 663.9528095680696 |
    | 10         | 5000       | 1511.2081259707413 |
    | 10         | 5000       | 1693.7138680032904 |
    | 20         | 800        | 0.0        |
    | 20         | 1100       | 212.13203435596427 |
    | 20         | 2975       | 1178.7175234126282 |
    | 20         | 3000       | 1182.7536725793752 |
    | 20         | 3000       | 1123.3320969330487 |
    | 30         | 950        | 0.0        |
    | 30         | 1250       | 212.13203435596427 |
    | 30         | 1250       | 173.20508075688772 |
    | 30         | 1500       | 225.0      |
    | 30         | 1600       | 253.4758371127315 |
    | 30         | 2850       | 668.331255192114 |
    +------------+------------+------------+
    ```

3.9.6. 聚合函数

聚合（Aggregate）函数的输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值，可以与MaxCompute SQL中的 `group by` 语句配合使用。本文为您提供MaxCompute SQL支持的聚合函数的命令格式、参数说明及示例，指导您使用聚合函数完成开发。

MaxCompute SQL支持的聚合函数如下。

函数	功能
AVG	计算平均值。
COUNT	计算记录数。
COUNT_IF	计算指定表达式为True的记录数。
MAX	计算最大值。
MIN	计算最小值。
MEDIAN	计算中位数。
STDDEV	计算总体标准差。
STDDEV_SAMP	计算样本标准差。
SUM	计算汇总值。
WM_CONCAT	用指定的分隔符连接字符串。
ANY_VALUE	在指定范围内任选一个值返回。
APPROX_DISTINCT	返回输入的非重复值的近似数目。
ARG_MAX	返回指定列的最大值对应行的列值。
ARG_MIN	返回指定列的最小值对应行的列值。
COLLECT_LIST	将指定的列聚合为一个数组。
COLLECT_SET	将指定的列聚合为一个无重复元素的数组。
COVAR_POP	计算指定两个数值列的总体协方差。
COVAR_SAMP	计算指定两个数值列的样本协方差。
NUMERIC_HISTOGRAM	统计指定列的近似直方图。
PERCENTILE	计算精确百分位数，适用于小数据量。
PERCENTILE_APPROX	计算近似百分位数，适用于大数据量。
VARIANCE/VAR_POP	计算指定数值列的方差。
VAR_SAMP	计算指定数值列的样本方差。
BITWISE_OR_AGG	计算输入Value的bit OR聚合值。
MAP_AGG	构造两个输入字段的Map。
MULTIMAP_AGG	构造两个输入字段的Map，第一个字段作为Map的Key，第二个字段构造数组作为Map的Value。
MAP_UNION	对输入Map进行Union操作来构造输出Map。
MAP_UNION_SUM	对输入Map进行Union操作并对相同Key的Value求和来构造输出Map。
HISTOGRAM	构造输入Map的Key值出现次数的Map。

注意事项

升级到MaxCompute 2.0后，产品扩展了部分函数。如果您用到的函数涉及新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP或BINARY），在使用扩展函数时，需要执行如下语句开启新数据类型开关：

- Session级别：如果使用新数据类型，您需要在SQL语句前加上语句 `set odps.sql.type.system.odps2=true;` ，并与SQL语句一起提交执行。
- Project级别：Project Owner可根据需要对Project进行设置，等待10~15分钟后才会生效。命令如下。

```
setproject odps.sql.type.system.odps2=true;
```

对 `setproject` 的详细说明请参见项目空间操作。关于开启Project级别数据类型的注意事项，请参见数据类型版本说明。

在同一条SQL语句中同时使用多个聚合函数时，如果项目资源不足，会出现内存溢出问题，请您根据实际业务情况优化SQL或购买计算资源。

聚合函数语法

聚合函数的语法声明如下。

```
<aggregate_name>(<expression>[,...]) [within group (order by <col1>[,<col2>…])] [filter (where <where_condition>)]
```

- `<aggregate_name>(<expression>[,...])`：内建聚合函数或用户自定义聚合函数UDAF，具体格式以实际聚合函数语法为准。
- `within group (order by <col1>[,<col2>…])`：当聚合函数中携带该表达式时，默认会对`<col1>[,<col2>…]`的输入数据进行升序排列。如果需要降序排列，表达式为`within group (order by <col1>[,<col2>…] [desc])`。

  在使用该表达式时，您需要注意：
  - 仅支持WM_CONCAT、COLLECT_LIST、COLLECT_SET及UDAF使用该表达式。
  - 一个SELECT语句中如果多个聚合函数携带`within group (order by <col1>[,<col2>…])`表达式时，`order by <col1>[,<col2>…]`必须相同。
  - 如果聚合函数的参数中携带了DISTINCT关键字，`order by <col1>[,<col2>…]`中必须包含DISTINCT的列。

  命令示例如下：

```
--示例一，对输入数据升序排列后输出。
select
  x,
  wm_concat(',', y) within group (order by y)
from values('k', 1),('k', 3),('k', 2) as t(x, y)
group by x;
--返回结果如下。
+------------+------------+
| x          | _c1        |
+------------+------------+
| k          | 1,2,3      |
+------------+------------+
--示例二，对输入数据降序排列后输出。
select
  x,
  wm_concat(',', y) within group (order by y desc)
from values('k', 1),('k', 3),('k', 2) as t(x, y)
group by x;
--返回结果如下。
+------------+------------+
| x          | _c1        |
+------------+------------+
| k          | 3,2,1      |
+------------+------------+
```

- `[filter (where <where_condition>)]`：当聚合函数中携带该表达式时，聚合函数只会处理满足`<where_condition>`的数据。更多`<where_condition>`信息，请参见WHERE子句（where_condition）。

  在使用该表达式时，您需要注意：
  - 仅支持内建聚合函数使用该表达式，不支持UDAF使用该表达式。
  - `count(*)`不支持`[filter (where <where_condition>)]`表达式。如果您需要对`count(*)`添加过滤条件，可以使用COUNT_IF。
  - COUNT_IF也不支持`[filter (where <where_condition>)]`表达式。

  命令示例如下：

```
--示例一，过滤并聚合数据。
select
  sum(x),
  sum(x) filter (where y > 1),
  sum(x) filter (where y > 2)
  from values(null, 1),(1, 2),(2, 3),(3, null) as t(x, y);
--返回结果如下。
+------------+------------+------------+
| _c0        | _c1        | _c2        |
+------------+------------+------------+
| 6          | 3          | 2          |
+------------+------------+------------+
--示例二，使用多个聚合函数过滤并聚合数据。
select
  count_if(x > 2),
  sum(x) filter (where y > 1),
  sum(x) filter (where y > 2)
  from values(null, 1),(1, 2),(2, 3),(3, null) as t(x, y);
--返回结果如下。
+------------+------------+------------+
| _c0        | _c1        | _c2        |
+------------+------------+------------+
| 1          | 3          | 2          |
+------------+------------+------------+
```

示例数据

为便于理解各函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表emp，并添加数据，命令示例如下：

```
create table if not exists emp
   (empno bigint,
    ename string,
    job string,
    mgr bigint,
    hiredate datetime,
    sal bigint,
    comm bigint,
    deptno bigint);
tunnel upload emp.txt emp;
```

emp.txt中的数据如下：

```
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

过滤条件表达式

- 使用限制。
  - 仅MaxCompute内置的聚合函数支持添加过滤条件表达式，自定义聚合函数UDAF暂不支持。
  - `count(*)` 不能与过滤条件表达式同时使用，请使用COUNT_IF函数。
- 命令格式。

  ```
  <aggregate_name>(<expression>[,...]) [filter (where <where_condition>)]
  ```

- 命令说明。

  聚合函数都可以增加过滤条件表达式。如果指定了过滤条件，则只有满足过滤条件的行数据才会传给对应的聚合函数进行处理。

- 参数说明。
  - aggregate_name：必填。聚合函数名称，请根据实际需求选择下方的聚合函数。
  - expression：必填。聚合函数的参数。请根据各个聚合函数的参数说明填写。
  - where_condition：可选。过滤条件。更多过滤条件信息，请参见WHERE子句（where_condition）。
- 返回值说明。

  请参见各个聚合函数的返回值说明。
- 使用示例。

```
select sum(sal) filter (where deptno=10), sum(sal) filter (where deptno=20), sum(sal) filter (where deptno=30) from emp;
```

返回结果如下：

```
+------------+------------+------------+
| _c0        | _c1        | _c2        |
+------------+------------+------------+
| 17500      | 10875      | 9400       |
+------------+------------+------------+
```

AVG

- 命令格式。

```
double avg(double <colname>)
decimal avg(decimal <colname>)
```

- 命令说明。

  计算平均值。
- 参数说明。

  colname：必填。列值可以为DOUBLE或DECIMAL类型。如果输入为STRING或BIGINT类型，会隐式转换为DOUBLE类型后参与运算。
- 返回值说明。

  返回DECIMAL或DOUBLE类型。返回规则如下：
  - colname为DECIMAL类型时，返回DECIMAL类型。
  - colname为其他合法的类型时，返回DOUBLE类型。
  - colname为BOOLEAN类型时，不允许参与运算。
  - colname值为NULL时，该行不参与计算。
- 示例。
  - 示例1：计算所有职工的平均薪资（sal）。命令示例如下：

```
select avg(sal) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 2222.0588235294117 |
+------------+
```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门员工的平均薪资（sal）。命令示例如下：

```
select deptno, avg(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 2916.6666666666665 |
| 20         | 2175.0     |
| 30         | 1566.6666666666667 |
+------------+------------+
```

COUNT

- 命令格式。

```
bigint count([distinct|all] <colname>)
```

- 命令说明。

计算记录数。

- 参数说明。
  - distinct|all：可选。表示在计数时是否去除重复记录，默认为all，即计算全部记录。如果指定distinct，则只计算唯一值数量。
  - colname：必填。列值可以为任意类型。colname可以为 `*` ，即 `count(*)` ，返回所有行数。
- 返回值说明。

  返回BIGINT类型。colname值为NULL时，该行不参与计算。
- 示例。
  - 示例1：计算所有部门的总职工人数。命令示例如下：

    ```
    select count(*) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 17         |
    +------------+
    ```
  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门（deptno）的职工人数。命令示例如下：

    ```
    select deptno, count(*) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | 6          |
    | 20         | 5          |
    | 30         | 6          |
    +------------+------------+
    ```
  - 示例3：通过distinct去重，计算部门数量。命令示例如下：

    ```
    select count(distinct deptno) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 3          |
    +------------+
    ```

COUNT_IF

- 命令格式。

  ```
  bigint count_if(boolean <expr>)
  ```
- 命令说明。

  计算expr值为True的记录数。
- 参数说明。

  expr：必填。BOOLEAN类型表达式。
- 返回值说明。

  返回BIGINT类型。expr值为False或expr中指定的列的值为NULL时，该行不参与计算。
- 示例。

  ```
  select count_if(sal > 1000), count_if(sal <=1000) from emp;
  ```

  返回结果如下：

  ```
  +------------+------------+
  | _c0        | _c1        |
  +------------+------------+
  | 15         | 2          |
  +------------+------------+
  ```

MAX

- 命令格式。

```
max(<colname>)
```

- 命令说明。

  计算最大值。

- 参数说明。

  colname：必填。列值可以为除BOOLEAN外的任意类型。

- 返回值说明。

  返回值的类型与colname类型相同。返回规则如下：

  - colname值为NULL时，该行不参与计算。
  - colname为BOOLEAN类型时，不允许参与运算。

- 示例。
  - 示例1：计算所有职工的最高薪资（sal）。命令示例如下：

    ```
    select max(sal) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 5000       |
    +------------+
    ```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门员工的最高薪资（sal）。命令示例如下：

    ```
    select deptno, max(sal) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | 5000       |
    | 20         | 3000       |
    | 30         | 2850       |
    +------------+------------+
    ```

MIN

- 命令格式。

```
min(<colname>)
```

- 命令说明。

  计算最小值。

- 参数说明。

  colname：必填。列值可以为除BOOLEAN外的任意类型。

- 返回值说明。

  返回值的类型与colname类型相同。返回规则如下：

  - colname值为NULL时，该行不参与计算。
  - colname为BOOLEAN类型时，不允许参与运算。

- 示例。
  - 示例1：计算所有职工的最低薪资（sal）。命令示例如下：

    ```
    select min(sal) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 800        |
    +------------+
    ```

- 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门员工的最低薪资（sal）。命令示例如下：

```
select deptno, min(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 1300       |
| 20         | 800        |
| 30         | 950        |
+------------+------------+
```

MEDIAN

- 命令格式。

```
double median(double <colname>)
decimal median(decimal <colname>)
```

- 命令说明。

  计算中位数。

- 参数说明。

  colname：必填。列值可以为DOUBLE或DECIMAL类型。如果输入为STRING或BIGINT类型，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明。

  返回DECIMAL或DOUBLE类型。返回规则如下：

  - colname为DECIMAL类型时，返回DECIMAL类型。
  - colname为其他合法的类型时，返回DOUBLE类型。
  - colname为BOOLEAN类型时，不允许参与运算。
  - colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：计算所有职工的薪资（sal）中位数。命令示例如下：

```
select median(sal) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 1600.0     |
+------------+
```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门员工的薪资（sal）中位数。命令示例如下：

```
select deptno, median(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 2450.0     |
| 20         | 2975.0     |
| 30         | 1375.0     |
+------------+------------+
```

STDDEV

- 命令格式。

```
double stddev(double <colname>)
decimal stddev(decimal <colname>)
```

- 命令说明。

  计算总体标准差。

- 参数说明。

  colname：必填。DOUBLE或DECIMAL类型。如果输入为STRING或BIGINT类型，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明。

返回DECIMAL或DOUBLE类型。返回规则如下：

- colname为DECIMAL类型时，返回DECIMAL类型。
- colname为其他合法的类型时，返回DOUBLE类型。
- colname为BOOLEAN类型时，不允许参与运算。
- colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：计算所有职工的薪资（sal）的总体标准差。命令示例如下：

```
select stddev(sal) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 1262.7549932628976 |
+------------+
```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门员工的薪资（sal）总体标准差。命令示例如下：

```
select deptno, stddev(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 1546.1421524412158 |
| 20         | 1004.7387720198718 |
| 30         | 610.1001739241043 |
+------------+------------+
```

STDDEV_SAMP

- 命令格式。

```
double stddev_samp(double <colname>)
decimal stddev_samp(decimal <colname>)
```

- 命令说明。

  计算样本标准差。

- 参数说明。

  colname：必填。列值可以为DOUBLE或DECIMAL类型。如果输入为STRING或BIGINT类型，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明。

  返回DECIMAL或DOUBLE类型。返回规则如下：

  - colname为DECIMAL类型时，返回DECIMAL类型。
  - colname为其他合法的类型时，返回DOUBLE类型。
  - colname为BOOLEAN类型时，不允许参与运算。
  - colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：计算所有职工的薪资（sal）的样本标准差。命令示例如下：

```
select stddev_samp(sal) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 1301.6180541247609 |
+------------+
```

- 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门员工的薪资（sal）样本标准差。命令示例如下：

```
select deptno, stddev_samp(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 1693.7138680032901 |
| 20         | 1123.3320969330487 |
| 30         | 668.3312551921141 |
+------------+------------+
```

SUM

- 命令格式。

```
sum(<colname>)
```

- 命令说明。

  计算汇总值。

- 参数说明。

  colname：必填。列值可以为DOUBLE、DECIMAL或BIGINT类型。如果输入为STRING类型，会隐式转换为DOUBLE类型后参与运算。

- 返回值说明。

  返回DECIMAL或DOUBLE类型。返回规则如下：

  - colname为DECIMAL类型时，返回DECIMAL类型。
  - colname为其他合法的类型时，返回DOUBLE类型。
  - colname为BOOLEAN类型时，不允许参与运算。
  - colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：计算所有职工的薪资（sal）总和。命令示例如下：

```
select sum(sal) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 37775      |
+------------+
```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，计算各部门员工的薪资（sal）总和。命令示例如下：

```
select deptno, sum(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 17500      |
| 20         | 10875      |
| 30         | 9400       |
+------------+------------+
```

WM_CONCAT

- 命令格式。

```
string wm_concat(string <separator>, string <colname>)
```

- 命令说明。

  用指定的separator做分隔符，连接colname中的值。

- 参数说明。
  - separator：必填。STRING类型常量，分隔符。
  - colname：必填。STRING类型。如果输入为BIGINT、DOUBLE或DATETIME类型，会隐式转换为STRING类型后参与运算。

- 返回值说明。

返回STRING类型。返回规则如下：

- separator非STRING类型常量时，返回报错。
- colname非STRING、BIGINT、DOUBLE或DATETIME类型时，返回报错。
- colname值为NULL时，该行不会参与计算。

> 说明 `select wm_concat(',', name) from table_name;` 语句中，如果 `table_name` 为空集合，该语句返回NULL值。

- 示例。
  - 示例1：对所有职工的姓名（ename）进行合并。命令示例如下：

```
select wm_concat(',', ename) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| SMITH,ALLEN,WARD,JONES,MARTIN,BLAKE,CLARK,SCOTT,KING,TURNER,ADAMS,JAMES,FORD,MILLER,JACCKA,WELAN,TEBAGE |
+------------+
```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并将同组的职工姓名（ename）进行合并。命令示例如下：

```
select deptno, wm_concat(',', ename) from emp group by deptno order by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | CLARK,KING,MILLER,JACCKA,WELAN,TEBAGE |
| 20         | SMITH,JONES,SCOTT,ADAMS,FORD |
| 30         | ALLEN,WARD,MARTIN,BLAKE,TURNER,JAMES |
+------------+------------+
```

  - 示例3：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并将同组的薪资（sal）去重后进行合并。命令示例如下：

```
select deptno, wm_concat(distinct ',', sal) from emp group by deptno order by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 1300,2450,5000 |
| 20         | 1100,2975,3000,800 |
| 30         | 1250,1500,1600,2850,950 |
+------------+------------+
```

ANY_VALUE

- 命令格式。

```
any_value(<colname>)
```

- 命令说明。

  在指定范围内任选一个值返回。此函数为MaxCompute 2.0扩展函数。

- 参数说明。

  colname：必填。可以为任意类型。

- 返回值说明。

  返回值类型同colname对应值类型。colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：在所有职工中任选一名。命令示例如下：

```
select any_value(ename) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| SMITH      |
+------------+
```

- 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并在各组中任选一名。命令示例如下：

  ```
  select deptno, any_value(ename) from emp group by deptno;
  ```

  返回结果如下：

  ```
  +------------+------------+
  | deptno     | _c1        |
  +------------+------------+
  | 10         | CLARK      |
  | 20         | SMITH      |
  | 30         | ALLEN      |
  +------------+------------+
  ```

ARG_MAX

- 命令格式。

  ```
  arg_max(<valueToMaximize>, <valueToReturn>)
  ```

- 命令说明。

  返回valueToMaximize最大值对应行的valueToReturn。此函数为MaxCompute 2.0扩展函数。

- 参数说明。
  - valueToMaximize：必填。可以为任意类型。
  - valueToReturn：必填。可以为任意类型。
- 返回值说明。

  返回值类型和valueToReturn类型相同，如果存在多行最大值时，随机返回最大值中的一行对应的值。valueToMaximize值为NULL时，该行不参与计算。

- 示例。
  - 示例1：返回薪资最高的员工姓名。命令示例如下：

    ```
    select arg_max(sal, ename) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | KING       |
    +------------+
    ```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并返回各组中薪资最高职工姓名。命令示例如下：

    ```
    select deptno, arg_max(sal, ename) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | KING       |
    | 20         | SCOTT      |
    | 30         | BLAKE      |
    +------------+------------+
    ```

ARG_MIN

- 命令格式。

  ```
  arg_min(<valueToMinimize>, <valueToReturn>)
  ```

- 命令说明。

  返回valueToMinimize最小值对应行的valueToReturn。此函数为MaxCompute 2.0扩展函数。

- 参数说明。
  - valueToMinimize：必填。可以为任意类型。
  - valueToReturn：必填。可以为任意类型。
- 返回值说明。

  返回值类型和valueToReturn类型相同，如果存在多行最小值时，随机返回最小值其中的一行对应的值。valueToMinimize值为NULL时，该行不参与计算。

- 示例。

- 示例1：返回薪资最低的员工姓名。命令示例如下：

```
select arg_min(sal, ename) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| SMITH      |
+------------+
```

- 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并返回各组中薪资最低职工姓名。命令示例如下：

```
select deptno, arg_min(sal, ename) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | MILLER     |
| 20         | SMITH      |
| 30         | JAMES      |
+------------+------------+
```

APPROX_DISTINCT

- 命令格式。

```
approx_distinct(<colname>)
```

- 命令说明。

  计算指定列的非重复值的近似数目。此函数为MaxCompute 2.0扩展函数。

- 参数说明。

  colname：必填。需要统计去重的列。

- 返回值说明。

  返回BIGINT类型。此函数会产生5%的标准误差。colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：统计薪资（sal）列非重复值的近似数目。命令示例如下：

```
select approx_distinct(sal) from emp;
```

返回结果如下：

```
+-------------------+
| numdistinctvalues |
+-------------------+
| 12                |
+-------------------+
```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，统计薪资（sal）列非重复值的近似数目。命令示例如下：

```
select deptno, approx_distinct(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+-------------------+
| deptno     | numdistinctvalues |
+------------+-------------------+
| 10         | 3                 |
| 20         | 4                 |
| 30         | 5                 |
+------------+-------------------+
```

COLLECT_LIST

- 命令格式。

```
array collect_list(<colname>)
```

- 命令说明。

  将colname指定的列值聚合为一个数组。此函数为MaxCompute 2.0扩展函数。

- 参数说明。

colname：必填。表的列名称，可为任意类型。

- 返回值说明。

  返回ARRAY类型。colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：将所有职工薪资（sal）聚合为一个数组。命令示例如下：

    ```
    select collect_list(sal) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | [800,1600,1250,2975,1250,2850,2450,3000,5000,1500,1100,950,3000,1300,5000,2450,1300] |
    +------------+
    ```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并将同组的职工薪资（sal）聚合为一个数组。命令示例如下：

    ```
    select deptno, collect_list(sal) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | [2450,5000,1300,5000,2450,1300] |
    | 20         | [800,2975,3000,1100,3000] |
    | 30         | [1600,1250,1250,2850,1500,950] |
    +------------+------------+
    ```

  - 示例3：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并将同组的职工薪资（sal）去重后聚合为一个数组。命令示例如下：

    ```
    select deptno, collect_list(distinct sal) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | [1300,2450,5000] |
    | 20         | [800,1100,2975,3000] |
    | 30         | [950,1250,1500,1600,2850] |
    +------------+------------+
    ```

COLLECT_SET

- 命令格式。

  ```
  array collect_set(<colname>)
  ```

- 命令说明。

  将colname指定的列值聚合为一个无重复元素的数组。此函数为MaxCompute 2.0扩展函数。

- 参数说明。

  colname：必填。表的列名称，可以为任意类型。

- 返回值说明。

  返回ARRAY类型。colname值为NULL时，该行不参与计算。

- 示例。
  - 示例1：将所有职工薪资（sal）聚合为一个无重复值的数组。命令示例如下：

    ```
    select collect_set(sal) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | [800,950,1100,1250,1300,1500,1600,2450,2850,2975,3000,5000] |
    +------------+
    ```

- 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并将同组的职工薪资（sal）聚合为一个无重复值的数组。命令示例如下：

  ```
  select deptno, collect_set(sal) from emp group by deptno;
  ```

  返回结果如下：

  ```
  +------------+------------+
  | deptno     | _c1        |
  +------------+------------+
  | 10         | [1300,2450,5000] |
  | 20         | [800,1100,2975,3000] |
  | 30         | [950,1250,1500,1600,2850] |
  +------------+------------+
  ```

COVAR_POP

- 命令格式。

  ```
  double covar_pop(<colname1>, <colname2>)
  ```

- 命令说明。

  计算指定两个数值列的总体协方差。此函数为MaxCompute 2.0扩展函数。

- 参数说明。

  colname1、colname2：必填。数据类型为数值类型的列。其他类型返回NULL。

- 示例。

  在示例表emp中执行如下命令追加数据：

  ```
  --sal_new为新薪资列。
  alter table emp add columns (sal_new bigint);
  insert overwrite table emp select empno, ename, job, mgr, hiredate, sal, comm, deptno, sal+1000 from emp;
  ```

  - 示例1：计算sal与sal_new的总体协方差。命令示例如下：

    ```
    select covar_pop(sal, sal_new) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 1594550.1730103805 |
    +------------+
    ```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工sal与sal_new的总体协方差。命令示例如下：

    ```
    select deptno, covar_pop(sal, sal_new) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | 2390555.5555555555 |
    | 20         | 1009500.0  |
    | 30         | 372222.2222222222 |
    +------------+------------+
    ```

COVAR_SAMP

- 命令格式。

  ```
  double covar_samp(<colname1>, <colname2>)
  ```

- 命令说明。

  计算指定两个数值列的样本协方差。此函数为MaxCompute 2.0扩展函数。

- 参数说明。

  colname1、colname2：必填。数据类型为数值类型的列。其他类型返回NULL。

- 示例。

- 示例1：计算sal与sal_new的样本协方差。命令示例如下：

```
select covar_samp(sal, sal_new) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 1694209.5588235292 |
+------------+
```

- 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工sal与sal_new的样本协方差。命令示例如下：

```
select deptno, covar_samp(sal, sal_new) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 2868666.6666666665 |
| 20         | 1261875.0  |
| 30         | 446666.6666666666 |
+------------+------------+
```

NUMERIC_HISTOGRAM

- 命令格式。

```
map<double key, double value> numeric_histogram(bigint <buckets>, double <colname>)
```

- 命令说明。

  统计指定列的近似直方图。此函数为MaxCompute 2.0扩展函数。

- 参数说明。
  - buckets：必填。BIGINT类型，表示返回的近似直方图列的最大个数。
  - colname：必填。DOUBLE类型，需要统计近似直方图的列。
- 返回值说明。

  `map<double key, double value>` 类型，返回值中key是近似直方图的X轴坐标点，value是近似直方图的Y轴的近似高度。返回规则如下：
  - buckets值为NULL时，返回NULL。
  - colname值为NULL时，该行不参与计算。
- 示例。

  统计薪资（sal）的近似直方图。命令示例如下：

```
select numeric_histogram(5, sal) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| {"1328.5714285714287":7.0,"2450.0":2.0,"5000.0":2.0,"875.0":2.0,"2956.25":4.0} |
+------------+
```

PERCENTILE

- 命令格式。

```
double percentile(bigint <colname>, <p>)
--以数组形式返回多个百分位精确计算结果。
array percentile(bigint <colname>, array(<p1> [, <p2>...]))
```

- 命令说明。

  计算精确百分位数，适用于小数据量。先对指定列升序排列，然后取精确的第p位百分数。p必须在0和1之间。 `percentile` 是从编号0开始计算，例如某列数据为100、200、300，列数据的编号顺序为0、1、2，计算该列的0.3百分位点， `percentile` 结果是2×0.3=0.6，即值位于编号0和1之间，结果为 `100+(200-100)×0.6=160` 。此函数为MaxCompute 2.0扩展函数。

- 参数说明。
  - colname：必填。值为BIGINT类型的列。
  - p：必填。需要精确的百分位数。取值为 `[0.0,1.0]` 。

- 返回值说明。

  返回DOUBLE或ARRAY类型。

- 示例。
  - 示例1：计算0.3百分位的薪资（sal）。命令示例如下：

    ```
    select percentile(sal, 0.3) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 1290.0     |
    +------------+
    ```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工0.3百分位的薪资（sal）。命令示例如下：

    ```
    select deptno, percentile(sal, 0.3) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | 1875.0     |
    | 20         | 1475.0     |
    | 30         | 1250.0     |
    +------------+------------+
    ```

  - 示例3：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工0.3、0.5、0.8百分位的薪资（sal）。命令示例如下：

    ```
    set odps.sql.type.system.odps2=true;
    select deptno, percentile(sal, array(0.3, 0.5, 0.8)) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | [1875.0,2450.0,5000.0] |
    | 20         | [1475.0,2975.0,3000.0] |
    | 30         | [1250.0,1375.0,1600.0] |
    +------------+------------+
    ```

PERCENTILE_APPROX

- 命令格式。

  ```
  double percentile_approx (double <colname>, <p> [, <B>]))
  --以数组形式返回多个百分位近似计算结果。
  array<double> percentile_approx (double <colname>, array(<p1> [, <p2>...]) [, <B>])
  ```

- 命令说明。

  计算近似百分位数，适用于大数据量。先对指定列升序排列，然后取第p位百分数对应的值。`percentile_approx` 是从编号1开始计算，例如某列数据为100、200、300，列数据的编号顺序为1、2、3，计算该列的0.6百分位点，`percentile_approx` 结果是3×0.6=1.8，即值位于编号1和2之间，结果为 `100+(200-100)×0.8=180` 。此函数为MaxCompute 2.0扩展函数。

  > **说明** `percentile_approx` 与 `percentile` 的区别如下：
  > - `percentile_approx` 用于计算近似的百分位数，`percentile` 用于计算精确的百分位数。在数据量较大时，`percentile` 可能会因内存限制而执行失败，而 `percentile_approx` 无此问题。
  > - `percentile_approx` 的实现与Hive的 `percentile_approx` 函数实现一致，计算的算法与 `percentile` 不同，因此在数据量非常少的部分场景下的执行结果与 `percentile` 的执行结果会有一定差别。

- 参数说明。
  - colname：必填。值为DOUBLE类型的列。
  - p：必填。需要近似的百分位数。取值为 `[0.0,1.0]` 。
  - B：精度参数。精度越高产生的近似值误差越小。如果不设置该参数，默认值为10000。

- 返回值说明。

  返回DOUBLE或ARRAY类型。返回规则如下：

- colname值为NULL时，该行不参与计算。
- p或B值为NULL时，返回报错。

- 示例。
  - 示例1：计算0.3百分位的薪资（sal）。命令示例如下：

    ```
    select percentile_approx(sal, 0.3) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 1252.5     |
    +------------+
    ```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工0.3百分位的薪资（sal）。命令示例如下：

    ```
    select deptno, percentile_approx(sal, 0.3) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | 1300.0     |
    | 20         | 950.0      |
    | 30         | 1070.0     |
    +------------+------------+
    ```

  - 示例3：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工0.3、0.5、0.8百分位的薪资（sal）。命令示例如下：

    ```
    set odps.sql.type.system.odps2=true;
    select deptno, percentile_approx(sal, array(0.3, 0.5, 0.8), 1000) from emp group by deptno;
    ```

    返回结果如下：

    ```
    +------------+------------+
    | deptno     | _c1        |
    +------------+------------+
    | 10         | [1300.0,1875.0,3470.000000000001] |
    | 20         | [950.0,2037.5,2987.5] |
    | 30         | [1070.0,1250.0,1580.0] |
    +------------+------------+
    ```

VARIANCE/VAR_POP

- 命令格式。

  ```
  double variance(<colname>)
  double var_pop(<colname>)
  ```

- 命令说明。

  计算指定数值列的方差。

- 参数说明。

  colname：必填。数据类型为数值的列。参数为其他类型的列返回NULL。此函数为MaxCompute 2.0扩展函数。

- 返回值说明。

  返回DOUBLE类型。

- 示例。
  - 示例1：计算所有职工薪资（sal）的方差。命令示例如下：

    ```
    select variance(sal) from emp;
    --等效于如下语句。
    select var_pop(sal) from emp;
    ```

    返回结果如下：

    ```
    +------------+
    | _c0        |
    +------------+
    | 1594550.1730103805 |
    +------------+
    ```

- 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工薪资（sal）的方差。命令示例如下：

```
select deptno, variance(sal) from emp group by deptno;
--等效于如下语句。
select deptno, var_pop(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 2390555.5555555555 |
| 20         | 1009500.0  |
| 30         | 372222.22222222225 |
+------------+------------+
```

VAR_SAMP

- 命令格式。

```
double var_samp(<colname>)
```

- 命令说明。

  计算指定数值列的样本方差。此函数为MaxCompute 2.0扩展函数。

- 参数说明。

  colname：必填。数据类型为数值的列。其他类型返回NULL。

- 返回值说明。

  返回DOUBLE类型。

- 示例。
  - 示例1：计算所有职工薪资（sal）的样本方差。命令示例如下：

```
select var_samp(sal) from emp;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 1694209.5588235292 |
+------------+
```

  - 示例2：与 `group by` 配合使用，对所有职工按照部门（deptno）进行分组，并计算同组职工薪资（sal）的样本方差。命令示例如下：

```
select deptno, var_samp(sal) from emp group by deptno;
```

返回结果如下：

```
+------------+------------+
| deptno     | _c1        |
+------------+------------+
| 10         | 2868666.666666667 |
| 20         | 1261875.0  |
| 30         | 446666.6666666667 |
+------------+------------+
```

BITWISE_OR_AGG

- 函数声明。

```
bigint bitwise_or_agg(bigint value)
```

- 命令说明。

  对于输入的value，按照bit OR操作计算聚合值。

- 参数说明。

  value：必填。BIGINT类型的值，NULL值不参与计算。

- 返回值说明。

  返回BIGINT类型。

- 示例。

```
select id, bitwise_or_agg(v) from
    values (1L, 2L), (1L, 1L), (2L, null), (1L, null) t(id, v) group by id;
```

返回结果如下。

```
+------------+------------+
| id         | _c1        |
+------------+------------+
| 1          | 3          |
| 2          | NULL       |
+------------+------------+
```

MAP_AGG

- 函数声明。

```
map<K, V> map_agg(K a, V b);
```

- 命令说明。

  使用输入的两个字段分别作为Key和Value来构造Map，以第一个字段作为Map的Key，以第二个字段作为Map的Value。如果key为NULL，则被忽略。如果Key的取值有重复，则会随机只保留其中的一个。

- 参数说明。
  - a：输入字段，将作为Map的Key。
  - b：输入字段，将作为Map的value。
- 返回值说明。

  返回构造的Map。

- 示例。

```
select map_agg(a, b) from
        values (1L, 'apple'), (2L, 'hi'), (null, 'good'), (1L, 'pie') t(a, b);
```

返回结果如下。

```
+------------------------+
| _c0                    |
+------------------------+
| {"2":"hi","1":"apple"} |
+------------------------+
```

MULTIMAP_AGG

- 函数声明。

```
map<K, array<V>> multimap_agg(K a, V b);
```

- 命令说明。

  使用输入的两个字段分别作为Key和Value来构造Map，以第一个字段作为Map的Key，以第二个字段构造数组来作为Map的Value。如果key为NULL，则被忽略。

- 参数说明。
  - a：输入字段，将作为Map的Key。
  - b：输入字段，相同Key的字段被放到同一个数组中作为Map的value。
- 返回值说明。

  返回构造的Map。

- 示例。

```
select multimap_agg(a, b) from
        values (1L, 'apple'), (2L, 'hi'), (null, 'good'), (1L, 'pie') t(a, b);
```

返回结果如下。

```
+-----------------------------------+
| _c0                               |
+-----------------------------------+
| {"2":["hi"],"1":["apple","pie"]} |
+-----------------------------------+
```

MAP_UNION

- 函数声明。

```
map<K, V> map_union(map<K, V> input);
```

- 命令说明。

对输入Map进行Union操作构造输出Map，如果某一个Key在多个输入Map中都存在，则会随机只保留其中的一个。

- 参数说明。

  input：输入Map。

- 返回值说明。

  返回构造的Map。

- 示例。

```
select map_union(a) from values
    (map(1L, 'hi', 2L, 'apple', 3L, 'pie')), (map(1L, 'good', 4L, 'this')), (null) t(a);
```

返回结果如下。

```
+-----------------------------------------------+
| _c0                                           |
+-----------------------------------------------+
| {"4":"this","1":"good","2":"apple","3":"pie"} |
+-----------------------------------------------+
```

MAP_UNION_SUM

- 函数声明。

```
map<K, V> map_union_sum(map<K, V> input);
```

- 命令说明。

  对输入Map进行Union并对同一个Key的Value进行求和操作构造输出Map，如果某一个Key对应的Value为NULL，则将其转换为0。

  > **说明** 输入Map的Value的类型必须是数值类型，例如BININT、INT、SMALLINT、TINYINT、FLOAT、DOUBLE、DECIMAL。

- 参数说明。

  input：输入Map。

- 返回值说明。

  返回构造的Map。

  > **说明** 输出Map的Value类型是BININT、DOUBLE、DECIMAL。

- 示例。

```
select map_union_sum(a) from values
    (map('hi', 2L, 'apple', 3L, 'pie', 1L)), (map('apple', null, 'hi', 4L)), (null) t(a);
```

返回结果如下。

```
+----------------------------+
| _c0                        |
+----------------------------+
| {"apple":3,"hi":6,"pie":1} |
+----------------------------+
```

HISTOGRAM

- 函数声明。

```
map<K, bigint> map_agg(K input);
```

- 命令说明。

  返回一个Map，Map的Key是输入input值，Map的Value是input值出现的次数。NULL值将被忽略。

- 参数说明。

  input：输入字段，将作为Map的Key。

- 返回值说明。

  返回构造的Map，描述每个input值出现的次数。

- 示例。

```
select histogram(a) from values
    ('hi'), (null), ('apple'), ('pie'), ('apple') t(a);
```

返回结果如下。

```
+----------------------------+
| _c0                        |
+----------------------------+
| {"pie":1,"hi":1,"apple":2} |
+----------------------------+
```

3.9.7. 字符串函数

您可以在MaxCompute SQL中使用字符串函数对指定字符串进行灵活处理。本文为您提供MaxCompute SQL支持的字符串函数的命令格式、参数说明及示例，指导您使用字符串函数完成开发。

MaxCompute SQL支持的字符串函数如下。

函数	功能
ASCII	返回字符串的第一个字符的ASCII码。
CHAR_MATCHCOUNT	计算A字符串出现在B字符串中的字符个数。
CHR	将指定ASCII码转换成字符。
CONCAT	将字符串连接在一起。
CONCAT_WS	将参数中的所有字符串按照指定的分隔符连接在一起。
ENCODE	将字符串按照指定编码格式编码。
FIND_IN_SET	在以逗号分隔的字符串中查找指定字符串的位置。
FORMAT_NUMBER	将数字转化为指定格式的字符串。
FROM_JSON	根据给定的JSON字符串和输出格式信息，返回ARRAY、MAP或STRUCT类型。
GET_JSON_OBJECT	在一个标准JSON字符串中，按照指定方式抽取指定的字符串。
INSTR	计算A字符串在B字符串中的位置。
IS_ENCODING	判断字符串是否可以从指定的A字符集转换为B字符集。
KEYVALUE	将字符串拆分为Key-Value对，并将Key-Value对分开，返回Key对应的Value。
KEYVALUE_TUPLE	将字符串拆分为多个Key-Value对，并将Key-Value对分开，返回多个Key对应的Value。
LENGTH	计算字符串的长度。
LENGTHB	计算字符串以字节为单位的长度。
LOCATE	在字符串中查找另一指定字符串的位置。
LTRIM	去除字符串的左端字符。
MD5	计算字符串的MD5值。
PARSE_URL	对URL进行解析返回指定部分的信息。
PARSE_URL_TUPLE	对URL进行解析返回多个部分的信息。
REGEXP_COUNT	计算字符串从指定位置开始，匹配指定规则的子串数。
REGEXP_EXTRACT	将字符串按照指定规则拆分为组后，返回指定组的字符串。
REGEXP_INSTR	返回字符串从指定位置开始，与指定规则匹配指定次数的子串的起始或结束位置。
REGEXP_REPLACE	将字符串中，与指定规则在指定次数匹配的子串替换为另一字符串。
REGEXP_SUBSTR	返回字符串中，从指定位置开始，与指定规则匹配指定次数的子串。
REPEAT	返回将字符串重复指定次数后的结果。
REVERSE	返回倒序字符串。
RTRIM	去除字符串的右端字符。
SPACE	生成空格字符串。

函数	功能
SPLIT_PART	按照分隔符拆分字符串，返回指定部分的子串。
SUBSTR	返回STRING类型字符串从指定位置开始，指定长度的子串。
SUBSTRING	返回STRING或BINARY类型字符串从指定位置开始，指定长度的子串。
TO_CHAR	将BOOLEAN、BIGINT、DECIMAL或DOUBLE类型值转为对应的STRING类型表示。
TO_JSON	将指定的复杂类型输出为JSON字符串。
TOLOWER	将字符串中的英文字符转换为小写形式。
TOUPPER	将字符串中的英文字符转换为大写形式。
TRIM	去除字符串的左右两端字符。
URL_DECODE	将字符串编码为 `application/x-www-form-urlencoded MIME` 格式。
URL_ENCODE	将字符串从 `application/x-www-form-urlencoded MIME` 格式转为常规字符。
JSON_TUPLE	在一个标准的JSON字符串中，按照输入的一组键抽取各个键指定的字符串。
LPAD	将字符串向左补足到指定位数。
RPAD	将字符串向右补足到指定位数。
REPLACE	将字符串中与指定字符串匹配的子串替换为另一字符串。
SOUNDEX	将普通字符串替换为SOUNDEX字符串。
SUBSTRING_INDEX	截取字符串指定分隔符前的字符串。
TRANSLATE	将A出现在B中的字符串替换为C字符串。

注意事项

升级到MaxCompute 2.0后，产品扩展了部分函数。如果您用到的函数涉及新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP或BINARY），在使用扩展函数时，需要执行如下语句开启新数据类型开关：

- Session级别：如果使用新数据类型，您需要在SQL语句前加上语句 `set odps.sql.type.system.odps2=true;` ，并与SQL语句一起提交执行。
- Project级别：Project Owner可根据需要对Project进行设置，等待10~15分钟后才会生效。命令如下。

```
setproject odps.sql.type.system.odps2=true;
```

对 `setproject` 的详细说明请参见项目空间操作。关于开启Project级别数据类型的注意事项，请参见数据类型版本说明。

ASCII

- 命令格式

```
bigint ascii(string <str>)
```

- 命令说明

  返回字符串str第一个字符的ASCII码。

- 参数说明

  str：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - str值为NULL时，返回NULL。

- 示例
  - 示例1：返回字符串 `abcde` 第一个字符的ASCII码。命令示例如下。

```
--返回97。
select ascii('abcde');
```

- 示例2：输入参数为NULL。命令示例如下。

```
--返回NULL。
select ascii(null);
```

CHAR_MATCHCOUNT

- 命令格式

```
bigint char_matchcount(string <str1>, string <str2>)
```

- 命令说明

  计算str1中有多少个字符出现在str2中。

- 参数说明

  str1、str2：必填。STRING类型，必须为有效的UTF-8字符串。如果对比过程中发现有无效字符（非UNICODE编码），则返回负值。

- 返回值说明

  返回BIGINT类型。str1或str2值为NULL时，返回NULL。

- 示例
  - 示例1：计算字符串 `aabc` 出现在 `abcde` 中的字符个数。命令示例如下。

```
--返回4。
select char_matchcount('aabc','abcde');
```

  - 示例2：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select char_matchcount(null,'abcde');
```

CHR

- 命令格式

```
string chr(bigint <ascii>)
```

- 命令说明

  将指定ASCII码转换为字符。

- 参数说明

  ascii：必填。BIGINT类型的ASCII值。取值范围为0~128。如果输入为STRING、DOUBLE或DECIMAL类型，则会隐式转换为BIGINT类型后参与运算。

- 返回值说明

  返回STRING类型。返回规则如下：
  - ascii值不在取值范围内时，返回报错。
  - ascii非BIGINT、STRING、DOUBLE或DECIMAL类型时，返回报错。
  - ascii值为NULL时，返回NULL。

- 示例
  - 示例1：将ASCII码 `100` 转换为字符。命令示例如下。

```
--返回d。
select chr(100);
```

  - 示例2：输入参数为NULL。命令示例如下。

```
--返回NULL。
select chr(null);
```

  - 示例3：输入为STRING类型字符。命令示例如下。

```
--隐式转换为BIGINT类型后参与运算，返回d。
select chr('100');
```

CONCAT

- 命令格式

```
array<T> concat(array<T> <a>, array<T> <b>[,...])
string concat(string <str1>, string <str2>[,...])
```

- 命令说明
  - 输入为ARRAY数组：将多个ARRAY数组中的所有元素连接在一起，生成一个新的ARRAY数组。
  - 输入为字符串：将多个字符串连接在一起，生成一个新的字符串。

- 参数说明
  - a、b：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。a和b中元素的数据类型必须一致。数组中的元素为NULL值时会参与运算。
  - str1、str2：必填。STRING类型。如果输入参数为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
- 返回值说明
  - 返回ARRAY类型。如果任一输入ARRAY数组为NULL，返回结果为NULL。
  - 返回STRING类型。如果没有参数或任一参数为NULL，返回结果为NULL。
- 示例
  - 示例1：连接ARRAY数组`array(10, 20)`和`array(20, -20)`。命令示例如下。

    ```
    --返回[10, 20, 20, -20]。
    select concat(array(10, 20), array(20, -20));
    ```

  - 示例2：ARRAY数组元素包含NULL。命令示例如下。

    ```
    --返回[10, NULL, 20, -20]。
    select concat(array(10, null), array(20, -20));
    ```

  - 示例3：任一ARRAY数组为NULL。命令示例如下。

    ```
    --返回NULL。
    select concat(array(10, 20), null);
    ```

  - 示例4：连接字符串`aabc`和`abcde`。命令示例如下。

    ```
    --返回aabcabcde。
    select concat('aabc','abcde');
    ```

  - 示例5：输入为空。命令示例如下。

    ```
    --返回NULL。
    select concat();
    ```

  - 示例6：任一字符串输入为NULL。命令示例如下。

    ```
    --返回NULL。
    select concat('aabc', 'abcde', null);
    ```

CONCAT_WS

- 命令格式

```
string concat_ws(string <separator>, string <str1>, string <str2>[,...])
string concat_ws(string <separator>, array<string> <a>)
```

- 命令说明

  返回将参数中的所有字符串或ARRAY数组中的元素按照指定的分隔符连接在一起的结果。此函数为MaxCompute 2.0扩展函数。
- 参数说明
  - separator：必填。STRING类型的分隔符。
  - str1、str2：至少要指定2个字符串。STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - a：必填。ARRAY数组。数组中的元素为STRING类型。
- 返回值说明

  返回STRING类型或STRUCT类型。返回规则如下：
  - str1或str2非STRING、BIGINT、DECIMAL、DOUBLE或DATETIME类型时，返回报错。
  - 如果没有输入参数或任一输入参数值为NULL，返回NULL。
- 示例
  - 示例1：将字符串`name`和`hanmeimei`通过`:`连接。命令示例如下。

    ```
    --返回name:hanmeimei。
    select concat_ws(':','name','hanmeimei');
    ```

    -
  - 示例2：任一输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select concat_ws(':','avg',null,'34');
    ```

- 示例3：将ARRAY数组 `array('name', 'hanmeimei')` 中的元素通过 `:` 连接。命令示例如下。

```
--返回name:hanmeimei。
select concat_ws(':',array('name', 'hanmeimei'));
```

ENCODE

- 命令格式

```
binary encode(string <str>, string <charset>)
```

- 命令说明

  将str按照charset格式进行编码。

- 参数说明
  - str：必填。STRING类型。待重新编码的字符串。
  - charset：必填。STRING类型。编码格式。取值范围为：UTF-8、UTF-16、UTF-16LE、UTF-16BE、ISO-8859-1、US-ASCII。
- 返回值说明

  返回BINARY类型。str或charset值为NULL时，返回NULL。

- 示例
  - 示例1：将字符串 `abc` 按照 `UTF-8` 格式编码。命令示例如下。

```
--返回abc。
select encode("abc", "UTF-8");
```

  - 示例2：将字符串 `abc` 按照 `UTF-16BE` 格式编码。命令示例如下。

```
--返回=00a=00b=00c。
select encode("abc", "UTF-16BE");
```

  - 示例3：任一输入参数为NULL。命令示例如下。

```
--返回结果为NULL。
select encode("abc", null);
```

FIND_IN_SET

- 命令格式

```
bigint find_in_set(string <str1>, string <str2>)
```

- 命令说明

  查找字符串str1在以逗号（,）分隔的字符串str2中的位置，从1开始计数。

- 参数说明
  - str1：必填。STRING类型。待查找的字符串。
  - str2：必填。STRING类型。以逗号（,）分隔的字符串。
- 返回值说明

  返回BIGINT类型。返回规则如下：
  - 当str2中无法匹配到str1或str1中包含逗号（,）时，返回0。
  - 当str1或str2值为NULL时，返回NULL。
- 示例
  - 示例1：查找字符串 `ab` 在字符串 `abc,hello,ab,c` 中的位置。命令示例如下。

```
--返回3。
select find_in_set('ab', 'abc,hello,ab,c');
```

  - 示例2：查找字符串 `hi` 在字符串 `abc,hello,ab,c` 中的位置。命令示例如下。

```
--返回0。
select find_in_set('hi', 'abc,hello,ab,c');
```

  - 示例3：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select find_in_set(null, 'abc,hello,ab,c');
```

FORMAT_NUMBER

- 命令格式

```
string format_number(float|double|decimal <expr1>, int <expr2>)
```

- 命令说明

  将expr1转化为满足expr2格式的字符串。

- 参数说明
  - expr1：必填。FLOAT、DOUBLE、DECIMAL类型。需要格式化的数据。
  - expr2：必填。INT类型，取值范围为0~340。指代需要保留的小数位数。也可以为类似 `#,###,###.##` 格式的描述。不同取值返回的小数位数不同。

- 返回值说明

  返回STRING类型。返回规则如下：
  - 当 `0<expr2≤340` 时，四舍五入到小数点后指定位数。
  - 当 `expr2=0` 时，只保留整数，无小数点或小数部分。
  - 当 `expr2<0` 或 `expr2>340` 时，会返回报错。
  - expr1或expr2值为空或NULL时，返回NULL。

- 示例
  - 示例1：对给定数字按照指定格式输出。命令示例如下。

    ```
    --返回5.230。
    select format_number(5.230134523424545456,3);
    --返回12,332.123。
    select format_number(12332.123456, '#,###,###,###.###');
    ```

  - 示例2：任一输入参数为空或NULL。命令示例如下。

    ```
    --返回NULL。
    select format_number('',3);
    --返回NULL。
    select format_number(null,3);
    ```

FROM_JSON

- 命令格式

```
from_json(<jsonStr>, <schema>)
```

- 命令说明

  根据JSON字符串jsonStr和schema信息，返回ARRAY、MAP或STRUCT类型。

- 参数说明
  - jsonStr：必填。输入的JSON字符串。
  - schema：必填。写法与建表语句的类型一致。例如 `array<bigint>` 、 `map<string, array<string>>` 或 `struct<a:int, b:double, `C`:map<string,string>>` 。

    > **说明** STRUCT的Key区分大小写。此外，STRUCT类型还有一种写法 `a bigint, b double` ，等同于 `struct<a:bigint, b:double>` 。

    JSON数据类型与MaxCompute数据类型的对应关系如下。

JSON数据类型	MaxCompute数据类型
OBJECT	STRUCT、MAP、STRING
ARRAY	ARRAY、STRING
NUMBER	TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、STRING
BOOLEAN	BOOLEAN、STRING
STRING	STRING、CHAR、VARCHAR、BINARY、DATE、DATETIME
NULL	所有类型

> **说明** 对于OBJECT和ARRAY类型，会采用尽可能解析的方式，不匹配的类型会忽略。为了便于使用，所有的JSON类型都可以转换为MaxCompute的STRING类型。同时您需要注意，对应FLOAT、DOUBLE、DECIMAL三种数据类型无法保证小数的精度，如果需要确保精度可以先用STRING类型取出数据，再转换为对应数值类型。

- 返回值说明

  返回ARRAY、MAP或STRUCT类型。

- 示例
  - 示例1：将指定JSON字符串以指定格式输出。命令示例如下。

```
--返回{"a":1,"b":0.8}。
select from_json('{"a":1, "b":0.8}', 'a int, b double');
--返回{"time":"26/08/2015"}。
select from_json('{"time":"26/08/2015"}', 'time string');
--返回{"a":1,"b":0.8}。
select from_json('{"a":1, "b":0.8}', 'a int, b double, c string');
--返回[1,2,3]。
select from_json('[1, 2, 3, "a"]', 'array<bigint>');
--返回{"d":"v","a":"1","b":"[1,2,3]","c":"{}"}。
select from_json('{"a":1,"b":[1,2,3],"c":{},"d":"v"}', 'map<string, string>');
```

  - 示例2：使用 `map_keys` 函数和 `from_json` 函数实现类似JSON_KEYS的作用，获取JSON字符串中所有key的集合。命令示例如下。

```
--返回["a","b"]。
select map_keys(from_json('{"a":1,"b":2}','map<string,string>'));
```

GET_JSON_OBJECT

- 命令格式

```
string get_json_object(string <json>, string <path>)
```

- 命令说明

  在一个标准JSON字符串中，按照path抽取指定的字符串。每次调用该函数时，都会读一次原始数据，因此反复调用可能影响性能和产生费用。您可以通过 `get_json_object` ，结合UDTF，轻松转换JSON格式日志数据，避免多次调用函数，详情请参见利用MaxCompute内建函数及UDTF转换JSON格式日志数据。

- 参数说明
  - json：必填。STRING类型。标准的JSON格式对象，格式为 `{Key:Value, Key:Value,...}` 。如果遇到英文双引号（"），需要用两个反斜杠（\\）进行转义。如果遇到英文单引号（'），需要用一个反斜杠（\）进行转义。
  - path：必填。STRING类型。表示在json中的path，以 `$` 开头。更多path信息，请参见LanguageManual UDF。相关最佳实践案例，请参见JSON数据从OSS迁移至MaxCompute。不同字符的含义如下：
    - `$` ：表示根节点。
    - `.` 或 `['']` ：表示子节点。MaxCompute支持用这两种字符解析JSON对象，当JSON的Key本身包含 `.` 时，可以用 `['']` 来替代。
    - `[]` ： `[number]` 表示数组下标，从0开始。
    - `*` ： `Wildcard for []` ，返回整个数组。 `*` 不支持转义。

- 返回值说明
  - 如果json为空或非法的json格式，返回NULL。
  - 如果json合法，path也存在，则返回对应字符串。

- 您可以通过在Session级别设置 `odps.sql.udf.getjsonobj.new` 属性来控制函数的返回方式：
  - 当设置 `set odps.sql.udf.getjsonobj.new=true;` 时，函数返回行为采用了保留原始字符串的方式进行输出。

    推荐您使用此配置，函数返回行为更标准，处理数据更方便，性能更好。**如果MaxCompute项目有使用JSON保留字符转义行为的存量作业，建议保留原有行为方式，避免因未验证而直接使用该行为产生错误或正确性问题。**函数返回行为规则如下：
    - 返回值仍是一个JSON字符串，可以继续当做JSON来解析，而不再需要额外使用 `replace` 或 `regexp_replace` 等函数替换反斜线。
    - 一个JSON对象中可以出现相同的Key，可以成功解析。

      ```
      --返回1。
      select get_json_object('{"a":"1","a":"2"}', '$.a');
      ```
    - 支持Emoji表情符号对应的编码字符串。但DataWorks暂不支持输入Emoji表情符号，仅支持通过数据集成等工具直接将Emoji表情符号对应的编码字符串写入MaxCompute，再用 `get_json_object` 函数处理。

      ```
      --返回Emoji符号。
      select get_json_object('{"a":"<Emoji符号>"}', '$.a');
      ```
    - 输出结果按照JSON字符串的原始排序方式输出。

      ```
      --返回{"b":"1","a":"2"}。
      select get_json_object('{"b":"1","a":"2"}', '$');
      ```
  - 当设置 `set odps.sql.udf.getjsonobj.new=false;` 时，函数返回行为采用了JSON保留字符转义的方式进行输出。函数返回行为规则如下：
    - 换行符（\n）、引号（"）等JSON保留字符使用字符串 `'\n'` 、 `'\"'` 显示。
    - 一个JSON对象中不可以出现相同的Key，可能导致无法解析。

      ```
      --返回NULL。
      select get_json_object('{"a":"1","a":"2"}', '$.a');
      ```
    - 不支持解析Emoji表情符号编码的字符串。

      ```
      --返回NULL。
      select get_json_object('{"a":"<Emoji符号>"}', '$.a');
      ```
    - 输出结果按照字典排序方式输出。

      ```
      --返回{"a":"2","b":"1"}。
      select get_json_object('{"b":"1","a":"2"}', '$');
      ```

  > **说明** 自2021年1月21日及之后新创建的MaxCompute项目中，`get_json_object` 函数的返回行为默认为保留原始字符串。2021年1月21日之前创建的MaxCompute项目中，`get_json_object` 函数的返回行为默认为JSON保留字符转义。您可以通过以下示例判断MaxCompute项目中 `get_json_object` 函数采用了哪种行为，执行命令如下：
  >
  > ```
  > select get_json_object('{"a":"[\\"1\\"]"}', '$.a');
  > --JSON保留字符转义的行为返回：
  > [\"1\"]
  > --保留原始字符串的行为返回：
  > ["1"]
  > ```
  >
  > 您可以提工单联系MaxCompute技术支持团队，将您的项目中的 `get_json_object` 函数返回行为切换为保留原始字符串的行为，避免在Session级别频繁设置属性。

- 示例
  - 示例1：提取JSON对象 `src_json.json` 中的信息。命令示例如下。

    ```
    --JSON对象src_json.json的内容。
    +----+
    json
    +----+
    {"store":
    {"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],
    "bicycle":{"price":19.95,"color":"red"}
    },
    "email":"amy@only_for_json_udf_test.net",
    "owner":"amy"
    }
    --提取owner字段信息，返回amy。
    select get_json_object(src_json.json, '$.owner') from src_json;
    --提取store.fruit字段第一个数组信息，返回{"weight":8,"type":"apple"}。
    select get_json_object(src_json.json, '$.store.fruit[0]') from src_json;
    --提取不存在的字段信息，返回NULL。
    select get_json_object(src_json.json, '$.non_exist_key') from src_json;
    ```

- 示例2：提取数组型JSON对象的信息。命令示例如下。

```
--返回2222。
select get_json_object('{"array":[["aaaa",1111],["bbbb",2222],["cccc",3333]]}','$.array[1][1]');
--返回["h0","h1","h2"]。
set odps.sql.udf.getjsonobj.new=true;
select get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[*]');
--返回["h0","h1","h2"]。
set odps.sql.udf.getjsonobj.new=false;
select get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh');
--返回h1。
select get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[1]');
```

- 示例3：提取带有 `.` 的JSON对象中的信息。命令示例如下。

```
--创建一张表。
create table mf_json (id string, json string);
--向表中插入数据，Key带.。
insert into table mf_json (id, json) values ("1", "{
\"China.beijing\":{\"school\":{\"id\":0,\"book\":[{\"title\": \"A\",
\"price\": 8.95},{\"title\": \"B\",\"price\": 10.2}]}}}");
--向表中插入数据，Key不带.。
insert into table mf_json (id, json) values ("2", "{
\"China_beijing\":{\"school\":{\"id\":0,\"book\":[{\"title\": \"A\",
\"price\": 8.95},{\"title\": \"B\",\"price\": 10.2}]}}}");
--取id的值，查询key为China.beijing，返回0。由于包含.，只能用['']来解析。
select get_json_object(json, "$['China.beijing'].school['id']") from mf_json where id =1;
--取id的值，查询key为China_beijing，返回0。查询方法有如下两种。
select get_json_object(json, "$['China_beijing'].school['id']") from mf_json where id =2;
select get_json_object(json, "$.China_beijing.school['id']") from mf_json where id =2;
```

- 示例4：JSON输入为空或非法格式。命令示例如下。

```
--返回NULL。
select get_json_object('','$.array[1][1]');
--返回NULL。
select get_json_object('"array":["aaaa",1111],"bbbb":["cccc",3333]','$.array[1][1]');
```

- 示例5：JSON字符串涉及转义。命令示例如下。

```
set odps.sql.udf.getjsonobj.new=true;
--返回"1"。
select get_json_object('{"a":"\\"1\\"","b":"2"}', '$.a');
--返回'1'。
select get_json_object('{"a":"\'1\'","b":"2"}', '$.a');
```

INSTR

- 命令格式

```
bigint instr(string <str1>, string <str2>[, bigint <start_position>[, bigint <nth_appearance>]])
```

- 命令说明

  计算子串str2在字符串str1中的位置。

- 参数说明
  - str1：必填。STRING类型。待搜索的目标字符串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
  - str2：必填。STRING类型。待匹配的子串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
  - start_position：可选。BIGINT类型，其他类型会返回报错。表示从str1的第几个字符开始搜索，默认起始位置是第一个字符位置1。
    当start_position为负数时表示开始位置是从字符串的结尾往前倒数，最后一个字符是-1，依次往前倒数。
  - nth_appearance：可选。BIGINT类型，大于0。表示str2在str1中第nth_appearance次匹配的位置。如果nth_appearance为其他类型或小于等于0，则返回报错。

- 返回值说明

  返回BIGINT类型。返回规则如下：
  - 如果在str1中未找到str2，则返回0。
  - 如果str2为空串，则总能匹配成功，例如 `select instr('abc','');` 会返回1。
  - str1、str2、start_position或nth_appearance值为NULL时，返回NULL。

- 示例

- 示例1：计算字符 `e` 在字符串 `Tech on the net` 中的位置。命令示例如下。

```
--返回2。
select instr('Tech on the net', 'e');
```

- 示例2：计算子串 `on` 在字符串 `Tech on the net` 中的位置。命令示例如下。

```
--返回6。
select instr('Tech on the net', 'on');
```

- 示例3：计算字符 `e` 在字符串 `Tech on the net` 中，从第3个字符开始，第2次出现的位置。命令示例如下。

```
--返回14。
select instr('Tech on the net', 'e', 3, 2);
```

- 示例4：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select instr('Tech on the net', null);
```

IS_ENCODING

- 命令格式

```
boolean is_encoding(string <str>, string <from_encoding>, string <to_encoding>)
```

- 命令说明

  判断输入的字符串str是否可以从指定的一个字符集from_encoding转为另一个字符集to_encoding。也可以用于判断输入是否为乱码，通常您可以将from_encoding设为UTF-8，to_encoding设为GBK。

- 参数说明
  - str：必填。STRING类型。空字符串可以被认为属于任何字符集。
  - from_encoding、to_encoding：必填。STRING类型，源及目标字符集。

- 返回值说明

  返回BOOLEAN类型。返回规则如下：
  - 如果str能够成功转换，则返回True，否则返回False。
  - str、from_encoding或to_encoding值为NULL时，返回NULL。

- 示例
  - 示例1：判断字符 `测试` 或 `測試` 是否可以从 `utf-8` 字符集转为 `gbk` 字符集。命令示例如下。

```
--返回true。
select is_encoding('测试', 'utf-8', 'gbk');
--返回true。
select is_encoding('測試', 'utf-8', 'gbk');
```

  - 示例2：判断字符 `測試` 是否可以从 `utf-8` 字符集转为 `gb2312` 字符集。命令示例如下。

```
--GB2312字库中不包括这两个字，返回false。
select is_encoding('測試', 'utf-8', 'gb2312');
```

  - 示例3：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select is_encoding('测试', null, 'gbk');
```

KEYVALUE

- 命令格式

```
keyvalue(string <str>,[string <split1>,string <split2>,] string <key>)
keyvalue(string <str>,string <key>)
```

- 命令说明

  将字符串str按照split1分成Key-Value对，并按split2将Key-Value对分开，返回key所对应的Value。

- 参数说明
  - str：必填。STRING类型。待拆分的字符串。
  - split1、split2：可选。STRING类型。用于作为分隔符的字符串，按照指定的两个分隔符拆分源字符串。如果表达式中没有指定这两项，默认split1为 `";"` ，split2为 `":"` 。当某个被split1拆分后的字符串中有多个split2时，返回结果未定义。
  - key：必填。STRING类型。将字符串按照split1和split2拆分后，返回key值对应的Value。

- 返回值说明

返回STRING类型。返回规则如下：

- split1或split2值为NULL时，返回NULL。
- str或key值为NULL或没有匹配的key时，返回NULL。
- 如果有多个Key-Value匹配，返回第一个匹配上的key对应的Value。

- 示例
  - 示例1：将字符串 `0:1\;1:2` 拆分为Key-Value对，返回Key值 `1` 对应的Value。命令示例如下。

    ```
    --返回2。
    select keyvalue('0:1\;1:2', 1);
    ```

    没有指定split1和split2，默认split1为 `";"` ，split2为 `":"` 。

    经过split1拆分后，Key-Value对为 `0:1\,1:2` 。经过split2拆分后变为如下。

    ```
    0 1/
    1 2
    ```

    返回Key为1所对应的Value值2。
  - 示例2：将字符串 `"\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;"` 按照 `"\;"` 拆分为Key-Value对，再按照 `":"` 将Key-Value分开，返回Key值 `tf` 对应的Value。命令示例如下。

    ```
    --返回21910。
    select keyvalue("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;","
    \;",":","tf");
    ```

    将 `"\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;"` 按照 `"\;"` 拆分后，得出的Key-Value对如下所示。

    ```
    decreaseStore:1, xcard:1, isB2C:1, tf:21910, cart:1, shipping:2, pf:0, market:shoes, instPayAmount:0
    ```

    按照 `":"` 拆分后，结果如下所示。

    ```
    decreaseStore 1
    xcard 1
    isB2C 1
    tf 21910
    cart 1
    shipping 2
    pf 0
    market shoes
    instPayAmount 0
    ```

    返回Key为 `tf` 对应的Value值21910。

KEYVALUE_TUPLE

- 命令格式

```
KEYVALUE_TUPLE(str, split1, split2, key1, key2, ..., keyN)
```

- 命令说明

  将字符串str按照split1分成Key-Value对，并按split2将Key-Value对分开，返回多个key所对应的Value。

- 参数说明
  - str：必填。STRING类型。待拆分的字符串。
  - split1、split2：必填。STRING类型。用于作为分隔符的字符串，按照指定的两个分隔符拆分源字符串。当某个被split1拆分后的字符串中有多个split2时，返回结果未定义。
  - key：必填。STRING类型。将字符串按照split1和split2拆分后，返回key值对应的Value。

- 返回值说明

  返回STRING类型。返回规则如下：
  - split1或split2值为NULL时，返回NULL。
  - str或key值为NULL或没有匹配的key时，返回NULL。

- 示例

```
--创建表
create table mf_user (
user_id string,
user_info string
);
--插入数据
insert into mf_user values('1','age:18;genda:f;address:abc'),('2','age:20;genda:m;address:bcd');
--查询
SELECT user_id,
KEYVALUE(user_info,';',':','age') as age,
KEYVALUE(user_info,';',':','genda') as genda,
KEYVALUE(user_info,';',':','address') as address
FROM mf_user;
--等同于使用KEYVALUE查询
SELECT user_id,
age,
genda,
address
FROM mf_user LATERAL VIEW KEYVALUE_TUPLE(user_info,';', ':','age','genda','address') ui AS age,genda,address;
```

结果如下所示。

```
+------------+------------+------------+------------+
| user_id    | age        | genda      | address    |
+------------+------------+------------+------------+
| 1          | 18         | f          | abc        |
| 2          | 20         | m          | bcd        |
+------------+------------+------------+------------+
```

LENGTH

- 命令格式

```
bigint length(string <str>)
```

- 命令说明

  计算字符串str的长度。

- 参数说明

  str：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - str值为NULL时，返回NULL。
  - str为非UTF-8编码格式时，返回-1。

- 示例
  - 示例1：计算字符串 `Tech on the net` 的长度。命令示例如下。

    ```
    --返回15。
    select length('Tech on the net');
    ```

  - 示例2：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select length(null);
    ```

LENGTHB

- 命令格式

```
bigint lengthb(string <str>)
```

- 命令说明

  计算字符串str以字节为单位的长度。

- 参数说明

  str：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。

- str值为NULL时，返回NULL。

- 示例
  - 示例1：计算字符串 `Tech on the net` 以字节为单位的长度。命令示例如下。

    ```
    --返回15。
    select lengthb('Tech on the net');
    ```

  - 示例2：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select lengthb(null);
    ```

LOCATE

- 命令格式

```
bigint locate(string <substr>, string <str>[, bigint <start_pos>])
```

- 命令说明

  在str中查找substr的位置。您可以通过start_pos指定开始查找的位置，从1开始计数。

- 参数说明
  - substr：必填。STRING类型。待查找的字符串。
  - str：必填。STRING类型。待匹配的字符串。
  - start_pos：可选。BIGINT类型。指定查找的起始位置。

- 返回值说明

  返回为BIGINT类型。返回规则如下：
  - str中无法匹配到substr时，返回0。
  - str或substr值为NULL时，返回NULL。
  - start_pos值为NULL时，返回0。

- 示例
  - 示例1：查找字符串 `ab` 在字符串 `abchelloabc` 中的位置。命令示例如下。

    ```
    --返回1。
    select locate('ab', 'abchelloabc');
    ```

  - 示例2：查找字符串 `hi` 在字符串 `abchelloabc` 中的位置。命令示例如下。

    ```
    --返回0。
    select locate('hi', 'abc,hello,ab,c');
    ```

  - 示例3：start_pos为NULL。命令示例如下。

    ```
    --返回0。
    select locate('ab', 'abhelloabc', null);
    ```

LTRIM

- 命令格式

```
string ltrim(string <str>[, <trimChars>])
string trim(leading [<trimChars>] from <str>)
```

- 命令说明

  从str的左端去除字符：
  - 如果未指定trimChars，则默认去除空格字符。
  - 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的左端去除尽可能长的所有字符都在集合trimChars中的子串。

- 参数说明
  - str：必填。STRING类型。待去除左端字符的字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - trimChars：可选。String类型。待去除的字符。

- 返回值说明

  返回为STRING类型。返回规则如下：
  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - str或trimChars值为NULL时，返回NULL。

- 示例
  - 示例1：去除字符串 `yxTxyomxx` 的左边空格。命令示例如下。

```
--返回字符串yxTxyomxx 。
select ltrim(' yxTxyomxx ');
--等效于如下语句。
select trim(leading from ' yxTxyomxx ');
```

  - 示例2：去除字符串 `yxTxyomxx` 左端所有字符都在集合 `xy` 中的子串。

```
--返回Txyomxx，只要左端遇到x或者y就会被去掉。
select ltrim('yxTxyomxx', 'xy');
--等效于如下语句。
select trim(leading 'xy' from 'yxTxyomxx');
```

  - 示例3：输入参数为NULL。命令示例如下。

```
--返回NULL。
select ltrim(null);
select ltrim('yxTxyomxx', null);
```

MD5

- 命令格式

```
string md5(string <str>)
```

- 命令说明

  计算字符串str的MD5值。

- 参数说明

  str：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型参与运算。

- 返回值说明

  返回STRING类型。返回规则如下：

  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - str值为NULL时，返回NULL。

- 示例
  - 示例1：计算字符串 `Tech on the net` 的MD5值。命令示例如下。

```
--返回ddc4c4796880633333d77a60fcda9af6。
select md5('Tech on the net');
```

  - 示例2：输入参数为NULL。命令示例如下。

```
--返回NULL。
select md5(null);
```

PARSE_URL

- 命令格式

```
string parse_url(string <url>, string <part>[, string <key>])
```

- 命令说明

  对url解析后，按照part提取信息。

- 参数说明
  - url：必填。STRING类型。URL链接。无效URL链接会返回报错。
  - part：必填。STRING类型。取值包含：HOST、PATH、QUERY、REF、PROTOCOL、AUTHORITY、FILE和USERINFO，不区分大小写。
  - key：可选。当part取值为QUERY时，根据key值取出对应的Value值。

- 返回说明

  返回STRING类型。返回规则如下：

  - url、part或key值为NULL时，返回NULL。
  - part取值不符合要求时，返回报错。

- 示例

```
--返回example.com。
select parse_url('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'HOST');
--返回/over/there/index.dtb。
select parse_url('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'PATH');
--返回animal。
select parse_url('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'QUERY', 'typ
e');
--返回nose。
select parse_url('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'REF');
--返回file。
select parse_url('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'PROTOCOL');
--返回 username:password@example.com:8042。
select parse_url('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'AUTHORITY');
--返回username:password。
select parse_url('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'USERINFO');
```

PARSE_URL_TUPLE

- 命令格式

```
string parse_url_tuple(string <url>, string <key1>, string <key2>,...)
```

- 命令说明

  对url解析后，按照输入的一组键key1、key2等抽取各个键指定的字符串。该功能与PARSE_URL类似，但它可以同时提取多个键对应的字符串，性能更优。

- 参数说明
  - url：必填。STRING类型。URL链接。无效URL链接会返回报错。
  - key1、key2：必填。STRING类型。指定要抽取的键。取值范围如下：
    - HOST：获取主机地址，可以为域名或IP地址。
    - PATH：获取网络资源在服务器中的路径。
    - QUERY：查询字符串，指代待查询的内容。
    - REF：获取URL注释。鼠标悬停至URL链接上时显示的内容。
    - PROTOCOL：获取协议类型。
    - AUTHORITY：获取服务器的域名或IP地址、端口号和用户鉴权信息（例如用户名、密码）。
    - FILE：获取网络资源在服务器中的路径和待查询的内容，即由PATH和QUERY组成。
    - USERINFO：用户鉴权信息。
    - QUERY:<KEY>：获取查询字符串中指定字段（Key）的取值。

    不区分大小写。不在该范围内的取值会返回报错。

- 返回说明

  返回STRING类型。url或key值为NULL时，返回报错。

- 示例

  抽取 `file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose` 中各个键对应的字符串。命令示例如下。

```
select parse_url_tuple('file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose', 'HOST',
'PATH', 'QUERY', 'REF', 'PROTOCOL', 'AUTHORITY', 'FILE', 'USERINFO', 'QUERY:type', 'QUERY:name') as (item0, item1, item2, item
3, item4, item5, item6, item7, item8, item9);
```

  返回结果如下。

```
+------------+------------+------------+------------+------------+------------+------------+------------+------------+--------
----+
| item0      | item1      | item2      | item3      | item4      | item5      | item6      | item7      | item8      | item9
|
+------------+------------+------------+------------+------------+------------+------------+------------+------------+--------
----+
| example.com | /over/there/index.dtb | type=animal&name=narwhal | nose       | file       | username:password@example.com:804
2 | /over/there/index.dtb?type=animal&name=narwhal | username:password | animal     | narwhal    |
+------------+------------+------------+------------+------------+------------+------------+------------+------------+--------
----+
```

REGEXP_COUNT

- 命令格式

```
bigint regexp_count(string <source>, string <pattern>[, bigint <start_position>])
```

- 命令说明

计算source中从start_position位置开始，匹配指定pattern的子串数。

- 参数说明
  - source：必填。STRING类型。待搜索的字符串，其他类型会返回报错。
  - pattern：必填。STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。pattern为空串或其他类型时返回报错。
  - start_position：可选。BIGINT类型常量，必须大于0。其他类型或值小于等于0时返回报错。不指定时默认为1，表示从source的第一个字符开始匹配。
- 返回值说明

  返回BIGINT类型。返回规则如下：
  - 如果没有匹配成功，返回0。
  - source、pattern或start_position值为NULL时，返回NULL。
- 示例
  - 示例1：计算 `abababc` 中从指定位置开始，匹配指定规则的子串数。命令示例如下。

```
--返回1。
select regexp_count('abababc', 'a.c');
--返回2。
select regexp_count('abababc', '[[:alpha:]]{2}', 3);
```

  - 示例2：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select regexp_count('abababc', null);
```

  - 示例3：计算 `:` 出现在JSON字符串 `{"account_id":123456789,"account_name":"allen","location":"hangzhou","bill":100}` 中的次数。命令示例如下。

```
--返回4。
select regexp_count('{"account_id":123456789,"account_name":"allen","location":"hangzhou","bill":100}',':');
```

REGEXP_EXTRACT

- 命令格式

```
string regexp_extract(string <source>, string <pattern>[, bigint <groupid>])
```

- 命令说明

  将字符串source按照pattern的分组规则进行字符串匹配，返回第groupid个组匹配到的字符串内容。
- 参数说明
  - source：必填。STRING类型，待拆分的字符串。
  - pattern：必填。STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。
  - groupid：可选。BIGINT类型常量，必须大于等于0。

> **说明** 数据以UTF-8格式保存。对于中文字符可以用对应的16进制编码表示，中文字符编码的范围是[\\x{4e00},\\x{9fa5}]。

- 返回值说明

  返回STRING类型。返回规则如下：
  - 如果pattern为空串或pattern中没有分组，返回报错。
  - groupid非BIGINT类型或小于0时，返回报错。不指定时默认为1，表示返回第一个组。如果groupid等于0，则返回满足整个pattern的子串。
  - source、pattern或groupid值为NULL时，返回NULL。
- 示例
  - 示例1：将 `foothebar` 按照 `foo(.*?)(bar)` 拆分。命令示例如下。

```
--返回the。
select regexp_extract('foothebar', 'foo(.*?)(bar)');
--返回foothebar。
select regexp_extract('foothebar', 'foo(.*?)(bar)', 0);
```

  - 示例2：将 `8d99d8` 按照 `8d(\\d+)d8` 拆分。命令示例如下。

```
--返回99。在MaxCompute客户端上提交正则计算的SQL，需要使用两个"\"作为转义字符。
select regexp_extract('8d99d8', '8d(\\d+)d8');
```

- 示例3：提取 `【阿里云】aliyun` 中的中文字符及标点。命令示例如下。

```
--返回【阿里云】。
select regexp_extract('【阿里云】aliyun', '([^\\x{00}-\\x{ff}]+)');
```

- 示例4：提取 `【阿里云】aliyun` 中的中文字符。命令示例如下。

```
--返回阿里云。
select regexp_extract('【阿里云】aliyun', '([\\x{4e00}-\\x{9fa5}]+)');
```

- 示例5：提取 `【阿里云】aliyun阿里云` 中的中文字符，该场景较为复杂，无法通过REGEXP_EXTRACT函数实现，需要通过REGEXP_REPLACE函数实现。命令示例如下。

```
--返回【阿里云】阿里云。
select regexp_replace('【阿里云】aliyun阿里云', '([\\x{00}-\\x{ff}])', '');
```

- 示例6：没有分组。**错误命令示例**如下：

```
select regexp_extract('foothebar', 'foothebar');
```

REGEXP_INSTR

- 命令格式

```
bigint regexp_instr(string <source>, string <pattern>[,bigint <start_position>[, bigint <occurrence>[, bigint <return_option>]
]])
```

- 命令说明

  计算字符串source从start_position开始，与pattern第occurrence次匹配的子串的起始或结束位置。

- 参数说明
  - source：必填。STRING类型。源字符串。
  - pattern：必填。STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。pattern为空串时返回报错。
  - start_position：可选。BIGINT类型常量。搜索的开始位置。不指定时默认值为1。
  - occurrence：可选。BIGINT类型常量。指定匹配次数，不指定时默认值为1，表示搜索第一次出现的位置。
  - return_option：可选。BIGINT类型常量。指定返回的位置。值为0或1，不指定时默认值为0，其他类型或不允许的值会返回报错。0表示返回匹配的开始位置，1表示返回匹配的结束位置。

- 返回值说明

  返回BIGINT类型。return_option指定匹配的子串在source中的开始或结束位置。返回规则如下：
  - 如果pattern为空串，返回报错。
  - start_position或occurrence非BIGINT类型或小于等于0时，返回报错。
  - source、pattern、start_position、occurrence或return_option值为NULL时，返回NULL。

- 示例
  - 示例1：计算字符串 `i love www.taobao.com` 从 `3` 开始，与 `o[[:alpha:]]{1}` 第 `2` 次匹配的子串开始位置。命令示例如下。

```
--返回14。
select regexp_instr('i love www.taobao.com', 'o[[:alpha:]]{1}', 3, 2);
```

  - 示例2：计算字符串 `i love www.taobao.com` 从 `3` 开始，与 `o[[:alpha:]]{1}` 第 `2` 次匹配的子串结束位置。命令示例如下。

```
--返回16。
select regexp_instr('i love www.taobao.com', 'o[[:alpha:]]{1}', 3, 2, 1);
```

  - 示例3：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select regexp_instr('i love www.taobao.com', null, 3, 2);
```

REGEXP_REPLACE

- 命令格式

```
string regexp_replace(string <source>, string <pattern>, string <replace_string>[, bigint <occurrence>])
```

- 命令说明

  将source字符串中第occurrence次匹配pattern的子串替换成指定字符串replace_string后返回结果字符串。

- 参数说明
  - source：必填。STRING类型，待替换的字符串。

- pattern：必填。STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。pattern为空串时返回报错。
- replace_string：必填。STRING类型，将匹配pattern的字符串替换后的字符串。
- occurrence：可选。BIGINT类型常量，必须大于等于0，表示将第occurrence次匹配的字符串替换为replace_string，为0时表示替换所有匹配的子串。为其他类型或小于0时，返回报错。默认值为0。

- 返回值说明

  返回STRING类型。返回规则如下：

  - 当引用不存在的组时，不进行替换。
  - 如果replace_string值为NULL且pattern有匹配，返回NULL。
  - 如果replace_string值为NULL但pattern不匹配，返回原字符串。
  - source、pattern或occurrence值为NULL时，返回NULL。

- 示例
  - 示例1：将 `123.456.7890` 字符串中与 `([[:digit:]]{3})\\.([[:digit:]]{3})\\.([[:digit:]]{4})` 匹配的所有字符串替换为 `(\\1)\\2-\\3` 。命令示例如下。

    ```
    --返回(123)456-7890。
    select regexp_replace('123.456.7890', '([[:digit:]]{3})\\.([[:digit:]]{3})\\.([[:digit:]]{4})',
    '(\\1)\\2-\\3', 0);
    ```

  - 示例2：将 `abcd` 字符串中与指定规则匹配的字符串进行替换。命令示例如下。

    ```
    --返回a b c d。
    select regexp_replace('abcd', '(.)', '\\1 ', 0);
    --返回a bcd。
    select regexp_replace('abcd', '(.)', '\\1 ', 1);
    --返回abcd。
    select regexp_replace('abcd', '(.)', '\\2', 1);
    ```

  - 示例3：假设表url_set中列名为url的数据格式为 `www.simple@xxx.com` ，且每行的 `xxx` 完全不同，现需要将列中 `www` 后的所有内容都替换掉。命令示例如下。

    ```
    --返回结果为wwwtest。
    select regexp_replace(url,'(www)(.*)','wwwtest',0) from url_set;
    ```

  - 示例4：任一输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select regexp_replace('abcd', '(.)', null, 0);
    ```

  - 示例5：引用不存在的组。命令示例如下。

    ```
    --因为pattern中只定义了一个组，引用的第二个组不存在。
    --请避免这样使用，引用不存在的组的结果未定义。
    regexp_replace("abcd", "(.*)(.)$", "\\2", 0) = "d"
    --因为在pattern中没有组的定义，所以\1引用了不存在的组，
    --请避免这样使用，引用不存在的组的结果未定义。
    regexp_replace("abcd", "a", "\\1", 0) = "bcd"
    ```

REGEXP_SUBSTR

- 命令格式

  ```
  string regexp_substr(string <source>, string <pattern>[, bigint <start_position>[, bigint <occurrence>]])
  ```

- 命令说明

  返回从start_position位置开始，source中第occurrence次匹配指定pattern的子串。

- 参数说明
  - source：必填。STRING类型。待搜索的字符串。
  - pattern：必填。STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。
  - start_position：可选。其他BIGINT常量，必须大于0。不指定时默认为1，表示从source的第一个字符开始匹配。
  - occurrence：可选。BIGINT常量，必须大于0。不指定时默认为1，表示返回第一次匹配的子串。

- 返回值说明

  返回STRING类型。返回规则如下：

  - 如果pattern为空串，返回报错。
  - 没有匹配时，返回NULL。
  - start_position或occurrence非BIGINT类型或小于等于0时，返回报错。
  - source、pattern、start_position、occurrence或return_option值为NULL时，返回NULL。

- 示例
  - 示例1：返回 `I love aliyun very much` 字符串中与指定规则匹配的字符串。命令示例如下。

```
--返回aliyun。
select regexp_substr('I love aliyun very much', 'a[[:alpha:]]{5}');
--返回have。
select regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 1);
--返回2。
select regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 2);
```

  - 示例2：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select regexp_substr('I love aliyun very much', null);
```

REPEAT

- 命令格式

```
string repeat(string <str>, bigint <n>)
```

- 命令说明

  返回将str重复n次后的字符串。

- 参数说明
  - str：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - n：必填。BIGINT类型。长度不超过2 MB。
- 返回值说明

  返回STRING类型。返回规则如下：
  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - n为空时，返回报错。
  - str或n值为NULL时，返回NULL。
- 示例
  - 示例1：将字符串 `abc` 重复 `5` 次。命令示例如下。

```
--返回abcabcabcabcabc。
select repeat('abc', 5);
```

  - 示例2：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select repeat('abc', null);
```

REVERSE

- 命令格式

```
string reverse(string <str>)
```

- 命令说明

  返回倒序字符串。

- 参数说明

  str：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
- 返回值说明

  返回STRING类型。返回规则如下：
  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - str值为NULL时，返回NULL。
- 示例
  - 示例1：返回字符串 `I love aliyun very much` 的倒序。命令示例如下。

```
--返回字符串hcum yrev nuyila evol I。
select reverse('I love aliyun very much');
```

  - 示例2：输入参数为NULL。命令示例如下。

```
--返回NULL。
select reverse(null);
```

RTRIM

- 命令格式

```
string rtrim(string <str>[, <trimChars>])
string trim(trailing [<trimChars>] from <str>)
```

- 命令说明

  从str的右端去除字符：

  - 如果未指定trimChars，则默认去除空格字符。
  - 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的右端去除尽可能长的所有字符都在集合trimChars中的子串。

- 参数说明
  - str：必填。STRING类型。待去除右端字符的字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - trimChars：可选。String类型。待去除的字符。
- 返回值说明

  返回为STRING类型。返回规则如下：

  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - str或trimChars值为NULL时，返回NULL。
- 示例
  - 示例1：去除字符串 `yxTxyomxx` 的右边空格。命令示例如下。

    ```
    --返回字符串 yxTxyomxx。
    select rtrim(' yxTxyomxx ');
    --等效于如下语句。
    select trim(trailing from ' yxTxyomxx ');
    ```

  - 示例2：去除字符串 `yxTxyomxx` 右端所有字符都在集合 `xy` 中的子串。

    ```
    --返回yxTxyom，只要右端遇到x或者y就会被去掉。
    select ltrim('yxTxyomxx', 'xy');
    --等效于如下语句。
    select trim(trailing 'xy' from 'yxTxyomxx');
    ```

  - 示例3：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select rtrim(null);
    select ltrim('yxTxyomxx', 'null');
    ```

SPACE

- 命令格式

```
string space(bigint <n>)
```

- 命令说明

  生成空格字符串，长度为n。

- 参数说明

  n：必填。BIGINT类型。长度不超过2 MB。

- 返回值说明

  返回STRING类型。返回规则如下：

  - n为空时，返回报错。
  - n值为NULL时，返回NULL。
- 示例

```
--返回10。
select length(space(10));
```

SPLIT_PART

- 命令格式

```
string split_part(string <str>, string <separator>, bigint <start>[, bigint <end>])
```

- 命令说明

依照分隔符separator拆分字符串str，返回从start部分到end部分的子串（闭区间）。

- 参数说明
  - str：必填。STRING类型。待拆分的字符串。如果是BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - separator：必填。STRING类型常量。拆分用的分隔符，可以是一个字符，也可以是一个字符串。
  - start：必填。BIGINT类型常量，必须大于0。表示返回段的开始编号（从1开始）。
  - end：BIGINT类型常量，大于等于start。表示返回段的截止编号，可省略，缺省时表示和start取值相等，返回start指定的段。
- 返回值说明

  返回STRING类型。返回规则如下：
  - 如果start的值大于切分后实际的分段数，例如字符串拆分完有6个片段，start大于6，返回空串。
  - 如果separator不存在于str中，且start指定为1，返回整个str。如果str为空串，则输出空串。
  - 如果separator为空串，则返回原字符串str。
  - 如果end大于片段个数，返回从start开始的子串。
  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - separator非STRING类型常量时，返回报错。
  - start或end非BIGINT类型常量时，返回报错。
  - 除separator外，如果任一参数值为NULL，返回NULL。
- 示例
  - 示例1：依照分隔符 `,` 拆分字符串 `a,b,c,d` ，返回指定部分的子串。命令示例如下。

    ```
    --返回a。
    select split_part('a,b,c,d', ',', 1);
    --返回a,b。
    select split_part('a,b,c,d', ',', 1, 2);
    ```
  - 示例2：start的值大于切分后实际的分段数。命令示例如下。

    ```
    --返回空串。
    select split_part('a,b,c,d', ',', 10);
    ```
  - 示例3：separator不存在于str中。命令示例如下。

    ```
    --返回a,b,c,d。
    select split_part('a,b,c,d', ':', 1);
    --返回空串。
    select split_part('a,b,c,d', ':', 2);
    ```
  - 示例4：separator为空串。命令示例如下。

    ```
    --返回a,b,c,d。
    select split_part('a,b,c,d', '', 1);
    ```
  - 示例5：end的值大于切分后实际的分段数。命令示例如下。

    ```
    --返回b,c,d。
    select split_part('a,b,c,d', ',', 2, 6);
    ```
  - 示例6：除separator外，任一输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select split_part('a,b,c,d', ',', null);
    ```

SUBSTR

- 命令格式

```
string substr(string <str>, bigint <start_position>[, bigint <length>])
```

- 命令说明

  返回字符串str从start_position开始，长度为length的子串。
- 参数说明
  - str：必填。STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - start_position：必填。BIGINT类型，默认起始位置为1。
    - Hive兼容数据类型版本：当start_position为0时，与起始位置为1时相同。
    - 1.0和2.0数据类型版本：当start_position为0时，返回NULL。
  - length：可选。BIGINT类型，表示子串的长度值。值必须大于0。
- 返回值说明

返回STRING类型。返回规则如下：

- str非STRING、BIGINT、DECIMAL、DOUBLE或DATETIME类型时，返回报错。
- length非BIGINT类型或值小于等于0时，返回报错。
- 当length被省略时，返回到str结尾的子串。
- str、start_position或length值为NULL时，返回NULL。

- 示例
  - 示例1：返回字符串 `abc` 从指定位置开始，指定长度的子串。命令示例如下。

```
--返回bc。
select substr('abc', 2);
--返回b。
select substr('abc', 2, 1);
--返回bc。
select substr('abc',-2 , 2);
```

  - 示例2：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select substr('abc', null);
```

SUBSTRING

- 命令格式

```
string substring(string|binary <str>, int <start_position>[, int <length>])
```

- 命令说明

  返回字符串str从start_position开始，长度为length的子串。

- 参数说明
  - str：必填。STRING或BINARY类型。
  - start_position：必填。INT类型，起始位置为1。当start_position为0时，返回空串。当start_position为负数时，表示开始位置是从字符串的结尾往前倒数，最后一个字符是-1，依次往前倒数。
  - length：可选。BIGINT类型，表示子串的长度值。值必须大于0。

- 返回值说明

  返回STRING类型。返回规则如下：
  - str非STRING或BINARY类型时，返回报错。
  - length非BIGINT类型或值小于等于0时，返回报错。
  - 当length被省略时，返回到str结尾的子串。
  - str、start_position或length值为NULL时，返回NULL。

- 示例
  - 示例1：返回字符串 `abc` 从指定位置开始，指定长度的子串。命令示例如下。

```
--返回bc。
select substring('abc', 2);
--返回b。
select substring('abc', 2, 1);
--返回bc。
select substring('abc',-2,2);
--返回ab。
select substring('abc',-3,2);
--返回001。
substring(bin(2345), 2, 3);
```

  - 示例2：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select substring('abc', null, null);
```

TO_CHAR

- 命令格式

```
string to_char(boolean <value>)
string to_char(bigint <value>)
string to_char(double <value>)
string to_char(decimal <value>)
```

- 命令说明

将BOOLEAN、BIGINT、DECIMAL或DOUBLE类型值转换为对应的STRING类型表示。

- 参数说明

  value：必填。BOOLEAN、BIGINT、DECIMAL或DOUBLE类型。

- 返回值说明

  返回STRING类型。返回规则如下：

  - value非BOOLEAN、BIGINT、DECIMAL或DOUBLE类型时，返回报错。
  - value值为NULL时，返回NULL。

- 示例
  - 示例1：将如下值转换为STRING类型表示。命令示例如下。

```
--返回字符串123。
select to_char(123);
--返回字符串TRUE。
select to_char(true);
--返回字符串1.23。
select to_char(1.23);
```

  - 示例2：输入参数为NULL。命令示例如下。

```
--返回NULL。
select to_char(null);
```

TO_JSON

- 命令格式

```
to_json(<expr>)
```

- 命令说明

  将给定的复杂类型expr，以JSON字符串格式输出。

- 参数说明

  expr：必填。ARRAY、MAP、STRUCT复杂类型。

> 说明 如果输入为STRUCT类型（`struct<key1:value1, key2:value2`）：
>
> - 转换为JSON字符串时，Key会全部转为小写。
> - `value`如果为NULL，则不输出`value`本组的数据。例如`value2`为NULL，则`key2:value2`不会输出到JSON字符串。

- 示例
  - 示例1：将指定复杂类型以指定格式输出。命令示例如下。

```
--返回{"a":1,"b":2}。
select to_json(named_struct('a', 1, 'b', 2));
--返回{"time":"26/08/2015"}。
select to_json(named_struct('time', "26/08/2015"));
--返回[{"a":1,"b":2}]。
select to_json(array(named_struct('a', 1, 'b', 2)));
--返回{"a":{"b":1}}。
select to_json(map('a', named_struct('b', 1)));
--返回{"a":1}。
select to_json(map('a', 1));
--返回[{"a":1}]。
select to_json(array((map('a', 1))));
```

  - 示例2：输入为STRUCT类型的特殊情况。命令示例如下。

```
--返回{"a":"B"}。STRUCT类型转换为JSON字符串时，key会全部转为小写。
select to_json(named_struct("A", "B"));
--返回{"k2":"v2"}。NULL值所在组的数据，不会输出到JSON字符串。
select to_json(named_struct("k1", cast(null as string), "k2", "v2"));
```

TOLOWER

- 命令格式

```
string tolower(string <source>)
```

- 命令说明

  将字符串source中的大写字符转换为对应的小写字符。

- 参数说明

  source：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。

- 返回值说明

  返回STRING类型。返回规则如下：

  - source非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - source值为NULL时，返回NULL。

- 示例
  - 示例1：将字符串中的大写字符转换为小写字符。命令示例如下。

    ```
    --返回abcd。
    select tolower('aBcd');
    --返回中国fighting。
    select tolower('中国Fighting');
    ```

  - 示例2：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select tolower(null);
    ```

TOUPPER

- 命令格式

  ```
  string toupper(string <source>)
  ```

- 命令说明

  将字符串source中的小写字符转换为对应的大写字符。

- 参数说明

  source：必填。STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。

- 返回值说明

  返回STRING类型。返回规则如下：

  - source非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
  - source值为NULL时，返回NULL。

- 示例
  - 示例1：将字符串中的小写字符转换为大写字符。命令示例如下。

    ```
    --返回ABCD。
    select toupper('aBcd');
    --返回中国FIGHTING。
    select toupper('中国Fighting');
    ```

  - 示例2：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select toupper(null);
    ```

TRIM

- 命令格式

  ```
  string trim(string <str>[,<trimChars>])
  string trim([BOTH] [<trimChars>] from <str>)
  ```

- 命令说明

  从str的左右两端去除字符：

  - 如果未指定trimChars，则默认去除空格字符。
  - 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的左右两端去除尽可能长的所有字符都在集合trimChars中的子串。

- 参数说明
  - str：必填。STRING类型。待去除左右两端字符的字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
  - trimChars：可选。String类型。待去除的字符。

- 返回值说明

  返回为STRING类型。返回规则如下：

  - str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。

- str或trimChars值为NULL时，返回NULL。

- 示例
  - 示例1：去除字符串 `yxTxyomxx` 的左右空格。命令示例如下。

```
--返回字符串yxTxyomxx。
select trim(' yxTxyomxx ');
--等效于如下语句。
select trim(both from ' yxTxyomxx ');
select trim(from ' yxTxyomxx ');
```

  - 示例2：去除字符串 `yxTxyomxx` 左右两端所有字符都在集合 `xy` 中的子串。

```
--返回Txyom，只要左右两端遇到x或者y就会被去掉。
select trim('yxTxyomxx', 'xy');
--等效于如下语句。
select trim(both 'xy' from 'yxTxyomxx');
select trim('xy' from 'yxTxyomxx');
```

  - 示例3：输入参数为NULL。命令示例如下。

```
--返回NULL。
select trim(null);
select trim('yxTxyomxx', null);
```

URL_DECODE

- 命令格式

```
string url_decode(string <input>[, string <encoding>])
```

- 命令说明

  将输入字符串从 `application/x-www-form-urlencoded MIME` 格式转为常规字符串，是 `url_encode` 的逆过程。编码规则如下：
  - a~z、A~Z保持不变。
  - 英文句点（.）、短划线（-）、星号（*）和下划线（_）保持不变。
  - 加号（+）转为空格。
  - `%xy` 格式的序列转为对应的字节值，连续的字节值根据输入的encoding名称解码为对应的字符串。
  - 其余的字符保持不变。

- 参数说明
  - input：必填。STRING类型。要输入的字符串。
  - encoding：可选。指定编码格式，支持GBK或UTF-8等标准编码格式，不输入默认为UTF-8。

- 返回值说明

  返回STRING类型UTF-8编码的字符串。input或encoding值为NULL时，返回NULL。

- 示例

```
--返回示例for url_decode:// (fdsf)。
select url_decode('%E7%A4%BA%E4%BE%8Bfor+url_decode%3A%2F%2F+%28fdsf%29');
--返回Example for URL_DECODE:// dsf(fasfs)。
select url_decode('Example+for+url_decode+%3A%2F%2F+dsf%28fasfs%29', 'GBK');
```

URL_ENCODE

- 命令格式

```
string url_encode(string <input>[, string <encoding>])
```

- 命令说明

  将输入字符串编码为 `application/x-www-form-urlencoded MIME` 格式。编码格式如下：
  - a~z、A~Z保持不变。
  - 英文句点（.）、短划线（-）、星号（*）和下划线（_）保持不变。
  - 空格转为加号（+）。
  - 其余字符根据指定的encoding转为字节值，然后将每个字节值表示为 `%xy` 的格式，`xy` 是该字符的十六进制表示方式。

- 参数说明
  - input：必填。STRING类型。要输入的字符串。
  - encoding：可选。指定编码格式，支持GBK或UTF-8等标准编码格式，不输入默认为UTF-8。

- 返回值说明

返回STRING类型。input或encoding值为NULL时，返回NULL。

- 示例

```
--返回%E7%A4%BA%E4%BE%8Bfor+url_encode%3A%2F%2F+%28fdsf%29。
select url_encode('示例for url_encode:// (fdsf)');
--返回Example+for+url_encode+%3A%2F%2F+dsf%28fasfs%29。
select url_encode('Example for url_encode:// dsf(fasfs)', 'GBK');
```

JSON_TUPLE

- 命令格式

```
string json_tuple(string <json>, string <key1>, string <key2>,...)
```

- 命令说明

用于一个标准的JSON字符串中，按照输入的一组键 `(key1,key2,...)` 抽取各个键指定的字符串。

- 参数说明
  - json：必填。STRING类型，标准的JSON格式字符串。
  - key：必填。STRING类型，用于描述在JSON中的 `path` ，一次可输入多个，不能以美元符号（$）开头。MaxCompute支持用 `.` 或 `['']` 这两种字符解析JSON，当JSON的Key本身包含 `.` 时，可以用 `['']` 来替代。

- 返回值

返回STRING类型。

> 说明
> - 如果JSON为空或者为非法的JSON格式，返回NULL。
> - 如果键Key为空或者不合法（JSON中不存在）返回NULL。
> - 如果JSON合法，键Key也存在，则返回对应字符串。
> - 支持包含中文的JSON数据解析。
> - 支持多层嵌套的JSON数据解析。
> - 支持包含多重嵌套的数组的JSON数据解析。
> - 解析行为和设置了 `set odps.sql.udf.getjsonobj.new=true;` 后的GET_JSON_OBJECT的行为保持一致。在需要对同一个JSON字符串多次解析的情况下，相比于多次调用GET_JSON_OBJECT，JSON_TUPLE可以一次输入多个Key，且JSON字符串只被解析一次，效率更高。
> - JSON_TUPLE是UDTF，在需要选取其他列时应配合Lateral View使用。

- 示例

```
--创建一张表school。
create table school (id string, json string);
--向表中插入数据。
insert into school (id, json) values ("1", "{
\"school\": \"湖畔大学\", \"地址\":\"杭州\", \"SchoolRank\": \"00\",
\"Class1\":{\"Student\":[{\"studentId\":1,\"scoreRankIn3Year\":[1,2,[3,2,6]]},
{\"studentId\":2,\"scoreRankIn3Year\":[2,3,[4,3,1]]}]}}");
```

  - 示例1：提取JSON对象信息。命令示例如下。

```
select json_tuple(school.json,"SchoolRank","Class1") as (item0, item1) from school;
--等效于如下语句。
select get_json_object(school.json,"$.SchoolRank") item0,get_json_object(school.json,"$.Class1") item1 from school;
--返回结果如下。
+-------+-------+
| item0 | item1 |
+-------+-------+
| 00    | {"Student":[{"studentId":1,"scoreRankIn3Year":[1,2,[3,2,6]]},{"studentId":2,"scoreRankIn3Year":[2,3,[4,3,1]]}]} |
+-------+-------+
```

  - 示例2：使用 `['']` 提取JSON对象信息。命令示例如下。

```
select json_tuple(school.json,"school","['Class1'].Student") as (item0, item2) from school where id=1;
--返回结果如下。
+-------+-------+
| item0 | item2 |
+-------+-------+
| 湖畔大学 | [{"studentId":1,"scoreRankIn3Year":[1,2,[3,2,6]]},{"studentId":2,"scoreRankIn3Year":[2,3,[4,3,1]]}] |
```

- 示例3：支持解析包含中文的JSON数据。命令示例如下。

```
select json_tuple(school.json,"校名","地址") as (item0,item1) from school;
--返回结果如下。
+-------+-------+
| item0 | item1 |
+-------+-------+
| 湖畔大学 | 杭州 |
+-------+-------+
```

- 示例4：支持解析多层嵌套的JSON数据。命令示例如下。

```
select sc.Id, q.item0, q.item1
from school sc lateral view json_tuple(sc.json,"Class1.Student[*].studentId","Class1.Student[0].scoreRankIn3Year") q as item
0,item1;
--返回结果如下。
+------------+-------+-------+
| id         | item0 | item1 |
+------------+-------+-------+
| 1          | [1,2] | [1,2,[3,2,6]] |
+------------+-------+-------+
```

- 示例5：支持解析包含多重嵌套数组的JSON数据。命令示例如下。

```
select sc.Id, q.item0, q.item1
from school sc lateral view json_tuple(sc.json,"Class1.Student[0].scoreRankIn3Year[2]","Class1.Student[0].scoreRankIn3Year[2
][1]") q as item0,item1;
--返回结果如下。
+------------+-------+-------+
| id         | item0 | item1 |
+------------+-------+-------+
| 1          | [3,2,6] | 2     |
+------------+-------+-------+
```

LPAD

- 命令格式

```
string lpad(string <str1>, int <length>, string <str2>)
```

- 命令说明

  用字符串str2将字符串str1向左补足到length位。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - str1：必填。STRING类型。待向左补位的字符串。
  - length：必填。INT类型。向左补位位数。
  - str2：必填。用于补位的字符串。

- 返回值说明

  返回STRING类型。返回规则如下：
  - 如果length小于str1的位数，则返回str1从左开始截取length位的字符串。
  - 如果length为0，则返回空串。
  - 如果没有输入参数或任一输入参数值为NULL，返回NULL。

- 示例
  - 示例1：用字符串`12`将字符串`abcdefgh`向左补足到10位。命令示例如下。

```
--返回12abcdefgh。
select lpad('abcdefgh', 10, '12');
```

  - 示例2：用字符串`12`将字符串`abcdefgh`向左补足到5位。命令示例如下。

```
--返回abcde。
select lpad('abcdefgh', 5, '12');
```

  - 示例3：length为0。命令示例如下。

```
--返回空串。
select lpad('abcdefgh' ,0, '12');
```

  - 示例4：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select lpad(null ,0, '12');
```

RPAD

- 命令格式

```
string rpad(string <str1>, int <length>, string <str2>)
```

- 命令说明

  用字符串str2将字符串str1向右补足到length位。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - str1：必填。STRING类型。待向右补位的字符串。
  - length：必填。INT类型。向右补位位数。
  - str2：必填。用于补位的字符串。

- 返回值说明

  返回STRING类型。返回规则如下：
  - 如果length小于str1的位数，则返回str1从左开始截取length位的字符串。
  - 如果length为0，则返回空串。
  - 如果没有输入参数或任一输入参数值为NULL，返回NULL。

- 示例
  - 示例1：用字符串`12`将字符串`abcdefgh`向右补足到10位。命令示例如下。

    ```
    --返回abcdefgh12。
    select rpad('abcdefgh', 10, '12');
    ```

  - 示例2：用字符串`12`将字符串`abcdefgh`向右补足到5位。命令示例如下。

    ```
    --返回abcde。
    select rpad('abcdefgh', 5, '12');
    ```

  - 示例3：length为0。命令示例如下。

    ```
    --返回空串。
    select rpad('abcdefgh' ,0, '12');
    ```

  - 示例4：任一输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select rpad(null ,0, '12');
    ```

REPLACE

- 命令格式

```
string replace(string <str>, string <old>, string <new>)
```

- 命令说明

  用new字符串替换str字符串中与old字符串完全重合的部分并返回替换后的str。如果没有重合的字符串，返回原str。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - str：必填。STRING类型。待替换的字符串。
  - old：必填。待比较的字符串。
  - new：必填。替换后的字符串。

- 返回值说明

  返回STRING类型。如果任一输入参数值为NULL，返回NULL。

- 示例
  - 示例1：用字符串`12`替换字符串`abababab`中与字符串`abab`完全重合的部分。命令示例如下。

    ```
    --返回12ab。
    select replace('abababab','abab','12');
    ```

  - 示例2：任一输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select replace('123abab456ab',null,'abab');
    ```

SOUNDEX

- 命令格式

```
string soundex(string <str>)
```

- 命令说明

  将普通字符串转换为SOUNDEX字符串。

- 参数说明

  str：必填。STRING类型。待转换的字符串。此函数为MaxCompute 2.0扩展函数。

- 返回值说明

  返回STRING类型。str值为NULL时，返回NULL。

- 示例
  - 示例1：将字符串 `hello` 转换为SOUNDEX字符串。命令示例如下。

    ```
    --返回H400。
    select soundex('hello');
    ```

  - 示例2：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select soundex(null);
    ```

SUBSTRING_INDEX

- 命令格式

```
string substring_index(string <str>, string <separator>, int <count>)
```

- 命令说明

  截取字符串str第count个分隔符之前的字符串。如果count为正，则从左边开始截取。如果count为负，则从右边开始截取。此函数为MaxCompute 2.0扩展函数。

- 参数说明
  - str：必填。STRING类型。待截取的字符串。
  - separator：必填。STRING类型的分隔符。
  - count：必填。INT类型。指定分隔符位置。

- 返回值说明

  返回STRING类型。如果任一输入参数值为NULL，返回NULL。

- 示例
  - 示例1：截取字符串 `https://help.aliyun.com` 。命令示例如下。

    ```
    --返回https://help.aliyun。
    select substring_index('https://help.aliyun.com', '.', 2);
    --返回aliyun.com。
    select substring_index('https://help.aliyun.com', '.', -2);
    ```

  - 示例2：任一输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select substring_index('https://help.aliyun.com', null, 2);
    ```

TRANSLATE

- 命令格式

```
string translate(string|varchar <str1>, string|varchar <str2>, string|varchar <str3>)
```

- 命令说明

  将str1出现在str2中的每个字符替换成str3中相对应的字符。无匹配则不替换。此函数为MaxCompute 2.0扩展函数。

- 返回值说明

  返回STRING类型。如果任一输入参数值为NULL，返回NULL。

- 示例
  - 示例1：将字符串 `ababab` 出现在 `abab` 中的每个字符替换成 `cd` 中的相应字符。命令示例如下。

    ```
    --返回cdcdcd。
    select translate('abababab','abab','cd');
    ```

- 示例2：将字符串 `ababab` 出现在 `abab` 中的每个字符替换成 `cdefg` 中的相应字符。命令示例如下。

```
--返回cdcdcd。
select translate('ababab','abab','cdefg');
```

- 示例3：任一输入参数为NULL。命令示例如下。

```
--返回NULL。
select translate('ababab','cd',null);
```

3.9.8. 复杂类型函数

您可以在MaxCompute SQL中使用复杂类型函数处理复杂数据类型，例如ARRAY、MAP、STRUCT、JSON。本文为您提供MaxCompute SQL支持的复杂类型函数的命令格式、参数说明及示例，指导您使用复杂类型函数完成开发。

MaxCompute SQL支持的复杂类型函数如下。

函数类别	函数	功能
ARRAY函数	ALL_MATCH	判断ARRAY数组中是否所有元素都满足指定条件。
	ANY_MATCH	判断ARRAY数组中是否存在满足指定条件的元素。
	ARRAY	使用给定的值构造ARRAY。
	ARRAY_CONTAINS	检测指定的ARRAY中是否包含指定的值。
	ARRAY_DISTINCT	去除ARRAY数组中的重复元素。
	ARRAY_EXCEPT	找出在ARRAY A中，但不在ARRAY B中的元素，并去掉重复的元素后，以ARRAY形式返回结果。
	ARRAY_INTERSECT	计算两个ARRAY数组的交集。
	ARRAY_JOIN	将ARRAY数组中的元素按照指定字符串进行拼接。
	ARRAY_MAX	计算ARRAY数组中的最大值。
	ARRAY_MIN	计算ARRAY数组中的最小值。
	ARRAY_POSITION	计算指定元素在ARRAY数组中第一次出现的位置。
	ARRAY_REDUCE	将ARRAY数组的元素进行聚合。
	ARRAY_REMOVE	在ARRAY数组中删除指定元素。
	ARRAY_REPEAT	返回将指定元素重复指定次数后的ARRAY数组。
	ARRAY_SORT	将ARRAY数组的元素进行排序。
	ARRAY_UNION	计算两个ARRAY数组的并集并去掉重复元素。
	ARRAYS_OVERLAP	判断两个ARRAY数组中是否包含相同元素。
	ARRAYS_ZIP	合并多个ARRAY数组。
	CONCAT	将ARRAY数组或字符串连接在一起。
	EXPLODE	将一行数据转为多行的UDTF。
	FILTER	将ARRAY数组中的元素进行过滤。
	INDEX	返回ARRAY数组指定位置的元素值。
	POSEXPLODE	将指定的ARRAY展开，每个Value一行，每行两列分别对应数组从0开始的下标和数组元素。
	SIZE	返回指定ARRAY中的元素数目。
	SLICE	对ARRAY数据切片，返回从指定位置开始、指定长度的数组。
	SORT_ARRAY	为指定的数组中的元素排序。
	TRANSFORM	将ARRAY数组中的元素进行转换。
	ZIP_WITH	将2个ARRAY数组按照位置进行元素级别的合并。
	EXPLODE	将一行数据转为多行的UDTF。

函数类别	函数	功能
MAP函数		
	INDEX	返回MAP类型参数中满足指定条件的Value。
	MAP	使用指定的Key-Value对建立MAP。
	MAP_CONCAT	返回多个MAP的并集。
	MAP_ENTRIES	将MAP中的Key、Value键值映射转换为STRUCT结构数组。
	MAP_FILTER	将MAP中的元素进行过滤。
	MAP_FROM_ARRAYS	通过给定的ARRAY数组构造MAP。
	MAP_FROM_ENTRIES	通过给定的结构体数组构造MAP。
	MAP_KEYS	将参数MAP中的所有Key作为数组返回。
	MAP_VALUES	将参数MAP中的所有Value作为数组返回。
	MAP_ZIP_WITH	对输入的两个MAP进行合并得到一个新MAP。
	SIZE	返回指定MAP中的K/V对数。
	TRANSFORM_KEYS	对MAP进行变换，保持Value不变，根据指定函数计算新的Key。
	TRANSFORM_VALUES	对MAP进行变换，保持Key不变，根据指定函数计算新的Value。
STRUCT函数	FIELD	获取STRUCT中的成员变量的取值。
	INLINE	将指定的STRUCT数组展开。每个数组元素对应一行，每行每个STRUCT元素对应一列。
	STRUCT	使用给定Value列表建立STRUCT。
	NAMED_STRUCT	使用给定的Name、Value列表建立STRUCT。
JSON函数	FROM_JSON	根据给定的JSON字符串和输出格式信息，返回ARRAY、MAP或STRUCT类型。
	GET_JSON_OBJECT	在一个标准JSON字符串中，按照指定方式抽取指定的字符串。
	JSON_TUPLE	在一个标准的JSON字符串中，按照输入的一组键抽取各个键指定的字符串。
	TO_JSON	将指定的复杂类型输出为JSON字符串。

ALL_MATCH

- 命令格式

```
boolean all_match(array<T> <a>, function<T, boolean> <predicate>)
```

- 命令说明

  判断ARRAY数组a中是否所有元素都满足predicate条件。

- 参数说明
  - a：必填。ARRAY数组。`array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - predicate：必填。用于对ARRAY数组a中的元素进行判断的函数（内建函数或自定义函数）或表达式。输入参数的数据类型必须与ARRAY数组a中元素的数据类型一致。

- 返回值说明

  返回BOOLEAN类型。返回规则如下：
  - 如果ARRAY数组a中所有的元素满足predicate条件或ARRAY数组为空，返回结果为True。
  - 如果ARRAY数组a中存在元素不满足predicate条件，返回结果为False。
  - 如果ARRAY数组a中存在元素为NULL，且其他元素都满足predicate条件，返回结果为NULL。

- 示例
  - 示例1：判断ARRAY数组 `array(4, 5, 6)` 的所有元素是否满足 `x-> x > 3` 条件（所有元素大于3）。命令示例如下。

```
--返回true。
select all_match(array(4, 5, 6), x -> x>3);
```

- 示例2：ARRAY数组为空。命令示例如下。

```
--返回true。
select all_match(array(), x -> x>3);
```

- 示例3：判断ARRAY数组 `array(1, 2, -10, 100, -30)` 的所有元素是否满足 `x-> x > 3` 条件。命令示例如下。

```
--返回false。
select all_match(array(1, 2, -10, 100, -30), x -> x>3);
```

- 示例4：判断存在NULL元素的ARRAY数组 `array(10, 100, 30, null)` 的所有元素是否满足 `x-> x > 3` 条件。命令示例如下。

```
--返回NULL。
select all_match(array(10, 100, 30, null), x -> x>3);
```

ANY_MATCH

- 命令格式

```
boolean any_match(array<T> <a>, function<T, boolean> <predicate>)
```

- 命令说明

  判断ARRAY数组a中是否存在元素满足predicate条件。

- 参数说明
  - a：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - predicate：必填。用于对ARRAY数组a中的元素进行判断的函数（内建函数或自定义函数）或表达式。输入参数的数据类型必须与ARRAY数组a中元素的数据类型一致。
- 返回值说明

  返回BOOLEAN类型。返回规则如下：
  - 如果ARRAY数组a中存在一个或多个元素满足predicate条件，返回结果为True。
  - 如果ARRAY数组a中没有元素满足predicate条件或ARRAY数组为空，返回结果为False。
  - 如果ARRAY数组a中存在元素为NULL，且其他元素都不满足predicate条件，返回结果为NULL。
- 示例
  - 示例1：判断ARRAY数组 `array(1, 2, -10, 100, -30)` 中是否有元素满足 `x-> x > 3` 条件。命令示例如下。

```
--返回true。
select any_match(array(1, 2, -10, 100, -30), x-> x > 3);
```

  - 示例2：ARRAY数组为空。命令示例如下。

```
--返回false。
select any_match(array(), x-> x > 3);
```

  - 示例3：判断ARRAY数组 `array(1, 2, -10, -20, -30)` 中是否有元素满足 `x-> x > 3` 条件。命令示例如下。

```
--返回false。
select any_match(array(1, 2, -10, -20, -30), x-> x > 3);
```

  - 示例4：判断存在NULL元素的ARRAY数组 `array(1, 2, null, -10)` 中是否有元素满足 `x-> x > 3` 条件。命令示例如下。

```
--返回NULL。
select any_match(array(1, 2, null, -10), x-> x > 3);
```

ARRAY

- 命令格式

```
array array(<value>,<value>[, ...])
```

- 命令说明

  使用指定的值构造ARRAY数组。

- 参数说明

  value：必填。可以为任意类型。所有value的数据类型必须一致。

- 返回值说明

  返回ARRAY类型。

- 示例

  例如表 `t_table` 的字段为 `c1 bigint, c2 string, c3 string, c4 bigint, c5 bigint` ，包含数据如下：

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
| 1000       | k11 | k21 | 86         | 15         |
| 1001       | k12 | k22 | 97         | 2          |
| 1002       | k13 | k23 | 99         | 1          |
+------------+----+----+------------+------------+
```

命令示例如下。

```
--根据c2、c4、c3、c5列的数据构造ARRAY数组。
select array(c2,c4,c3,c5) from t_table;
--返回结果如下。
+------+
| _c0  |
+------+
| [k11, 86, k21, 15] |
| [k12, 97, k22, 2] |
| [k13, 99, k23, 1] |
+------+
```

ARRAY_CONTAINS

- 命令格式

```
boolean array_contains(array<T> <a>, value <v>)
```

- 命令说明

  判断ARRAY数组a中是否存在元素v。

- 参数说明
  - a：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - v：必填。待判断的元素。必须与ARRAY数组a中元素的数据类型一致。

- 返回值说明

  返回BOOLEAN类型。

- 示例

  例如表`t_table_array`的字段为`c1 bigint, t_array array<string>`，包含数据如下：

```
+------------+---------+
| c1         | t_array |
+------------+---------+
| 1000       | [k11, 86, k21, 15] |
| 1001       | [k12, 97, k22, 2] |
| 1002       | [k13, 99, k23, 1] |
+------------+---------+
```

命令示例如下。

```
--检测t_array列是否包含1。
select c1, array_contains(t_array,'1') from t_table_array;
--返回结果如下。
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | false |
| 1001       | false |
| 1002       | true |
+------------+------+
```

ARRAY_DISTINCT

- 命令格式

```
array<T> array_distinct(array<T> <a>)
```

- 命令说明

  去除ARRAY数组a中的重复元素。

- 参数说明

  a：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。

- 返回值说明

返回ARRAY类型。返回规则如下：

- 新ARRAY数组无重复元素且元素顺序与a中的元素顺序保持一致。
- ARRAY数组a中存在元素为NULL时，NULL值会参与运算。
- 输入数组为空时，返回空数组。

- 示例
  - 示例1：去除ARRAY数组 `array(10, 20, 30, 30, 20, 10)` 中的重复元素。命令示例如下。

    ```
    --返回[10,20,30]。
    select array_distinct(array(10, 20, 30, 30, 20, 10));
    ```

  - 示例2：去除ARRAY数组 `array(10, 20, 20, null, null, 30, 20, null)` 中的重复元素。命令示例如下。

    ```
    --返回[10,20,null,30]。
    select array_distinct(array(10, 20, 20, null, null, 30, 20, null));
    ```

  - 示例3：ARRAY数组为空。命令示例如下。

    ```
    --返回[]。
    select array_distinct(array());
    ```

ARRAY_EXCEPT

- 命令格式

```
array<T> array_except(array<T> <a>, array<T> <b>)
```

- 命令说明

  找出在ARRAY数组a中，但不在ARRAY数组b中的元素，并去掉重复的元素后，返回新的ARRAY数组。

- 参数说明

  a、b：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。a和b的数据类型必须保持一致。

- 返回值说明

  返回ARRAY类型。返回规则如下：

  - 新ARRAY数组无重复元素且元素顺序与a中的元素顺序保持一致。
  - ARRAY数组中存在元素为NULL时，NULL值会参与运算。
  - 任一输入数组为空时，返回对非空数组去重后的新ARRAY数组。
  - 输入数组全部为空时，返回空数组。

- 示例
  - 示例1：找出在ARRAY数组 `array(1, 1, 3, 3, 5, 5)` 中，不在ARRAY数组 `array(1, 1, 2, 2, 3, 3)` 中的元素并去重。命令示例如下。

    ```
    --返回[5]。
    select array_except(array(1, 1, 3, 3, 5, 5), array(1, 1, 2, 2, 3, 3));
    ```

  - 示例2：找出在ARRAY数组 `array(1, 1, 3, 3, 5, 5, null, null)` 中，不在ARRAY数组 `array(1, 1, 2, 2, 3, 3)` 中的元素并去重。命令示例如下。

    ```
    --返回[5,null]。
    select array_except(array(1, 1, 3, 3, 5, 5, null, null), array(1, 1, 2, 2, 3, 3));
    ```

  - 示例3：任一输入ARRAY数组为空。命令示例如下。

    ```
    --返回[2,1]。
    select array_except(array(2, 1, 1, 2), cast(array() as array<int>));
    ```

  - 示例4：输入ARRAY数组全部为空。命令示例如下。

    ```
    --返回[]。
    select array_except(cast(array() as array<int>), cast(array() as array<int>));
    ```

ARRAY_INTERSECT

- 命令格式

```
array<T> array_intersect(array<T> <a>, array<T> <b>)
```

- 命令说明

  计算ARRAY数组a和b的交集，并去掉重复元素。

- 参数说明

a、b：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。a和b的数据类型必须保持一致。

- 返回值说明

  返回ARRAY类型。返回规则如下：

  - ARRAY数组中存在元素为NULL时，NULL值会参与运算。
  - 新ARRAY数组无重复元素且元素顺序与a中的元素顺序保持一致。
  - 如果ARRAY数组a或b为NULL，返回NULL。

- 示例
  - 示例1：计算ARRAY数组`array(1, 2, 3)`和`array(1, 3, 5)`的交集，并去掉重复元素。命令示例如下。

    ```
    --返回[1,3]。
    select array_intersect(array(1, 2, 3), array(1, 3, 5));
    ```

  - 示例2：计算ARRAY数组`array(10, 20, 20, 30, 30, null, null)`和`array(30, 30, 20, 20, 40, null, null)`的交集，并去掉重复元素。命令示例如下。

    ```
    --返回[20,30,null]。
    select array_intersect(array(10, 20, 20, 30, 30, null, null), array(30, 30, 20, 20, 40, null, null));
    ```

ARRAY_JOIN

- 命令格式

  ```
  array_join(array<T> <a>, <delimiter>[, <nullreplacement>])
  ```

- 命令说明

  将ARRAY数组a中的元素使用delimiter拼接为字符串。当数组中元素为NULL时，用nullreplacement替代，没有设置nullreplacement时，会忽略NULL元素。

- 参数说明
  - a：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型。

    > **说明** 当ARRAY数组中的元素非STRING类型时，MaxCompute会将非STRING类型元素转换为STRING类型。

  - delimiter：必填。STRING类型。连接ARRAY数组a中元素的字符串。
  - nullreplacement：可选。替代NULL元素的字符串。

- 返回值说明

  返回STRING类型。

- 示例

  ```
  --返回10,20,20,30。
  select array_join(array(10, 20, 20, null, null, 30), ",");
  --返回10##20##20##null##null##30。
  select array_join(array(10, 20, 20, null, null, 30), "##", "null");
  ```

ARRAY_MAX

- 命令格式

  ```
  T array_max(array<T> <a>)
  ```

- 命令说明

  计算ARRAY数组a中的最大元素。

- 参数说明

  a：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型。

  数组中的元素可以为如下类型：

  - TINYINT、SMALLINT、INT、BIGINT
  - FLOAT、DOUBLE
  - BOOLEAN
  - DECIMAL、DECIMALVAL
  - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
  - STRING、BINARY、VARCHAR、CHAR
  - ARRAY、STRUCT、MAP

- 返回值说明

返回ARRAY数组a中的最大元素。返回规则如下：

- 如果ARRAY数组a为NULL，返回NULL。
- 如果ARRAY数组a中存在元素为NULL，NULL值不参与运算。

- 示例

```
--返回20。
select array_max(array(1, 20, null, 3));
```

ARRAY_MIN

- 命令格式

```
T array_min(array<T> <a>)
```

- 命令说明

  计算ARRAY数组a中的最小元素。

- 参数说明

  a：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型。

  数组中的元素可以为如下类型：

  - TINYINT、SMALLINT、INT、BIGINT
  - FLOAT、DOUBLE
  - BOOLEAN
  - DECIMAL、DECIMALVAL
  - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
  - STRING、BINARY、VARCHAR、CHAR
  - ARRAY、STRUCT、MAP

- 返回值说明

  返回ARRAY数组a中的最小元素。返回规则如下：

  - 如果ARRAY数组a为NULL，返回NULL。
  - 如果ARRAY数组a中存在元素为NULL时，NULL值不参与运算。

- 示例

```
--返回1。
select array_min(array(1, 20, null, 3));
```

ARRAY_POSITION

- 命令格式

```
bigint array_position(array<T> <a>, T <element>)
```

- 命令说明

  计算元素element在ARRAY数组a中第一次出现的位置。ARRAY数组元素位置编号自左往右，从1开始计数。

- 参数说明
  - a：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型。支持的数据类型如下：
    - TINYINT、SMALLINT、INT、BIGINT
    - FLOAT、DOUBLE
    - BOOLEAN
    - DECIMAL、DECIMALVAL
    - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
    - STRING、BINARY、VARCHAR、CHAR
    - ARRAY、STRUCT、MAP
  - element：必填。待查询的元素，数据类型必须与a中元素的数据类型相同。

- 返回值说明

  返回BIGINT类型。返回规则如下：

  - 如果ARRAY数组a或element为NULL，返回NULL。
  - 未找到元素时返回0。

- 示例

- 示例1：计算元素 `1` 第一次出现在ARRAY数组 `array(3, 2, 1)` 中的位置。命令示例如下。

```
--返回3。
select array_position(array(3, 2, 1), 1);
```

- 示例2：element为NULL。命令示例如下。

```
--返回NULL。
select array_position(array(3, 1, null), null);
```

ARRAY_REMOVE

- 命令格式

```
array<T> array_remove(array<T> <a>, T <element>)
```

- 命令说明

  在ARRAY数组a中删除与element相等的元素。

- 参数说明
  - a：必填。ARRAY数组。`array<T>` 中的 `T` 指代ARRAY数组元素的数据类型。支持的数据类型如下：
    - TINYINT、SMALLINT、INT、BIGINT
    - FLOAT、DOUBLE
    - BOOLEAN
    - DECIMAL、DECIMALVAL
    - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
    - STRING、BINARY、VARCHAR、CHAR
    - ARRAY、STRUCT、MAP
  - element：必填。待删除的元素，数据类型必须与a中元素的数据类型相同。
- 返回值说明

  返回ARRAY类型。返回规则如下：
  - 如果ARRAY数组a中存在元素为NULL时，NULL值不参与运算。
  - 如果ARRAY数组a或element为NULL，返回NULL。
  - ARRAY数组a中不存在element时返回原ARRAY数组a。
- 示例
  - 示例1：删除ARRAY数组 `array(3, 2, 1)` 中等于 `1` 的元素。命令示例如下。

```
--返回[3,2]。
select array_remove(array(3, 2, 1), 1);
```

  - 示例2：element为NULL。命令示例如下。

```
--返回NULL。
select array_remove(array(3, 1, null), null);
```

  - 示例3：删除ARRAY数组 `array(3, 1, null)` 中等于 `2` 的元素。命令示例如下。

```
--返回[3,1,null]。
select array_remove(array(3, 1, null), 2);
```

ARRAY_REDUCE

- 命令格式

```
R array_reduce(array<T> <a>, buf <init>, function<buf, T, buf> <merge>, function<buf, R> <final>)
```

- 命令说明

  对ARRAY数组a中的元素进行聚合。

- 参数说明
  - a：必填。ARRAY数组。`array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - init：必填。用于聚合的中间结果的初始值。
  - merge：必填。将ARRAY数组a中的每一个元素与中间结果进行运算的函数（内建函数或自定义函数）或表达式。它的两个输入参数为ARRAY数组a的元素和init。
  - final：必填。将中间结果转换为最终结果的函数（内建函数或自定义函数）或表达式。它的输入参数为merge运行结果，R指代输出结果的数据类型。
- 返回值说明

返回结果类型与final函数定义的输出结果类型一致。

- 示例

```
--返回6。
select array_reduce(array(1, 2, 3), 0, (buf, e)->buf + e, buf->buf);
--返回2.5。
select array_reduce(array(1, 2, 3, 4), named_struct('sum', 0, 'count', 0), (buf, e)->named_struct('sum', buf.sum + e, 'count',
buf.count + 1), buf -> buf.sum / buf.count);
```

ARRAY_REPEAT

- 命令格式

```
array<T> array_repeat(T <element>, int <count>)
```

- 命令说明

返回将元素t重复count次后新生成的ARRAY数组。

- 参数说明
  - t：必填。待重复的元素。支持的类型如下：
    - TINYINT、SMALLINT、INT、BIGINT
    - FLOAT、DOUBLE
    - BOOLEAN
    - DECIMAL、DECIMALVAL
    - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
    - STRING、BINARY、VARCHAR、CHAR
    - ARRAY、STRUCT、MAP
  - count：必填。重复的次数，INT类型。必须大于等于0。
- 返回值说明

返回ARRAY类型。返回规则如下：

  - 如果count为NULL，返回NULL。
  - 如果count小于0，返回空数组。
- 示例
  - 示例1：将 `123` 重复 `2` 次，生成新的ARRAY数组。命令示例如下。

```
--返回[123, 123]。
select array_repeat('123', 2);
```

  - 示例2：count为NULL。命令示例如下。

```
--返回NULL。
select array_repeat('123', null);
```

  - 示例3：count小于0。命令示例如下。

```
--返回[]。
select array_repeat('123', -1);
```

ARRAY_SORT

- 命令格式

```
array<T> array_sort(array<T> <a>, function<T, T, bigint> <comparator>)
```

- 命令说明

将ARRAY数组a中的元素根据comparator进行排序。

- 参数说明
  - a：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - comparator：必填。用于比较ARRAY数组中2个元素大小的函数（内建函数或自定义函数）或表达式。

    `comparator(a, b)` 的处理逻辑为：当a等于b时，返回0。当a小于b时，返回负整数。当a大于b时，返回正整数。如果 `comparator(a, b)` 返回NULL，则返回报错。
- 返回值说明

返回ARRAY类型。

- 示例

```
--返回[{"a":1,"b":10},{"a":2,"b":12},{"a":3,"b":11}]。
select array_sort(a, (a,b)->case when a.a> b.a then 1L when a.a=b.a then 0L else -1L end)
from values (
  array(named_struct('a', 1, 'b', 10),
        named_struct('a', 3, 'b', 11),
        named_struct('a', 2, 'b', 12)))
  as t(a);
```

ARRAY_UNION

- 命令格式

```
array<T> array_union(array<T> <a>,  array<T> <b>)
```

- 命令说明

  计算ARRAY数组a和b的并集，并去掉重复元素。

- 参数说明

  a、b：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。a和b中元素的数据类型必须一致。

  数组中的元素可以为如下类型：

  - TINYINT、SMALLINT、INT、BIGINT
  - FLOAT、DOUBLE
  - BOOLEAN
  - DECIMAL、DECIMALVAL
  - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
  - STRING、BINARY、VARCHAR、CHAR
  - ARRAY、STRUCT、MAP

- 返回值说明

  返回ARRAY类型。如果a或b为NULL，返回NULL。

- 示例

  - 示例1：计算ARRAY数组`array(1, 2, 3)`和`array(1, 3, 5)`的并集，并去掉重复元素。命令示例如下。

    ```
    --返回[1,2,3,5]。
    select array_union(array(1, 2, 3), array(1, 3, 5));
    ```

  - 示例2：任一ARRAY数组为NULL。命令示例如下。

    ```
    --返回NULL。
    select array_union(array(1, 2, 3), null);
    ```

ARRAYS_OVERLAP

- 命令格式

```
boolean arrays_overlap(array<T> <a>,  array<T> <b>)
```

- 命令说明

  判断ARRAY数组a和b是否存在相同元素。

- 参数说明

  a、b：必填。ARRAY数组。`array<T>`中的`T`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。a和b中元素的数据类型必须一致。

  数组中的元素可以为如下类型：

  - TINYINT、SMALLINT、INT、BIGINT
  - FLOAT、DOUBLE
  - BOOLEAN
  - DECIMAL、DECIMALVAL
  - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
  - STRING、BINARY、VARCHAR、CHAR
  - ARRAY、STRUCT、MAP

- 返回值说明

  返回BOOLEAN类型。返回规则如下：

  - 如果ARRAY数组a中至少包含ARRAY数组b中的一个非NULL元素，返回结果为True。

- 如果ARRAY数组a和b中没有公共元素、都非空，且其中任意一个数组中包含NULL元素，返回结果为NULL。
- 如果ARRAY数组a和b中没有公共元素、都非空，且其中任意一个数组中都不包含NULL元素，返回结果为False。

- 示例
  - 示例1：判断ARRAY数组 `array(1, 2, 3)` 和 `array(3, 4, 5)` 中是否存在相同元素。命令示例如下。

```
--返回true。
select arrays_overlap(array(1, 2, 3), array(3, 4, 5));
```

  - 示例2：判断ARRAY数组 `array(1, 2, 3)` 和 `array(6, 4, 5)` 中是否存在相同元素。命令示例如下。

```
--返回false。
select arrays_overlap(array(1, 2, 3), array(6, 4, 5));
```

  - 示例3：任一ARRAY数组中存在NULL元素。命令示例如下。

```
--返回NULL。
select arrays_overlap(array(1, 2, 3), array(5, 4, null));
```

ARRAYS_ZIP

- 命令格式

```
array<struct<T, U, ...>> arrays_zip(array<T> <a>, array<U> <b>[, ...])
```

- 命令说明

  合并多个给定数组并返回一个结构数组，其中第N个结构包含输入数组的所有第N个值。

- 参数说明

  a、b：必填。ARRAY数组。 `array<T>` 及 `array<U>` 中的 `T` 和 `U` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。

  数组中的元素可以为如下类型：

  - TINYINT、SMALLINT、INT、BIGINT
  - FLOAT、DOUBLE
  - BOOLEAN
  - DECIMAL、DECIMALVAL
  - DATE、DATETIME、TIMESTAMP、IntervalDayTime、IntervalYearMonth
  - STRING、BINARY、VARCHAR、CHAR
  - ARRAY、STRUCT、MAP

- 返回值说明

  返回ARRAY类型。返回规则如下：

  - 生成的结构数组中第N个结构包含输入数组的所有第N个值，不足N的元素以NULL填充。
  - 如果输入ARRAY数组中任意一个为NULL，返回结果为NULL。

- 示例
  - 示例1：通过ARRAY数组 `array(1, 2, 3)` 和 `array(2, 3, 4)` 构造结构数组。命令示例如下。

```
--返回[{0:1, 1:2}, {0:2, 1:3}, {0:3, 1:4}]。
select arrays_zip(array(1, 2, 3), array(2, 3, 4));
```

  - 示例2：通过ARRAY数组 `array(1, 2, 3)` 和 `array(4, 5)` 构造结构数组。命令示例如下。

```
--返回[{0:1, 1:4}, {0:2, 1:5}, {0:3, 1:NULL}]。
select arrays_zip(array(1, 2, 3), array(4, 5));
```

CONCAT

- 命令格式

```
array<T> concat(array<T> <a>, array<T> <b>[,...])
string concat(string <str1>, string <str2>[,...])
```

- 命令说明
  - 输入为ARRAY数组：将多个ARRAY数组中的所有元素连接在一起，生成一个新的ARRAY数组。
  - 输入为字符串：将多个字符串连接在一起，生成一个新的字符串。

- 参数说明
  - a、b：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。a和b中元素的数据类型必须一致。数组中的元素为NULL值时会参与运算。

- str1、str2：必填。STRING类型。如果输入参数为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。

- 返回值说明
  - 返回ARRAY类型。如果任一输入ARRAY数组为NULL，返回结果为NULL。
  - 返回STRING类型。如果没有参数或任一参数为NULL，返回结果为NULL。
- 示例
  - 示例1：连接ARRAY数组 `array(10, 20)` 和 `array(20, -20)` 。命令示例如下。

    ```
    --返回[10, 20, 20, -20]。
    select concat(array(10, 20), array(20, -20));
    ```

  - 示例2：ARRAY数组元素包含NULL。命令示例如下。

    ```
    --返回[10, NULL, 20, -20]。
    select concat(array(10, null), array(20, -20));
    ```

  - 示例3：任一ARRAY数组为NULL。命令示例如下。

    ```
    --返回NULL。
    select concat(array(10, 20), null);
    ```

  - 示例4：连接字符串 `aabc` 和 `abcde` 。命令示例如下。

    ```
    --返回aabcabcde。
    select concat('aabc','abcde');
    ```

  - 示例5：输入为空。命令示例如下。

    ```
    --返回NULL。
    select concat();
    ```

  - 示例6：任一字符串输入为NULL。命令示例如下。

    ```
    --返回NULL。
    select concat('aabc', 'abcde', null);
    ```

EXPLODE

- 使用限制
  - 在一个 `select` 中只能出现一个 `explode` 函数，不可以出现表的其他列。
  - 不可以与 `group by` 、 `cluster by` 、 `distribute by` 、 `sort by` 一起使用。
- 命令格式

  ```
  explode (<var>)
  ```

- 命令说明

  将一行数据转为多行的UDTF。
  - 如果参数是 `array<T>` 类型，则将列中存储的ARRAY转为多行。
  - 如果参数是 `map<K, V>` 类型，则将列中存储的MAP的每个Key-Value对转换为包含两列的行，其中一列存储Key，令一列存储Value。
- 参数说明

  var：必填。 `array<T>` 类型或 `map<K, V>` 类型。
- 返回值说明

  返回转换后的行。
- 示例

  例如表 `t_table_map` 的字段为 `c1 bigint, t_map map<string,bigint>` ，包含数据如下：

  ```
    +------------+--------+
  | c1         | t_map |
  +------------+--------+
  | 1000       | {k11:86, k21:15} |
  | 1001       | {k12:97, k22:2} |
  | 1002       | {k13:99, k23:1} |
  +------------+--------+
  ```

  命令示例如下。

```
select explode(t_map) from t_table_map;
--返回结果如下。
+-----+------------+
| key | value      |
+-----+------------+
| k11 | 86         |
| k21 | 15         |
| k12 | 97         |
| k22 | 2          |
| k13 | 99         |
| k23 | 1          |
+-----+------------+
```

FILTER

- 命令格式

```
array<T> filter(array<T> <a>, function<T,boolean> <func>)
```

- 命令说明

  将ARRAY数组a中的元素利用func进行过滤，返回一个新的ARRAY数组。

- 参数说明
  - a：必填。ARRAY数组。`array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - func：必填。用于对a中元素进行过滤的函数（内置函数或自定义函数）或表达式，其输入参数类型必须与a中元素的数据类型一致，其输出结果数据类型为BOOLEAN。
- 返回值说明

  返回ARRAY类型。

- 示例

```
--返回[2, 3]。
select filter(array(1, 2, 3), x -> x > 1);
```

INDEX

- 命令格式

```
index(<var1>[<var2>])
```

- 命令说明
  - 如果var1是 `array<T>` 类型，获取var1的第var2个元素。ARRAY数组元素编号自左往右，从0开始计数。
  - 如果var1是 `map<K, V>` 类型，获取var1中Key为var2的Value。

> **说明** 使用该函数时需要去掉 `index` ，请直接执行 `<var1>[<var2>]` ，否则会返回报错。

- 参数说明
  - var1：必填。`array<T>` 类型或 `map<K, V>` 类型。`array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。`map<K, V>` 中的 `K` 、 `V` 指代MAP对象的Key、Value。
  - var2：必填。
    - 如果var1是 `array<T>` 类型，则var2为BIGINT类型且大于等于0。
    - 如果var1是 `map<K, V>` 类型，则var2与K的类型保持一致。
- 返回值说明
  - 如果var1是 `array<T>` 类型，函数返回T类型。返回规则如下：
    - 如果var2超出var1的元素数目范围，返回结果为NULL。
    - 如果var1为NULL，返回结果为NULL。
  - 如果var1是 `map<K, V>` 类型，函数返回V类型。返回规则如下：
    - 如果 `map<K, V>` 中不存在Key为var2的情况，返回结果为NULL。
    - 如果var1为NULL，返回结果为NULL。
- 示例
  - 示例1：var1为 `array<T>` 类型。命令示例如下。

```
--返回c。
select array('a','b','c')[2];
```

- 示例2：var1为 `map<K, V>` 类型，命令示例如下。

```
--返回1。
select str_to_map("test1=1,test2=2")["test1"];
```

POSEXPLODE

- 命令格式

```
posexplode(array<T> <a>)
```

- 命令说明

  将ARRAY数组a展开，每个Value一行，每行两列分别对应数组从0开始的下标和数组元素。

- 参数说明

  a：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。

- 返回值说明

  返回表。

- 示例

```
select posexplode(array('a','c','f','b'));
--返回结果如下。
+------------+------------+
| pos        | val        |
+------------+------------+
| 0          | a          |
| 1          | c          |
| 2          | f          |
| 3          | b          |
+------------+------------+
```

SIZE

- 命令格式

```
int size(array<T> <a>)
int size(map<K, V> <b> )
```

- 命令说明
  - 输入为ARRAY数组：计算ARRAY数组a中的元素数目。
  - 输入为MAP对象：计算MAP对象b中的Key-Value对数。
- 参数说明
  - a：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - b：必填。MAP对象。 `map<K, V>` 中的 `K` 、 `V` 指代MAP对象的Key、Value。
- 返回值说明

  返回INT类型。

- 示例
  - 示例1：计算ARRAY数组 `array('a','b')` 中的元素数目。命令示例如下。

```
--返回2。
select size(array('a','b'));
```

  - 示例2：计算MAP对象 `map('a',123,'b',456)` 中的Key-Value对数。

```
--返回2。
select size(map('a',123,'b',456));
```

SLICE

- 命令格式

```
array<T> slice(array<T> <a>, <start>, <length>)
```

- 命令说明

  对ARRAY数组切片，截取从start位置开始长度为length的元素组成新的ARRAY数组。

- 参数说明
  - a：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - start：必填。切片起点，从1开始，表示从数组的首个元素开始向右切片。start可以为负数，表示从数组的末尾元素开始向右切片。

- length：必填。切片长度，必须大于或等于0。切片长度如果大于ARRAY数组长度时，会返回从start位置开始到末尾元素组成的ARRAY数组。

- 返回值说明

  返回ARRAY类型。

- 示例
  - 示例1：截取ARRAY数组 `array(10, 20, 20, null, null, 30)` 从第 `1` 个位置开始，切片长度为 `3` 的元素。命令示例如下。

    ```
    --返回[10, 20, 20]。
    select slice(array(10, 20, 20, null, null, 30), 1, 3);
    ```

  - 示例2：截取ARRAY数组 `array(10, 20, 20, null, null, 30)` 从第 `-2` 个位置开始，切片长度为 `2` 的元素。命令示例如下。

    ```
    --返回[NULL, 30]。
    select slice(array(10, 20, 20, null, null, 30), -2, 2);
    ```

  - 示例3：截取ARRAY数组 `array(10, 20, 20, null, null, 30)` 从第 `3` 个位置开始，切片长度为 `10` 的元素。命令示例如下。

    ```
    --返回[20, NULL, NULL, 30]。
    select slice(array(10, 20, 20, null, null, 30), 3, 10);
    ```

  - 示例4：截取ARRAY数组 `array(10, 20, 20, null, null, 30)` 从第 `3` 个位置开始，切片长度为 `0` 的元素。命令示例如下。

    ```
    --返回[]。
    select slice(array(10, 20, 20, null, null, 30), 3, 0);
    ```

SORT_ARRAY

- 命令格式

  ```
  array<T> sort_array(array<T> <a>[, <isasc>])
  ```

- 命令说明

  对ARRAY数组中的元素进行排序。

- 参数说明
  - a：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - isasc：可选。用于设置排序规则。取值为True（升序）或False（降序）。默认为升序。

- 返回值说明

  返回ARRAY类型。NULL值为最小值。

- 示例
  - 示例1：例如表 `t_array` 的字段为 `c1 array<string>,c2 array<int> ,c3 array<string>` ，包含数据如下：

    ```
    +------------+---------+--------------+
    | c1         | c2      | c3           |
    +------------+---------+--------------+
    | [a, c, f, b]  | [4, 5, 7, 2, 5, 8]  |  [你, 我, 他] |
    +------------+---------+--------------+
    ```

    对表的每列数据进行排序。命令示例如下。

    ```
    --返回[a, b, c, f] [2, 4, 5, 5, 7, 8] [他, 你, 我]。
    select sort_array(c1),sort_array(c2),sort_array(c3) from t_array;
    ```

  - 示例2：对ARRAY数组 `array(10, 20, 40, 30, 30, null, 50)` 进行降序排序。命令示例如下。

    ```
    --返回[50, 40, 30, 30, 20, 10, NULL]。
    select sort_array(array(10, 20, 40, 30, 30, null, 50), false);
    ```

TRANSFORM

- 命令格式

  ```
  array<R> transform(array<T> <a>, function<T, R> <func>)
  ```

- 命令说明

  将ARRAY数组a的元素利用func进行转换，返回一个新的ARRAY数组。

- 参数说明
  - a：必填。ARRAY数组。 `array<T>` 中的 `T` 指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - func：必填。用于对a中元素进行转换的函数（内建函数或自定义函数）或表达式，其输入类型应与a中的元素类型一致。 `R` 指代输出结果的数据类型。

- 返回值说明

返回ARRAY类型。

- 示例

```
--返回[2, 3, 4]。
select transform(array(1, 2, 3), x -> x + 1);
```

ZIP_WITH

- 命令格式

```
array<R> array_sort(array<T> <a>, array<S> <b>, function<T, S, R> <combiner>)
```

- 命令说明

  将ARRAY数组a和b的元素按照位置，使用combiner进行元素级别的合并，返回一个新的ARRAY数组。

- 参数说明
  - a、b：必填。ARRAY数组。`array<T>`、`array<S>`中的`T`、`S`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - combiner：必填。用于合并ARRAY数组a、b中元素的函数（内置函数或自定义函数）或表达式。它的两个输入参数类型分别与ARRAY数组a、b中元素的数据类型一致。

- 返回值说明

  返回ARRAY类型。返回规则如下：
  - 新生成的ARRAY数组中元素位置与a、b中相应元素的位置相同。
  - 如果ARRAY数组a和b的长度不一致，会将长度较短的ARRAY数组使用NULL值进行填充，然后进行合并。

- 示例

```
--返回[2, 4, 6, NULL]。
select zip_with(array(1,2,3), array(1,2,3,4), (x,y) -> x + y);
```

MAP

- 命令格式

```
map(K, V) map(K <key1>, V <value1>, K <key2>, V <value2>[, ...])
```

- 命令说明

  使用给定的Key-Value对生成MAP。

- 参数说明
  - key：必填。所有key类型一致（包括隐式转换后类型一致），必须是基本类型。
  - value：必填。所有value类型一致（包括隐式转换后类型一致），可为任意类型。

- 返回值说明

  返回MAP类型。

  > **说明** 您可以在Session级别通过`odps.sql.map.key.dedup.policy`参数设置出现重复Key时的处理方式。取值范围如下：
  > - exception：如果出现重复的Key，返回报错。
  > - last_win：如果出现重复的Key，后边的值将覆盖前边的值。
  >
  > 不设置时，该参数默认值为last_win。

- 示例

- 示例1：无重复Key。例如表 `t_table` 的字段为 `c1 bigint, c2 string, c3 string, c4 bigint, c5 bigint` ，包含数据如下。

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
| 1000       | k11 | k21 | 86         | 15         |
| 1001       | k12 | k22 | 97         | 2          |
| 1002       | k13 | k23 | 99         | 1          |
+------------+----+----+------------+------------+
```

命令示例如下。

```
--将c2、c4，c3、c5组成MAP。
select map(c2,c4,c3,c5) from t_table;
--返回结果如下。
+------+
| _c0  |
+------+
| {k11:86, k21:15} |
| {k12:97, k22:2} |
| {k13:99, k23:1} |
+------+
```

- 示例2。有重复Key。例如表 `t_table` 的字段为 `c1 bigint, c2 string, c3 string, c4 bigint, c5 bigint` ，包含数据如下。

```
1000,'k11','k11',86,15
1001,'k12','k22',97,2
1002,'k13','k23',99,1
1003,'k13','k24',100,1
1004,'k12','k25',95,1
```

命令示例如下。

```
--将c2、c4，c3、c5组成MAP。
select map(c2,c4,c3,c5) from t_table;
--返回结果如下。
+------+
| _c0  |
+------+
| {'k11':15} |
| {'k12':97, 'k22':2} |
| {'k13':99, 'k23':1} |
| {'k13':100, 'k24':1} |
| {'k12':95, 'k25':1} |
+------+
```

MAP_CONCAT

- 命令格式

```
map<K, V> map_concat([string <mapDupKeyPolicy>,] map<K, V> <a>, map<K, V> <b>[,...])
```

- 命令说明

  计算多个MAP对象的并集。

- 参数说明
  - mapDupKeyPolicy：可选。STRING类型。指定出现重复Key时的处理方式。取值范围如下：
    - exception：如果出现重复的Key，返回报错。
    - last_win：如果出现重复的Key，后边的值将覆盖前边的值。

    该参数也可以在Session级别通过 `odps.sql.map.key.dedup.policy` 参数进行设置，例如 `set odps.sql.map.key.dedup.policy=exception;` ，不设置时该参数默认值为last_win。

    > **说明** MaxCompute的行为实现优先以函数中mapDupKeyPolicy的取值为准，当函数未配置mapDupKeyPolicy时，以 `odps.sql.map.key.dedup.policy` 参数的取值为准。

  - a、b：必填。MAP对象。多个MAP对象的参数数据类型必须一致。 `map<K, V>` 中的 `K` 、 `V` 指代MAP对象的Key、Value。

- 返回值说明

  返回MAP类型。返回规则如下：
  - 某个MAP对象为NULL或某个MAP对象的Key为NULL时，返回报错。
  - 多个MAP对象的数据类型不一致时，返回报错。

- 示例

```
--返回{1:a, 2:b, 3:c}。
select map_concat(map(1, 'a', 2, 'b'), map(3, 'c'));
--返回{1:a, 2:d, 3:c}。
select map_concat('last_win', map(1, 'a', 2, 'b'), map(3, 'c'), map(2, 'd'));
```

MAP_ENTRIES

- 命令格式

```
array<struct<K, V>> map_entries(map<K, V> <a>):
```

- 命令说明

  将MAP对象a的K、Value映射转换为STRUCT结构数组。

- 参数说明

  a：必填。MAP对象。`map<K, V>`中的`K`、`V`指代MAP对象的Key、Value。

- 返回值说明

  返回STRUCT结构数组。如果输入为NULL，返回结果为NULL。

- 示例

```
--返回[{key:1, value:a}, {key:2, value:b}]。
select map_entries(map(1,  'a',  2,  'b'));
```

MAP_FILTER

- 命令格式

```
map<K, V> map_filter(map<K, V> <input>, function <K, V, boolean> <predicate>)
```

- 命令说明

  将MAP对象input的元素进行过滤，只保留满足predicate条件的元素。

- 参数说明
  - input：必填。MAP类型。`map<K, V>`中的`K`、`V`指代MAP对象的Key、Value。
  - predicate：必填。用于对输入MAP对象中的元素进行过滤的函数（内建函数或自定义函数）或表达式。它的两个输入参数，分别对应input中的Key和Value，输出结果为BOOLEAN类型。

- 返回值说明

  返回MAP类型。

- 示例

```
--返回{-30:100, 20:50}。
select map_filter(map(10, -20, 20, 50, -30, 100, 21, null), (k, v) -> (k+v) > 10);
```

MAP_FROM_ARRAYS

- 命令格式

```
map<K, V> map_from_arrays([string <mapDupKeyPolicy>,] array<K> <a>, array<V> <b>))
```

- 命令说明

  将ARRAY数组a和b组合成一个MAP对象。

- 参数说明
  - mapDupKeyPolicy：可选。STRING类型。指定出现重复Key时的处理方式。取值范围如下：
    - exception：如果出现重复的Key，返回报错。
    - last_win：如果出现重复的Key，后边的值将覆盖前边的值。

    该参数也可以在Session级别通过`odps.sql.map.key.dedup.policy`参数进行设置，例如`set odps.sql.map.key.dedup.policy=exception;`，不设置时该参数默认值为last_win。

    > **说明** MaxCompute的行为实现优先以函数中mapDupKeyPolicy的取值为准，当函数未配置mapDupKeyPolicy时，以`odps.sql.map.key.dedup.policy`参数的取值为准。

  - a：必填。ARRAY数组。对应生成MAP的Key值。`array<K>`中的`K`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。
  - b：必填。ARRAY数组。对应生成MAP的Value值。`array<V>`中的`V`指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。

- 返回值说明

返回MAP类型。返回规则如下：

- 如果a或b为NULL，返回结果为NULL。
- 如果a中元素包含NULL值或两个数组长度不相等，会返回报错。

- 示例

```
--返回{1:2, 3:4}。
select map_from_arrays(array(1.0, 3.0), array('2', '4'));
--返回{1:2, 3:6}。
select map_from_arrays('last_win', array(1.0, 3.0, 3), array('2', '4', '6'));
```

MAP_FROM_ENTRIES

- 命令格式

```
map<K, V> map_from_entries([string <mapDupKeyPolicy>,] array <struct<K, V> , struct<K, V>[,...]>)
```

- 命令说明

  将多个结构数组组合成一个MAP对象。

- 参数说明
  - mapDupKeyPolicy：可选。STRING类型。指定出现重复Key时的处理方式。取值范围如下：
    - exception：如果出现重复的Key，返回报错。
    - last_win：如果出现重复的Key，后边的值将覆盖前边的值。

    该参数也可以在Session级别通过 `odps.sql.map.key.dedup.policy` 参数进行设置，例如 `set odps.sql.map.key.dedup.policy=exception;` ，不设置时该参数默认值为last_win。

    > **说明** MaxCompute的行为实现优先以函数中mapDupKeyPolicy的取值为准，当函数未配置mapDupKeyPolicy时，以 `odps.sql.map.key.dedup.policy` 参数的取值为准。

  - 输入为STRUCT类型的数据。其中：K对应生成MAP的Key值，V对应生成MAP的Value值。 `struct<K, V>` 中的 `K` 、 `V` 指代STRUCT的Key、Value。

- 返回值说明

  返回MAP类型。返回规则如下：

  - 如果结构体数组为NULL，返回结果为NULL。
  - 如果结构体的Field数量不是2或K包含NULL值，会返回报错。

- 示例

```
--返回{1:a, 2:b}。
select map_from_entries(array(struct(1, 'a'), struct(2, 'b')));
--返回{1:a, 2:c}。
select map_from_entries(array(struct(1, 'a'), struct(2, 'b'), struct(2, 'c')));
```

MAP_KEYS

- 命令格式

```
array<K> map_keys(map<K, V> <a>)
```

- 命令说明

  将MAP对象a中的所有Key生成ARRAY数组。

- 参数说明

  a：必填。MAP对象。 `map<K, V>` 中的 `K` 、 `V` 指代MAP对象的Key、Value。

- 返回值说明

  返回ARRAY类型。输入MAP对象为NULL时，返回结果为NULL。

- 示例

  例如表 `t_table_map` 的字段为 `c1 bigint,t_map map<string,bigint>` ，包含数据如下：

```
+------------+-------+
| c1         | t_map |
+------------+-------+
| 1000       | {k11:86, k21:15} |
| 1001       | {k12:97, k22:2} |
| 1002       | {k13:99, k23:1} |
+------------+-------+
```

命令示例如下。

```
--将t_map中的Key作为数组返回。
select c1, map_keys(t_map) from t_table_map;
--返回结果如下。
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | [k11, k21] |
| 1001       | [k12, k22] |
| 1002       | [k13, k23] |
+------------+------+
```

MAP_VALUES

- 命令格式

```
array<V> map_values(map<K, V> <a>)
```

- 命令说明

  将MAP对象a中的所有Value生成ARRAY数组。

- 参数说明

  a：必填。MAP对象。`map<K, V>`中的`K`、`V`指代MAP对象的Key、Value。

- 返回值说明

  返回ARRAY类型。输入MAP对象为NULL时，返回结果为NULL。

- 示例

  例如表`t_table_map`的字段为`c1 bigint,t_map map<string,bigint>`，包含数据如下：

```
+------------+-------+
| c1         | t_map |
+------------+-------+
| 1000       | {k11:86, k21:15} |
| 1001       | {k12:97, k22:2} |
| 1002       | {k13:99, k23:1} |
+------------+-------+
```

命令示例如下。

```
--将t_map中的Key作为数组返回。
select c1,map_values(t_map) from t_table_map;
--返回结果如下。
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | [86, 15] |
| 1001       | [97, 2] |
| 1002       | [99, 1] |
+------------+------+
```

MAP_ZIP_WITH

- 命令格式

```
<K, V1, V2, V3> map<K, V3> map_zip_wuth(map<K, V1> <input1>, map<K, V2> <input2>, function<K, V1, V2, V3> <func>)
```

- 命令说明

  对输入的两个MAP对象input1和input2进行合并得到一个新MAP对象。新MAP的Key是两个MAP的Key的并集。针对新MAP的每一个Key，通过func来计算它的Value。

- 参数说明
  - input1、input2：必填。MAP对象。`map<K, V>`中的`K`、`V`指代MAP对象的Key、Value。
  - func：必填。func有三个输入参数，分别对应MAP的Key、Key相对应的input1以及input2的Value。如果Key在input1或者input2中不存在，func对应参数补充为NULL。

- 返回值说明

  返回func定义的类型。

- 示例

```
--返回{1:[1, 1, 4], 2:[2, 2, 5], 3:[3, NULL, NULL], 4:[4, NULL, 7]}。
select map_zip_with(map(1, 1, 2, 2, 3, null), map(1, 4, 2, 5, 4, 7), (k, v1, v2) -> array(k, v1, v2));
```

TRANSFORM_KEYS

- 命令格式

```
map<K2, V> transform_keys([string <mapDupKeyPolicy>,] map<K1, V> <input>, function<K1, V, K2> <func>)
```

- 命令说明

  对MAP对象input进行变换，保持Value不变，通过func计算新的Key值。

- 参数说明
  - mapDupKeyPolicy：可选。STRING类型。指定出现重复Key时的处理方式。取值范围如下：
    - exception：如果出现重复的Key，返回报错。
    - last_win：如果出现重复的Key，后边的值将覆盖前边的值。

    该参数也可以在Session级别通过 `odps.sql.map.key.dedup.policy` 参数进行设置，例如 `set odps.sql.map.key.dedup.policy=exception;` ，不设置时该参数默认值为last_win。

    > **说明** MaxCompute的行为实现优先以函数中mapDupKeyPolicy的取值为准，当函数未配置mapDupKeyPolicy时，以 `odps.sql.map.key.dedup.policy` 参数的取值为准。

  - input：必填。MAP对象。 `map<K1, V>` 中的 `K1` 、 `V` 指代MAP对象的Key、Value。
  - func：必填。变换的函数（内建函数或自定义函数）或表达式。它的两个输入参数分别对应input的Key和Value， `K2` 指代新MAP的Key类型。

- 返回值说明

  返回MAP类型。如果计算的新Key为NULL，会返回报错。

- 示例

```
--返回{-10:-20, 70:50, 71:101}。
select transform_keys(map(10, -20, 20, 50, -30, 101), (k, v) -> k + v);
--不报错，返回的结果依赖于输入map中元素的顺序。
select transform_keys("last_win", map(10, -20, 20, 50, -30, 100), (k, v) -> k + v);
--因出现重复Key，返回报错。
select transform_keys("exception", map(10, -20, 20, 50, -30, 100), (k, v) -> k + v);
```

TRANSFORM_VALUES

- 命令格式

```
map<K, V2> transform_values(map<K, V1> <input>, function<K, V1, V2> <func>)
```

- 命令说明

  对输入MAP对象input进行变换，保持Key不变，通过func计算新的Value值。

- 参数说明
  - input：必填。MAP对象。 `map<K, V1>` 中的 `K` 、 `V1` 指代MAP对象的Key、Value。
  - func：必填。变换的函数（内建函数或自定义函数）或表达式。它的两个输入参数分别对应input的Key和Value， `V2` 指代新MAP的Value类型。

- 返回值说明

  返回MAP类型。

- 示例

```
--返回{-30:71, 10:-10, 20:NULL}。
select transform_values(map(10, -20, 20, null, -30, 101), (k, v) -> k + v);
```

FIELD

- 命令格式

```
T field(struct <s>, string <fieldName>)
```

- 命令说明

  获取STRUCT对象中成员变量的取值。

- 参数说明
  - s：必填。STRUCT类型对象。STRUCT的结构为 `{f1:T1, f2:T2[, ...]}` ， `f1` 、 `f2` 代表成员变量， `T1` 、 `T2` 分别代表成员变量 `f1` 、 `f2` 的取值。
  - fieldName：必填。STRING类型。STRUCT类型对象的成员变量。

- 返回值说明

  返回STRUCT类型对象的成员变量的取值。

- 示例

```
--返回hello。
select field(named_struct('f1', 'hello', 'f2', 3), 'f1');
```

INLINE

- 命令格式

```
inline(array<struct<f1:T1, f2:T2[, ...]>>)
```

- 命令说明

  将给定的STRUCT数组展开。每个数组元素对应一行，每行每个STRUCT元素对应一列。

- 参数说明

  f1:T1、f2:T2：必填。可以为任意类型。`f1`、`f2`代表成员变量，`T1`、`T2`分别代表成员变量`f1`、`f2`的取值。

- 返回值说明

  返回STRUCT数组展开的数据。

- 示例

  例如表`t_table`的字段为`t_struct struct<user_id:bigint,user_name:string,married:string,weight:double>`，包含数据如下：

```
+-----------+
| t_struct |
+-----------+
| {user_id:10001, user_name:LiLei, married:N, weight:63.5} |
| {user_id:10002, user_name:HanMeiMei, married:Y, weight:43.5} |
+-----------+
```

  命令示例如下。

```
--将t_struct列展开。
select inline(array(t_struct)) from t_table;
--返回结果如下。
+------------+-----------+---------+------------+
| user_id    | user_name | married | weight     |
+------------+-----------+---------+------------+
| 10001      | LiLei     | N       | 63.5       |
| 10002      | HanMeiMei | Y       | 43.5       |
+------------+-----------+---------+------------+
```

STRUCT

- 命令格式

```
struct struct(<value1>,<value2>[, ...])
```

- 命令说明

  使用指定value列表建立STRUCT。

- 参数说明

  value：必填。可以为任意类型。

- 返回值说明

  返回STRUCT类型。Field的名称依次为`col1，col2，…`。

- 示例

```
--返回{col1:a, col2:123, col3:true, col4:56.9}。
select struct('a',123,'true',56.90);
```

NAMED_STRUCT

- 命令格式

```
struct named_struct(string <name1>, T1 <value1>, string <name2>, T2 <value2>[, ...])
```

- 命令说明

  使用指定的name、value列表建立STRUCT。

- 参数说明
  - value：必填。可以为任意类型。
  - name：必填。指定STRING类型的Field名称。此参数为常量。

- 返回值说明

  返回STRUCT类型。Field的名称依次为 `name1，name2，…` 。

- 示例

```
--返回{user_id:10001, user_name:LiLei, married:F, weight:63.5}。
select named_struct('user_id',10001,'user_name','LiLei','married','F','weight',63.50);
```

FROM_JSON

- 命令格式

```
from_json(<jsonStr>, <schema>)
```

- 命令说明

  根据JSON字符串jsonStr和schema信息，返回ARRAY、MAP或STRUCT类型。

- 参数说明
  - jsonStr：必填。输入的JSON字符串。
  - schema：必填。写法与建表语句的类型一致。例如 `array<bigint>` 、 `map<string, array<string>>` 或 `struct<a:int, b:double, `C`:map<string,string>>` 。

    > **说明** STRUCT的Key区分大小写。此外，STRUCT类型还有一种写法 `a bigint, b double` ，等同于 `struct<a:bigint, b:double>` 。

    JSON数据类型与MaxCompute数据类型的对应关系如下。

JSON数据类型	MaxCompute数据类型
OBJECT	STRUCT、MAP、STRING
ARRAY	ARRAY、STRING
NUMBER	TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、STRING
BOOLEAN	BOOLEAN、STRING
STRING	STRING、CHAR、VARCHAR、BINARY、DATE、DATETIME
NULL	所有类型

    > **说明** 对于OBJECT和ARRAY类型，会采用尽可能解析的方式，不匹配的类型会忽略。为了便于使用，所有的JSON类型都可以转换为MaxCompute的STRING类型。同时您需要注意，对应FLOAT、DOUBLE、DECIMAL三种数据类型无法保证小数的精度，如果需要确保精度可以先用STRING类型取出数据，再转换为对应数值类型。

- 返回值说明

  返回ARRAY、MAP或STRUCT类型。

- 示例
  - 示例1：将指定JSON字符串以指定格式输出。命令示例如下。

```
--返回{"a":1,"b":0.8}。
select from_json('{"a":1, "b":0.8}', 'a int, b double');
--返回{"time":"26/08/2015"}。
select from_json('{"time":"26/08/2015"}', 'time string');
--返回{"a":1,"b":0.8}。
select from_json('{"a":1, "b":0.8}', 'a int, b double, c string');
--返回[1,2,3]。
select from_json('[1, 2, 3, "a"]', 'array<bigint>');
--返回{"d":"v","a":"1","b":"[1,2,3]","c":"{}"}。
select from_json('{"a":1,"b":[1,2,3],"c":{},"d":"v"}', 'map<string, string>');
```

  - 示例2：使用 `map_keys` 函数和 `from_json` 函数实现类似JSON_KEYS的作用，获取JSON字符串中所有key的集合。命令示例如下。

```
--返回["a","b"]。
select map_keys(from_json('{"a":1,"b":2}','map<string,string>'));
```

GET_JSON_OBJECT

- 命令格式

```
string get_json_object(string <json>, string <path>)
```

- 命令说明

  在一个标准JSON字符串中，按照path抽取指定的字符串。每次调用该函数时，都会读一次原始数据，因此反复调用可能影响性能和产生费用。您可以通过 `get_json_object` ，结合UDTF，轻松转换JSON格式日志数据，避免多次调用函数，详情请参见利用MaxCompute内建函数及UDTF转换JSON格式日志数据。

- 参数说明
  - json：必填。STRING类型。标准的JSON格式对象，格式为 `{Key:Value, Key:Value,...}` 。如果遇到英文双引号（"），需要用两个反斜杠（\\）进行转义。如果遇到英文单引号（'），需要用一个反斜杠（\）进行转义。
  - path：必填。STRING类型。表示在json中的path，以 `$` 开头。更多path信息，请参见LanguageManual UDF。相关最佳实践案例，请参见JSON数据从OSS迁移至MaxCompute。不同字符的含义如下：
    - `$` ：表示根节点。
    - `.` 或 `['']` ：表示子节点。MaxCompute支持用这两种字符解析JSON对象，当JSON的Key本身包含 `.` 时，可以用 `['']` 来替代。
    - `[]` ： `[number]` 表示数组下标，从0开始。
    - `*` ： `Wildcard for []` ，返回整个数组。 `*` 不支持转义。
- 返回值说明
  - 如果json为空或非法的json格式，返回NULL。
  - 如果json合法，path也存在，则返回对应字符串。
  - 您可以通过在Session级别设置 `odps.sql.udf.getjsonobj.new` 属性来控制函数的返回方式：
    - 当设置 `set odps.sql.udf.getjsonobj.new=true;` 时，函数返回行为采用了保留原始字符串的方式进行输出。

      推荐您使用此配置，函数返回行为更标准，处理数据更方便，性能更好。**如果MaxCompute项目有使用JSON保留字符转义行为的存量作业，建议保留原有行为方式，避免因未验证而直接使用该行为产生错误或正确性问题。**函数返回行为规则如下：
      - 返回值仍是一个JSON字符串，可以继续当做JSON来解析，而不再需要额外使用 `replace` 或 `regexp_replace` 等函数替换反斜线。
      - 一个JSON对象中可以出现相同的Key，可以成功解析。

        ```
        --返回1。
        select get_json_object('{"a":"1","a":"2"}', '$.a');
        ```

      - 支持Emoji表情符号对应的编码字符串。但DataWorks暂不支持输入Emoji表情符号，仅支持通过数据集成等工具直接将Emoji表情符号对应的编码字符串写入MaxCompute，再用 `get_json_object` 函数处理。

        ```
        --返回Emoji符号。
        select get_json_object('{"a":"<Emoji符号>"}', '$.a');
        ```

      - 输出结果按照JSON字符串的原始排序方式输出。

        ```
        --返回{"b":"1","a":"2"}。
        select get_json_object('{"b":"1","a":"2"}', '$');
        ```

    - 当设置 `set odps.sql.udf.getjsonobj.new=false;` 时，函数返回行为采用了JSON保留字符转义的方式进行输出。函数返回行为规则如下：
      - 换行符（\n）、引号（"）等JSON保留字符使用字符串 `'\n'` 、 `'\"'` 显示。
      - 一个JSON对象中不可以出现相同的Key，可能导致无法解析。

        ```
        --返回NULL。
        select get_json_object('{"a":"1","a":"2"}', '$.a');
        ```

      - 不支持解析Emoji表情符号编码的字符串。

        ```
        --返回NULL。
        select get_json_object('{"a":"<Emoji符号>"}', '$.a');
        ```

      - 输出结果按照字典排序方式输出。

        ```
        --返回{"a":"2","b":"1"}。
        select get_json_object('{"b":"1","a":"2"}', '$');
        ```

    > **说明** 自2021年1月21日及之后新创建的MaxCompute项目中， `get_json_object` 函数的返回行为默认为保留原始字符串。2021年1月21日之前创建的MaxCompute项目中， `get_json_object` 函数的返回行为默认为JSON保留字符转义。您可以通过以下示例判断MaxCompute项目中 `get_json_object` 函数采用了哪种行为，执行命令如下：
    >
    > ```
    > select get_json_object('{"a":"[\\"1\\"]"}', '$.a');
    > --JSON保留字符转义的行为返回：
    > [\"1\"]
    > --保留原始字符串的行为返回：
    > ["1"]
    > ```
    >
    > 您可以提工单联系MaxCompute技术支持团队，将您的项目中的 `get_json_object` 函数返回行为切换为保留原始字符串的行为，避免在Session级别频繁设置属性。

- 示例
  - 示例1：提取JSON对象 `src_json.json` 中的信息。命令示例如下。

```
--JSON对象src_json.json的内容。
+----+
json
+----+
{"store":
{"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],
"bicycle":{"price":19.95,"color":"red"}
},
"email":"amy@only_for_json_udf_test.net",
"owner":"amy"
}
--提取owner字段信息，返回amy。
select get_json_object(src_json.json, '$.owner') from src_json;
--提取store.fruit字段第一个数组信息，返回{"weight":8,"type":"apple"}。
select get_json_object(src_json.json, '$.store.fruit[0]') from src_json;
--提取不存在的字段信息，返回NULL。
select get_json_object(src_json.json, '$.non_exist_key') from src_json;
```

  - 示例2：提取数组型JSON对象的信息。命令示例如下。

```
--返回2222。
select get_json_object('{"array":[["aaaa",1111],["bbbb",2222],["cccc",3333]]}','$.array[1][1]');
--返回["h0","h1","h2"]。
set odps.sql.udf.getjsonobj.new=true;
select get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[*]');
--返回["h0","h1","h2"]。
set odps.sql.udf.getjsonobj.new=false;
select get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh');
--返回h1。
select get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[1]');
```

  - 示例3：提取带有 `.` 的JSON对象中的信息。命令示例如下。

```
--创建一张表。
create table mf_json (id string, json string);
--向表中插入数据，Key带.。
insert into table mf_json (id, json) values ("1", "{
\"China.beijing\":{\"school\":{\"id\":0,\"book\":[{\"title\": \"A\",
\"price\": 8.95},{\"title\": \"B\",\"price\": 10.2}]}}}");
--向表中插入数据，Key不带.。
insert into table mf_json (id, json) values ("2", "{
\"China_beijing\":{\"school\":{\"id\":0,\"book\":[{\"title\": \"A\",
\"price\": 8.95},{\"title\": \"B\",\"price\": 10.2}]}}}");
--取id的值，查询key为China.beijing，返回0。由于包含.，只能用['']来解析。
select get_json_object(json, "$['China.beijing'].school['id']") from mf_json where id =1;
--取id的值，查询key为China_beijing，返回0。查询方法有如下两种。
select get_json_object(json, "$['China_beijing'].school['id']") from mf_json where id =2;
select get_json_object(json, "$.China_beijing.school['id']") from mf_json where id =2;
```

  - 示例4：JSON输入为空或非法格式。命令示例如下。

```
--返回NULL。
select get_json_object('','$.array[1][1]');
--返回NULL。
select get_json_object('"array":["aaaa",1111],"bbbb":["cccc",3333]','$.array[1][1]');
```

  - 示例5：JSON字符串涉及转义。命令示例如下。

```
set odps.sql.udf.getjsonobj.new=true;
--返回"1"。
select get_json_object('{"a":"\\"1\\"","b":"2"}', '$.a');
--返回'1'。
select get_json_object('{"a":"\'1\'","b":"2"}', '$.a');
```

JSON_TUPLE

- 命令格式

```
string json_tuple(string <json>, string <key1>, string <key2>,...)
```

- 命令说明

  用于一个标准的JSON字符串中，按照输入的一组键 `(key1,key2,...)` 抽取各个键指定的字符串。

- 参数说明
  - json：必填。STRING类型，标准的JSON格式字符串。
  - key：必填。STRING类型，用于描述在JSON中的 `path` ，一次可输入多个，不能以美元符号（$）开头。MaxCompute支持用 `.` 或 `['']` 这两种字符解析JSON，当JSON的Key本身包含 `.` 时，可以用 `['']` 来替代。
- 返回值

  返回STRING类型。

  > 说明
  > - 如果JSON为空或者为非法的JSON格式，返回NULL。
  > - 如果键Key为空或者不合法（JSON中不存在）返回NULL。
  > - 如果JSON合法，键Key也存在，则返回对应字符串。
  > - 支持包含中文的JSON数据解析。
  > - 支持多层嵌套的JSON数据解析。
  > - 支持包含多重嵌套的数组的JSON数据解析。
  > - 解析行为和设置了 `set odps.sql.udf.getjsonobj.new=true;` 后的GET_JSON_OBJECT的行为保持一致。在需要对同一个JSON字符串多次解析的情况下，相比于多次调用GET_JSON_OBJECT，JSON_TUPLE可以一次输入多个Key，且JSON字符串只被解析一次，效率更高。
  > - JSON_TUPLE是UDTF，在需要选取其他列时应配合Lateral View使用。
- 示例

```
--创建一张表school。
create table school (id string, json string);
--向表中插入数据。
insert into school (id, json) values ("1", "{
\"school\": \"湖畔大学\", \"地址\":\"杭州\", \"SchoolRank\": \"00\",
\"Class1\":{\"Student\":[{\"studentId\":1,\"scoreRankIn3Year\":[1,2,[3,2,6]]},
{\"studentId\":2,\"scoreRankIn3Year\":[2,3,[4,3,1]]}]}}");
```

  - 示例1：提取JSON对象信息。命令示例如下。

```
select json_tuple(school.json,"SchoolRank","Class1") as (item0, item1) from school;
--等效于如下语句。
select get_json_object(school.json,"$.SchoolRank") item0,get_json_object(school.json,"$.Class1") item1 from school;
--返回结果如下。
+-------+-------+
| item0 | item1 |
+-------+-------+
| 00    | {"Student":[{"studentId":1,"scoreRankIn3Year":[1,2,[3,2,6]]},{"studentId":2,"scoreRankIn3Year":[2,3,[4,3,1]]}]} |
+-------+-------+
```

  - 示例2：使用 `['']` 提取JSON对象信息。命令示例如下。

```
select json_tuple(school.json,"school","['Class1'].Student") as (item0, item2) from school where id=1;
--返回结果如下。
+-------+-------+
| item0 | item2 |
+-------+-------+
| 湖畔大学 | [{"studentId":1,"scoreRankIn3Year":[1,2,[3,2,6]]},{"studentId":2,"scoreRankIn3Year":[2,3,[4,3,1]]}] |
```

  - 示例3：支持解析包含中文的JSON数据。命令示例如下。

```
select json_tuple(school.json,"校名","地址") as (item0,item1) from school;
--返回结果如下。
+-------+-------+
| item0 | item1 |
+-------+-------+
| 湖畔大学 | 杭州 |
+-------+-------+
```

  - 示例4：支持解析多层嵌套的JSON数据。命令示例如下。

```
select sc.Id, q.item0, q.item1
from school sc lateral view json_tuple(sc.json,"Class1.Student[*].studentId","Class1.Student[0].scoreRankIn3Year") q as item
0,item1;
--返回结果如下。
+------------+-------+-------+
| id         | item0 | item1 |
+------------+-------+-------+
| 1          | [1,2] | [1,2,[3,2,6]] |
+------------+-------+-------+
```

- 示例5：支持解析包含多重嵌套数组的JSON数据。命令示例如下。

```
select sc.Id, q.item0, q.item1
from school sc lateral view json_tuple(sc.json,"Class1.Student[0].scoreRankIn3Year[2]","Class1.Student[0].scoreRankIn3Year[2
][1]") q as item0,item1;
--返回结果如下。
+------------+-------+-------+
| id         | item0 | item1 |
+------------+-------+-------+
| 1          | [3,2,6] | 2     |
+------------+-------+-------+
```

TO_JSON

- 命令格式

```
to_json(<expr>)
```

- 命令说明

  将给定的复杂类型expr，以JSON字符串格式输出。

- 参数说明

  expr：必填。ARRAY、MAP、STRUCT复杂类型。

  > **说明** 如果输入为STRUCT类型（ `struct<key1:value1, key2:value2` ）：
  > - 转换为JSON字符串时，Key会全部转为小写。
  > - `value` 如果为NULL，则不输出 `value` 本组的数据。例如 `value2` 为NULL，则 `key2:value2` 不会输出到JSON字符串。

- 示例
  - 示例1：将指定复杂类型以指定格式输出。命令示例如下。

```
--返回{"a":1,"b":2}。
select to_json(named_struct('a', 1, 'b', 2));
--返回{"time":"26/08/2015"}。
select to_json(named_struct('time', "26/08/2015"));
--返回[{"a":1,"b":2}]。
select to_json(array(named_struct('a', 1, 'b', 2)));
--返回{"a":{"b":1}}。
select to_json(map('a', named_struct('b', 1)));
--返回{"a":1}。
select to_json(map('a', 1));
--返回[{"a":1}]。
select to_json(array((map('a', 1))));
```

  - 示例2：输入为STRUCT类型的特殊情况。命令示例如下。

```
--返回{"a":"B"}。STRUCT类型转换为JSON字符串时，key会全部转为小写。
select to_json(named_struct("A", "B"));
--返回{"k2":"v2"}。NULL值所在组的数据，不会输出到JSON字符串。
select to_json(named_struct("k1", cast(null as string), "k2", "v2"));
```

3.9.9. 其他函数

MaxCompute SQL提供了开发过程中常见的其他函数，您可以根据实际需要选择合适的函数。本文为您提供MaxCompute SQL支持的CAST、DECODE、LEAST、SPLIT等函数的命令格式、参数说明及示例。

函数	功能
BASE64	将二进制表示值转换为BASE64编码格式字符串。
BETWEEN AND表达式	筛选满足区间条件的数据。
CASE WHEN表达式	根据表达式的计算结果，灵活地返回不同的值。
CAST	将表达式的结果转换为目标数据类型。
COALESCE	返回参数列表中第一个非NULL的值。
COMPRESS	对STRING或BINARY类型输入参数按照GZIP算法进行压缩。
CRC32	计算字符串或二进制数据的循环冗余校验值。

函数	功能
DECODE	实现 `if-then-else` 分支选择的功能。
DECOMPRESS	对BINARY类型输入参数按照GZIP算法进行解压。
GET_IDCARD_AGE	根据身份证号码返回当前的年龄。
GET_IDCARD_BIRTHDAY	根据身份证号码返回出生日期。
GET_IDCARD_SEX	根据身份证号码返回性别。
GET_USER_ID	获取当前账号的账号ID。
GREATEST	返回输入参数中最大的值。
HASH	根据输入参数计算Hash值。
IF	判断指定的条件是否为真。
LEAST	返回输入参数中最小的值。
MAX_PT	返回分区表的一级分区的最大值。
NULLIF	比较两个入参是否相等。
NVL	指定值为NULL的参数的返回结果。
ORDINAL	将输入变量按从小到大排序后，返回指定位置的值。
PARTITION_EXISTS	查询指定的分区是否存在。
SAMPLE	对所有读入的列值，采样并过滤掉不满足采样条件的行。
SHA	计算字符串或二进制数据的SHA-1哈希值。
SHA1	计算字符串或二进制数据的SHA-1哈希值。
SHA2	计算字符串或二进制数据的SHA-2哈希值。
SIGN	判断正负值属性。
SPLIT	将字符串按照指定的分隔符分割后返回数组。
STACK	将指定的参数组分割为指定的行数。
STR_TO_MAP	将字符串按照指定的分隔符分割得到Key和Value。
TABLE_EXISTS	查询指定的表是否存在。
TRANS_ARRAY	将一行数据转为多行的UDTF，将列中存储的以固定分隔符格式分隔的数组转为多行。
TRANS_COLS	将一行数据转为多行数据的UDTF，将不同的列拆分为不同的行。
UNBASE64	将BASE64编码格式字符串转换为二进制表示值。
UNIQUE_ID	返回一个随机ID，运行效率高于UUID函数。
UUID	返回一个随机ID。

BASE64

- 命令格式

```
string base64(binary <value>)
```

- 命令说明

  将value从二进制转换为BASE64编码格式字符串。

- 参数说明

  value：必填。BINARY类型。待转换参数值。

- 返回值说明

  返回STRING类型。输入参数为NULL时，返回结果为NULL。

- 示例

- 示例1：将 `cast ('alibaba' as binary)` 二进制结果转换为BASE64编码格式字符串。命令示例如下：

```
--返回YWxpYmFiYQ==。
select base64(cast ('alibaba' as binary));
```

- 示例2：输入参数为NULL。命令示例如下：

```
--返回NULL。
select base64(null);
```

BETWEEN AND表达式

- 命令格式

```
<a> [NOT] between <b> and <c>
```

- 命令说明

  筛选满足a的值位于b和c之间或不在b和c之间的数据。

- 参数说明
  - a：必填。待筛选的字段。
  - b、c：必填。指定的区间条件。数据类型必须与a的数据类型保持一致。
- 返回值说明

  返回满足条件的数据。

  如果a、b或c为空，返回结果为空。

- 示例

  例如表 `emp` 中的数据如下。

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

  查询 `sal` 大于等于1000小于等于1500之间的数据，命令示例如下。

```
select * from emp where sal between 1000 and 1500;
```

  返回结果如下。

```
+-------+-------+-----+------------+------------+------------+------------+------------+
| empno | ename | job | mgr        | hiredate   | sal        | comm       | deptno     |
+-------+-------+-----+------------+------------+------------+------------+------------+
| 7521  | WARD  | SALESMAN | 7698  | 1981-02-22 00:00:00 | 1250.0     | 500.0      | 30  |
| 7654  | MARTIN | SALESMAN | 7698 | 1981-09-28 00:00:00 | 1250.0     | 1400.0     | 30 |
| 7844  | TURNER | SALESMAN | 7698 | 1981-09-08 00:00:00 | 1500.0     | 0.0        | 30 |
| 7876  | ADAMS | CLERK | 7788  | 1987-05-23 00:00:00 | 1100.0     | NULL     | 20   |
| 7934  | MILLER | CLERK | 7782  | 1982-01-23 00:00:00 | 1300.0     | NULL      | 10  |
| 7956  | TEBAGE | CLERK | 7748  | 1982-12-30 00:00:00 | 1300.0     | NULL      | 10  |
+-------+-------+-----+------------+------------+------------+------------+------------+
```

CASE WHEN表达式

- 命令格式

  MaxCompute提供以下两种 `case when` 格式：

```
case <value>
when <value1> then <result1>
when <value2> then <result2>
...
else <resultn>
end
```

```
case
when (<_condition1>) then <result1>
when (<_condition2>) then <result2>
when (<_condition3>) then <result3>
...
else <resultn>
end
```

- 命令说明

  根据value或_condition的计算结果，灵活地返回不同的result值。

- 参数说明
  - value：必填。比较的值。
  - _condition：必填。指定判断条件。
  - result：必填。返回值。

- 返回值说明
  - 如果result类型只有BIGINT、DOUBLE，统一转为DOUBLE后，再返回结果。
  - 如果result类型中有STRING类型，则统一转为STRING后，再返回结果。如果无法进行类型转换，例如BOOLEAN类型无法转换为STRING类型，则会返回报错。
  - 不允许其他类型之间的转换。

- 示例

  例如表 `sale_detail` 的字段为 `shop_name string, customer_id string, total_price double,` ，包含数据如下。

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
+------------+-------------+-------------+------------+------------+
```

  命令示例如下。

```
select
case
when region='china' then 'default_region'
when region like 'shang%' then 'sh_region'
end as region
from sale_detail;
```

  返回结果如下。

```
+------------+
| region     |
+------------+
| default_region |
| default_region |
| default_region |
| sh_region  |
| sh_region  |
| sh_region  |
+------------+
```

CAST

- 命令格式

```
cast(<expr> as <type>)
```

- 命令说明

将expr的结果转换成目标数据类型type。

- 参数说明
  - expr：必填。待转换数据源。
  - type：必填。目标数据类型。用法如下：
    - `cast(double as bigint)`：将DOUBLE数据类型值转换成BIGINT数据类型。
    - `cast(string as bigint)`：在将字符串转为BIGINT数据类型时，如果字符串中是以整型表达的数字，则会直接将它们转为BIGINT类型。如果字符串中是以浮点数或指数形式表达的数字，则会先转为DOUBLE数据类型，再转为BIGINT数据类型。
    - `cast(string as datetime)` 或 `cast(datetime as string)`：会采用默认的日期格式 `yyyy-mm-dd hh:mi:ss`。
- 返回值说明

  返回值为转换后的目标数据类型。
- 示例
  - 示例1：常见用法。命令示例如下。

```
--返回1。
select cast('1' as bigint);
```

  - 示例2：异常用法，如果转换不成功或遇到不支持的类型转换，会引发异常。**错误命令示例**如下。

```
select cast('abc' as bigint);
```

COALESCE

- 命令格式

```
coalesce(<expr1>, <expr2>, ...)
```

- 命令说明

  返回 `<expr1>, <expr2>, ...` 中第一个非NULL的值。
- 参数说明

  expr：必填。待验证的值。
- 返回值说明

  返回值类型和参数数据类型相同。
- 示例
  - 示例1：常见用法。命令示例如下。

```
--返回1。
select coalesce(null,null,1,null,3,5,7);
```

  - 示例2：参数值类型未定义，返回报错。
    - **错误命令示例**

```
--参数abc未定义类型，系统引擎无法识别，导致返回报错。
select coalesce(null,null,1,null,abc,5,7);
```

    - **正确命令示例**

```
select coalesce(null,null,1,null,'abc',5,7);
```

  - 示例3：在非表读取的情况下，如果参数值全部为空，则返回报错。**错误命令示例**如下。

```
--返回报错，会提示至少有一个参数值非NULL。
select coalesce(null,null,null,null);
```

  - 示例4：在表读取的情况下，如果参数值全部为空，则返回NULL。

    原始数据表：

```
+------------+-------------+------------+
| shop_name | customer_id | toal_price |
+------------+-------------+------------+
| ad        | 10001       | 100.0      |
| jk        | 10002       | 300.0      |
| ad        | 10003       | 500.0      |
| tt        | NULL        | NULL       |
+------------+-------------+------------+
```

    由原始数据表可以看出，tt的值全部为空。则执行以下语句后返回值为NULL。

```
select coalesce(customer_id,total_price) from sale_detail where shop_name='tt';
```

COMPRESS

- 命令格式

```
binary compress(string <str>)
binary compress(binary <bin>)
```

- 命令说明

  将str或bin按照GZIP算法进行压缩。

- 参数说明
  - str：必填。STRING类型。
  - bin：必填。BINARY类型。
- 返回值说明

  返回BINARY类型。输入参数为NULL时，返回结果为NULL。

- 示例
  - 示例1：对字符串 `hello` 按照GZIP算法进行压缩。命令示例如下。

    ```
    --返回=1F=8B=08=00=00=00=00=00=00=03=CBH=CD=C9=C9=07=00=86=A6=106=05=00=00=00。
    select compress('hello');
    ```

  - 示例2：输入参数为空。命令示例如下。

    ```
    --返回=1F=8B=08=00=00=00=00=00=00=03=03=00=00=00=00=00=00=00=00=00。
    select compress('');
    ```

  - 示例3：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select compress(null);
    ```

CRC32

- 命令格式

```
bigint crc32(string|binary <expr>)
```

- 命令说明

  计算字符串或二进制类型的expr的循环冗余校验值。

- 参数说明

  expr：必填。STRING或BINARY类型。

- 返回值说明

  返回BIGINT类型。返回规则如下：
  - 输入参数为NULL时，返回结果为NULL。
  - 输入参数为空时，返回0。
- 示例
  - 示例1：计算字符串 `ABC` 的循环冗余校验值。命令示例如下。

    ```
    --返回2743272264。
    select crc32('ABC');
    ```

  - 示例2：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select crc32(null);
    ```

DECODE

- 命令格式

```
decode(<expression>, <search>, <result>[, <search>, <result>]...[, <default>])
```

- 命令说明

  实现 `if-then-else` 分支选择的功能。

- 参数说明
  - expression：必填。要比较的表达式。
  - search：必填。与expression进行比较的搜索项。
  - result：必填。search和expression的值匹配时的返回值。

- default：可选。如果所有的搜索项都不匹配，则返回default值，如果未指定，则返回NULL。

> 说明
> - 所有的result数据类型必须一致或为NULL。不一致的数据类型会返回报错。
> - 所有的search和expression数据类型必须一致，否则会返回报错。

- 返回值说明
  - 如果匹配，返回result。
  - 如果没有匹配，返回default。
  - 如果没有指定default，返回NULL。
  - 如果search选项有重复且匹配时，会返回第一个值。
  - 通常，MaxCompute SQL在计算 `NULL=NULL` 时返回NULL，但在该函数中，NULL与NULL的值是相等的。
- 示例

  例如表 `sale_detail` 的字段为 `shop_name string, customer_id string, total_price double,` ，包含数据如下。

```
+------------+-------------+-------------+------------+------------+
| shop_name  | customer_id | total_price | sale_date  | region     |
+------------+-------------+-------------+------------+------------+
| s1         | c1          | 100.1       | 2013       | china      |
| s2         | c2          | 100.2       | 2013       | china      |
| s3         | c3          | 100.3       | 2013       | china      |
| null       | c5          | NULL        | 2014       | shanghai   |
| s6         | c6          | 100.4       | 2014       | shanghai   |
| s7         | c7          | 100.5       | 2014       | shanghai   |
+------------+-------------+-------------+------------+------------+
```

  命令示例如下。

```
--当customer_id的值为c1时，返回Taobao；值为c2时，返回Alipay；值为c3时，返回Aliyun；值为NULL时，返回N/A；其他场景返回Others。
select
decode(customer_id,
'c1', 'Taobao',
'c2', 'Alipay',
'c3', 'Aliyun',
Null, 'N/A',
'Others') as result
from sale_detail;
--等效于如下语句。
if customer_id = c1 then
result := 'Taobao';
elsif customer_id = c2 then
result := 'Alipay';
elsif customer_id = c3 then
result := 'Aliyun';
...
else
result := 'Others';
end if;
```

  返回结果如下。

```
+------------+
| result     |
+------------+
| Others     |
| Others     |
| Others     |
| Taobao     |
| Alipay     |
| Aliyun     |
+------------+
```

DECOMPRESS

- 命令格式

```
binary decompress(binary <bin>)
```

- 命令说明

  将bin按照GZIP算法进行解压。

- 参数说明

  bin：必填。BINARY类型。

- 返回值说明

  返回BINARY类型。输入参数为NULL时，返回结果为NULL。

- 示例
  - 示例1：对字符串 `hello, world` 的压缩结果进行解压并转换为字符串格式。命令示例如下。

    ```
    --返回hello, world。
    select cast(decompress(compress('hello, world')) as string);
    ```

  - 示例2：输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select decompress(null);
    ```

GET_IDCARD_AGE

- 命令格式

  ```
  get_idcard_age(<idcardno>)
  ```

- 命令说明

  根据身份证号码返回当前的年龄，即当前年份减去身份证号码中标识的出生年份的差值。

- 参数说明

  idcardno：必填。STRING类型，15位或18位身份证号码。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性。如果校验不通过会返回NULL。

- 返回值说明

  返回BIGINT类型。输入为NULL时，返回NULL。

GET_IDCARD_BIRTHDAY

- 命令格式

  ```
  get_idcard_birthday(<idcardno>)
  ```

- 命令说明

  根据身份证号码返回出生日期。

- 参数说明

  idcardno：必填。STRING类型，15位或18位身份证号码。在计算时，会根据省份代码以及最后一位校验码检查身份证的合法性。如果校验不通过，则返回NULL。

- 返回值说明

  返回DATETIME类型。输入为NULL时，返回NULL。

GET_IDCARD_SEX

- 命令格式

  ```
  get_idcard_sex(<idcardno>)
  ```

- 命令说明

  根据身份证号码返回性别，值为 `M`（男）或 `F`（女）。

- 参数说明

  idcardno：必填。STRING类型，15位或18位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性。如果校验不通过，则返回NULL。

- 返回值说明

  返回STRING类型。输入为NULL时，返回NULL。

GET_USER_ID

- 命令格式

  ```
  get_user_id()
  ```

- 命令说明

  获取当前账号的账号ID，即用户ID或UID。

- 参数说明

无需输入参数。

- 返回值说明

  返回当前账号的账号ID。

- 示例

```
select get_user_id();
--返回结果如下。
+------------+
| _c0        |
+------------+
| 1117xxxxxxxx8519 |
+------------+
```

GREATEST

- 命令格式

```
greatest(<var1>, <var2>[,...])
```

- 命令说明

  返回输入参数中最大的值。

- 参数说明

  var1、var2：必填。BIGINT、DOUBLE、DECIMAL、DATETIME或STRING类型。

- 返回值说明
  - 返回输入参数中的最大值。当不存在隐式转换时，返回值同输入参数数据类型。
  - NULL为最小值。
  - 当输入参数数据类型不相同时，DOUBLE、BIGINT、DECIMAL、STRING之间的比较会转换为DOUBLE类型；STRING、DATETIME的比较会转换为DATETIME类型。不允许其他的隐式转换。
  - 当 `set odps.sql.hive.compatible=true;` 时，任意参数输入为NULL，返回结果为NULL。

HASH

- 命令格式
  - 当MaxCompute项目为Hive兼容模式时，命令格式如下。

```
int hash(<value1>, <value2>[, ...]);
```

  - 当MaxCompute项目非Hive兼容模式时，命令格式如下。

```
bigint hash(<value1>, <value2>[, ...]);
```

- 命令说明

  对value1、value2进行散列运算得到一个Hash值。

- 参数说明

  value1、value2：必填。待计算Hash值的参数，各参数的类型可以不相同。Hive及非Hive兼容模式下支持的数据类型不相同，具体如下：
  - Hive兼容模式：TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、BOOLEAN、STRING、CHAR、VARCHAR、DATETIME、DATE。
  - 非Hive兼容模式：BIGINT、DOUBLE、BOOLEAN、STRING、DATETIME。

> **说明** 对于两个相同的输入，返回的Hash值一定是相同的。但是反过来，如果两个Hash值相同，并不保证输入值一定相等（可能出现哈希碰撞）。

- 返回值说明

  返回INT或BIGINT类型。当输入参数为空或NULL时，返回结果为0。

- 示例
  - 示例1：计算相同数据类型的输入参数的Hash值。命令示例如下。

```
--返回66。
select hash(0, 2, 4);
```

  - 示例2：计算不同数据类型的输入参数的Hash值。命令示例如下。

```
--返回97。
select hash(0, 'a');
```

- 示例3：任一输入参数为空或NULL。命令示例如下。

```
--返回0。
select hash(0, null);
--返回0。
select hash(0, '');
```

IF

- 命令格式

```
if(<testCondition>, <valueTrue>, <valueFalseOrNull>)
```

- 命令说明

  判断testCondition是否为真。如果为真，返回valueTrue的值，否则返回valueFalseOrNull的值。

- 参数说明
  - testCondition：必填。要判断的表达式，BOOLEAN类型。
  - valueTrue：必填。表达式testCondition为True时，返回的值。
  - valueFalseOrNull：表达式testCondition为False时，返回的值，可以设为NULL。
- 返回值说明

  返回值类型和参数valueTrue或valueFalseOrNull的数据类型一致。

- 示例

```
--返回200。
select if(1=2, 100, 200);
```

LEAST

- 命令格式

```
least(<var1>, <var2>[,...])
```

- 命令说明

  返回输入参数中的最小值。

- 参数说明

  var：必填。输入参数值。BIGINT、DOUBLE、DECIMAL、DATETIME或STRING类型。

- 返回值说明
  - 输入参数中的最小值。当不存在隐式转换时，返回值同输入参数类型。
  - 当有类型转换时，DOUBLE、BIGINT、STRING之间的转换返回DOUBLE类型；STRING、DATETIME之间的转换返回DATETIME类型；DECIMAL、DOUBLE、BIGINT和STRING之间的转换返回DECIMAL类型。不允许其他的隐式类型转换。
  - NULL为最小值。
  - 如果所有参数值都为NULL，返回结果为NULL。
- 示例

```
--返回2。
select least(5, 2, 7);
```

MAX_PT

- 命令格式

```
max_pt(<table_full_name>)
```

- 命令说明

  返回分区表的一级分区中有数据的分区的最大值，按字母排序，且读取该分区下对应的数据。

  `max_pt` 函数也可以使用标准SQL实现，`select * from table where pt = max_pt("table");` 可以改写为 `select * from table where pt = (select max(pt) from table);` 。

  > **说明** MaxCompute未提供 `min_pt` 函数。如果您需要获取分区表中有数据的最小分区，无法使用SQL语句 `select * from table where pt = min_pt("table");` 实现与 `max_pt` 函数类似的功能，但可以使用标准SQL语句 `select * from table where pt = (select min(pt) from table);` 实现相同的效果。

- 参数说明

  table_full_name：必填。STRING类型。指定表名。必须对表有读权限。

- 返回值说明

返回最大的一级分区的值。

> 说明
> 如果只是用 `alter table` 的方式新加了一个分区，但是此分区中并无任何数据，则此分区不会做为返回值。

- 示例
  - 示例1：例如tbl是分区表，该表对应的分区为20120901和20120902，且都有数据。则以下语句中 `max_pt` 返回值为 `'20120902'` 。MaxCompute SQL语句会读出 `pt='20120902'` 分区下的数据。命令示例如下。

```
select * from tbl where pt=max_pt('myproject.tbl');
--等效于如下语句。
select * from tbl where pt = (select max(pt) from myproject.tbl);
```

  - 示例2：在多级分区场景，使用标准SQL实现获取最大分区下的数据。命令示例如下。

```
select * from table where pt1 = (select max(pt1) from table) and pt2 = (select max(pt2) from table where pt1 = (select max(p
t1) from table));
```

NULLIF

- 命令格式

```
T nullif(T <expr1>, T <expr2>)
```

- 命令说明

  比较expr1和expr2的值，二者相等时返回NULL，否则返回expr1。

- 参数说明

  expr1、expr2：必填。任意类型的表达式。 `T` 指代输入数据类型，可以是MaxCompute支持的所有数据类型。

- 返回值说明

  返回NULL或expr1。

- 示例

```
--返回2。
select nullif(2, 3);
--返回NULL。
select nullif(2, 2);
--返回3。
select nullif(3, null);
```

NVL

- 命令格式

```
nvl(T <value>, T <default_value>)
```

- 命令说明

  如果value值为NULL，返回default_value，否则返回value。两个参数的数据类型必须一致。

- 参数说明
  - value：必填。输入参数。 `T` 指代输入数据类型，可以是MaxCompute支持的所有数据类型。
  - default_value：必填。替换后的值。必须与value的数据类型保持一致。

- 示例

  例如表 `t_data` 的3个列分别为 `c1 string` 、 `c2 bigint` 、 `c3 datetime` 。表中数据如下。

```
+----+------------+------------+
| c1 | c2 | c3 |
+----+------------+------------+
| NULL | 20 | 2017-11-13 05:00:00 |
| ddd | 25 | NULL |
| bbb | NULL | 2017-11-12 08:00:00 |
| aaa | 23 | 2017-11-11 00:00:00 |
+----+------------+------------+
```

  通过 `nvl` 函数将 `c1` 中为NULL的值输出为00000， `c2` 中为NULL的值输出为0， `c3` 中为NULL的值输出为 `-` ，命令示例如下。

```
select nvl(c1,'00000'),nvl(c2,0),nvl(c3,'-') from nvl_test;
--返回结果如下。
+-----+------------+-----+
| _c0 | _c1 | _c2 |
+-----+------------+-----+
| 00000 | 20 | 2017-11-13 05:00:00 |
| ddd | 25 | - |
| bbb | 0 | 2017-11-12 08:00:00 |
| aaa | 23 | 2017-11-11 00:00:00 |
+-----+------------+-----+
```

ORDINAL

- 命令格式

```
ordinal(bigint <nth>, <var1>, <var2>[,...])
```

- 命令说明

  将输入变量按从小到大排序后，返回nth指定位置的值。

- 参数说明
  - nth：必填。BIGINT类型。指定要返回的位置值为NULL时，返回NULL。
  - var：必填。BIGINT、DOUBLE、DATETIME或STRING类型。待排序的值。
- 返回值说明
  - 排在第nth位的值，当不存在隐式转换时返回值同输入参数数据类型。
  - 当有类型转换时，DOUBLE、BIGINT、STRING之间的转换返回DOUBLE类型；STRING、DATETIME之间的转换返回DATETIME类型。不允许其他的隐式转换。
  - NULL为最小值。
- 示例

```
--返回2。
select ordinal(3, 1, 3, 2, 5, 2, 4, 6);
```

PARTITION_EXISTS

- 命令格式

```
boolean partition_exists(string <table_name>, string... <partitions>)
```

- 命令说明

  查询指定的分区是否存在。

- 参数说明
  - table_name：必填。表名称，STRING类型。表名称中可以指定项目空间名称，例如`my_proj.my_table`。如果不指定项目空间名称则默认为当前项目空间。
  - partitions：必填。分区名称，STRING类型。按照表分区列的顺序依次写出分区值，分区值数目必须与分区列数目一致。
- 返回值说明

  返回BOOLEAN类型。如果指定的分区存在返回True，否则返回False。

- 示例

```
--创建分区表foo。
create table foo (id bigint) partitioned by (ds string, hr string);
--为分区表foo新增分区。
alter table foo add partition (ds='20190101', hr='1');
--查询分区ds='20190101'和hr='1'是否存在。返回结果为True。
select partition_exists('foo', '20190101', '1');
```

SAMPLE

- 命令格式

```
boolean sample(<x>, <y>, [<column_name1>, <column_name2>[,...]])
```

- 命令说明

  基于所有读入的column_name的值，系统根据x、y的设置做采样，并过滤掉不满足采样条件的行。

- 参数说明

- x、y：x必填。BIGINT类型，取值范围为大0的整型常量。表示哈希为x份，取第y份。

  y可选，省略时默认取第一份。如果省略参数中的y，则必须同时省略column_name。

  x、y为其它类型或小于等于0时抛异常，如果y大于x时也返回异常。x、y任一输入为NULL时，返回NULL。

- column_name：可选。采样的目标列。该参数省略时将根据x、y的值随机采样。任意类型，列的值可以为NULL。不做隐式类型转换。如果column_name为常量NULL，则返回报错。

  > **说明** 为避免NULL值带来的数据倾斜，对于column_name中为NULL的值，会在x份中进行均匀哈希。如果不指定column_name，则数据量比较少时输出不一定均匀，在这种情况下建议指定column_name，以获得较好的输出结果。

- 返回值说明

  返回BOOLEAN类型。

- 示例

  例如存在表 `tbla` ，表内有列名为 `cola` 的列。

```
--表示数值会根据cola hash为4份，取第1份。返回值为True。
select * from tbla where sample (4, 1 , cola);
--表示数值会对每行数据做随机哈希分配为4份，取第2份。返回值为True。
select * from tbla where sample (4, 2);
```

SHA

- 命令格式

```
string sha(string|binary <expr>)
```

- 命令说明

  计算字符串或者二进制类型的expr的SHA-1哈希值，并以十六进制字符串格式返回。

- 参数说明

  expr：必填。STRING或BINARY类型。

- 返回值说明

  返回STRING类型。输入参数为NULL时，返回结果为NULL。

- 示例

  - 示例1：计算字符串 `ABC` 的SHA哈希值。命令示例如下。

```
--返回3c01bdbb26f358bab27f267924aa2c9a03fcfdb8。
select sha('ABC');
```

  - 示例2：输入参数为NULL。命令示例如下。

```
--返回NULL。
select sha(null);
```

SHA1

- 命令格式

```
string sha1(string|binary <expr>)
```

- 命令说明

  计算字符串或者二进制类型的expr的SHA-1哈希值，并以十六进制字符串格式返回。

- 参数说明

  expr：必填。STRING或BINARY类型。

- 返回值说明

  返回STRING类型。输入参数为NULL时，返回结果为NULL。

- 示例

  - 示例1：计算字符串 `ABC` 的SHA-1哈希值。命令示例如下。

```
--返回3c01bdbb26f358bab27f267924aa2c9a03fcfdb8。
select sha1('ABC');
```

  - 示例2：输入参数为NULL。命令示例如下。

```
--返回NULL。
select sha1(null);
```

SHA2

- 命令格式

```
string sha2(string|binary <expr>, bigint <number>)
```

- 命令说明

  计算字符串或者二进制类型的expr的SHA-2哈希值，以指定的number格式返回。

- 参数说明
  - expr：必填。STRING或BINARY类型。
  - number：必填。BIGINT类型。哈希位长，取值必须是224、256、384、512、0（同256）。
- 返回值说明

  返回STRING类型。返回规则如下：
  - 任一输入参数为NULL时，返回结果为NULL。
  - 如果number取值不在允许的取值范围内，返回结果为NULL。
- 示例
  - 示例1：计算字符串 `ABC` 的SHA-2哈希值。命令示例如下。

    ```
    --返回b5d4045c3f466fa91fe2cc6abe79232a1a57cdf104f7a26e716e0a1e2789df78。
    select sha2('ABC', 256);
    ```

  - 示例2：任一输入参数为NULL。命令示例如下。

    ```
    --返回NULL。
    select sha2('ABC', null);
    ```

SIGN

- 命令格式

```
sign(<x>)
```

- 命令说明

  用于判断x的正负值属性。

- 参数说明

  x：必填。DOUBLE类型或者DECIMAL类型。可以为常量、函数或表达式。

- 返回值说明
  - 当x为正值时，返回1.0。
  - 当x为负值时，返回-1.0。
  - 当x为0时，返回0.0。
  - 当x为空时，返回报错。
- 示例

```
--返回-1.0。
select sign(5-13);
```

SPLIT

- 命令格式

```
split(<str>, <pat>)
```

- 命令说明

  通过pat将str分割后返回数组。

- 参数说明
  - str：必填。STRING类型。指被分割的字符串。
  - pat：必填。STRING类型的分隔符。支持正则表达式。更多正则表达式信息，请参见正则表达式规范。
- 返回值说明

  返回ARRAY数组。数组中的元素为STRING类型。

- 示例

```
--返回[a,  b,  c]。
select split("a, b, c", ",");
```

STACK

- 命令格式

```
stack(n, expr1, ..., exprk)
```

- 命令说明

  将 `expr1, ..., exprk` 分割为n行，除非另有说明，否则输出结果使用默认的列名 `col0、col1...` 。

- 参数说明
  - n：必填。分割的行数。
  - expr：必填。待分割的参数。 `expr1, ..., exprk` 必须是整型，且参数数目必须是n的整数倍，需要能分割为完整的n行，否则返回报错。

- 返回值说明

  返回n行，列数为参数数量除以n的商的数据集。

- 示例

```
--将1, 2, 3, 4, 5, 6排为3行。
select stack(3, 1, 2, 3, 4, 5, 6);
--返回结果如下。
+------+------+
| col0 | col1 |
+------+------+
| 1    | 2    |
| 3    | 4    |
| 5    | 6    |
+------+------+
--将'A',10,date '2015-01-01','B',20,date '2016-01-01'排为两行.
select stack(2,'A',10,date '2015-01-01','B',20,date '2016-01-01') as (col0,col1,col2);
--返回结果如下。
+------+------+------+
| col0 | col1 | col2 |
+------+------+------+
| A    | 10   | 2015-01-01 |
| B    | 20   | 2016-01-01 |
+------+------+------+
--将a、b、c、d排列为两行，源表如果有多行，按行执行stack的分行处理。
select stack(2,a,b,c,d) as (col,value)
from values
    (1,1,2,3,4),
    (2,5,6,7,8),
    (3,9,10,11,12),
    (4,13,14,15,null)
as t(key,a,b,c,d);
--返回结果如下。
+------+-------+
| col  | value |
+------+-------+
| 1    | 2     |
| 3    | 4     |
| 5    | 6     |
| 7    | 8     |
| 9    | 10    |
| 11   | 12    |
| 13   | 14    |
| 15   | NULL  |
+------+-------+
--配合lateral view使用。
select tf.* from (select 0) t lateral view stack(2,'A',10,date '2015-01-01','B',20, date '2016-01-01') tf as col0,col1,col2;
--返回结果如下。
+------+------+------+
| col0 | col1 | col2 |
+------+------+------+
| A    | 10   | 2015-01-01 |
| B    | 20   | 2016-01-01 |
+------+------+------+
```

STR_TO_MAP

- 命令格式

```
str_to_map([string <mapDupKeyPolicy>,] <text> [, <delimiter1> [, <delimiter2>]])
```

- 命令说明

使用delimiter1将text分割成Key-Value对，然后使用delimiter2分割每个Key-Value对的Key和Value。

- 参数说明
  - mapDupKeyPolicy：可选。STRING类型。指定出现重复Key时的处理方式。取值范围如下：
    - exception：如果出现重复的Key，返回报错。
    - last_win：如果出现重复的Key，后边的值将覆盖前边的值。

    该参数也可以在Session级别通过 `odps.sql.map.key.dedup.policy` 参数进行设置，例如 `set odps.sql.map.key.dedup.policy=exception;` ，不设置时该参数默认值为last_win。

    > **说明** MaxCompute的行为实现优先以函数中mapDupKeyPolicy的取值为准，当函数未配置mapDupKeyPolicy时，以 `odps.sql.map.key.dedup.policy` 参数的取值为准。

  - text：必填。STRING类型，指被分割的字符串。
  - delimiter1：可选。STRING类型，分隔符，不指定时默认为英文逗号（ `,` ）。
  - delimiter2：可选。STRING类型，分隔符，不指定时默认为等于号（ `=` ）。

    > **说明** 当分隔符是正则表达式或特殊字符时，要在前面加两个反斜线（\\）来做转义。特殊字符包括冒号（:）、英文句点（.）、问号（?）、加号（+）或星号（*）。

- 返回值说明

  返回值类型为 `map<string, string>` 。返回值是text被delimiter1和delimiter2分割后的结果。

- 示例

```
--返回{test1:1, test2:2}。
select str_to_map('test1&1-test2&2','-','&');
--返回{test1:1, test2:2}。
select str_to_map("test1.1,test2.2", ",", "\\.");
--返回{test1:1, test2:3}。
select str_to_map("test1.1,test2.2,test2.3", ",", "\\.");
```

TABLE_EXISTS

- 命令格式

```
boolean table_exists(string <table_name>)
```

- 命令说明

  查询指定的表是否存在。

- 参数说明

  table_name：必填。表名称。STRING类型。表名称中可以指定项目名称（例如 `my_proj.my_table` ）。如果不指定项目名称则默认为当前项目。

- 返回值说明

  返回BOOLEAN类型。如果指定的表存在返回True，否则返回False。

- 示例

```
--在select列表中使用。
select if(table_exists('abd'), col1, col2) from src;
```

TRANS_ARRAY

- 使用限制
  - 所有作为 `key` 的列必须位于在前面，而要转置的列必须放在后面。
  - 在一个 `select` 中只能有一个UDTF，不可以再出现其他的列。
  - 不可以与 `group by` 、 `cluster by` 、 `distribute by` 、 `sort by` 一起使用。

- 命令格式

```
trans_array (<num_keys>, <separator>, <key1>,<key2>,…,<col1>,<col2>,<col3>) as (<key1>,<key2>,...,<col1>, <col2>)
```

- 命令说明

  将一行数据转为多行的UDTF，将列中存储的以固定分隔符格式分隔的数组转为多行。

- 参数说明
  - num_keys：必填。 BIGINT类型常量，值必须 `>=0` 。在转为多行时作为转置 `key` 的列的个数。
  - separator：必填。STRING类型常量，用于将字符串拆分成多个元素的分隔符。为空时返回报错。

- keys：必填。转置时作为 `key` 的列， 个数由num_keys指定。如果num_keys指定所有的列都作为 `key` （即num_keys等于所有列的个数），则只返回一行。
- cols： 必填。要转为行的数组， `keys` 之后的所有列视为要转置的数组，必须为STRING类型，存储的内容是字符串格式的数组，例如 `Hangzhou;Beijing;shanghai` ，是以分号（ `;` ）分隔的数组。

- 返回值说明

  返回转置后的行，新的列名由 `as` 指定。作为 `key` 的列类型保持不变，其余所有的列是STRING类型。拆分成的行数以个数多的数组为准，不足的补NULL。

- 示例
  - 示例1：例如表 `t_table` 中的数据如下。

```
+----------+----------+------------+
| login_id | login_ip | login_time |
+----------+----------+------------+
| wangwangA | 192.168.0.1,192.168.0.2 | 20120101010000,20120102010000 |
| wangwangB | 192.168.45.10,192.168.67.22,192.168.6.3 | 20120111010000,20120112010000,20120223080000 |
+----------+----------+------------+
--执行SQL。
select trans_array(1, ",", login_id, login_ip, login_time) as (login_id,login_ip,login_time) from t_table;
--返回结果如下。
+----------+----------+------------+
| login_id | login_ip | login_time |
+----------+----------+------------+
| wangwangB | 192.168.45.10 | 20120111010000 |
| wangwangB | 192.168.67.22 | 20120112010000 |
| wangwangB | 192.168.6.3 | 20120223080000 |
| wangwangA | 192.168.0.1 | 20120101010000 |
| wangwangA | 192.168.0.2 | 20120102010000 |
+----------+----------+------------+
--如果表中的数据如下所示。
Login_id LOGIN_IP LOGIN_TIME
wangwangA 192.168.0.1,192.168.0.2 20120101010000
--会对数组中不足的数据补NULL。
Login_id Login_ip Login_time
wangwangA 192.168.0.1 20120101010000
wangwangA 192.168.0.2 NULL
```

  - 示例2：例如表mf_fun_array_test_t中的数据如下。

```
+------------+------------+------------+------------+
| id         | name       | login_ip   | login_time |
+------------+------------+------------+------------+
| 1          | Tom        | 192.168.100.1,192.168.100.2 | 20211101010101,20211101010102 |
| 2          | Jerry      | 192.168.100.3,192.168.100.4 | 20211101010103,20211101010104 |
+------------+------------+------------+------------+
--用两个key，id和name进行转数组，执行SQL。
select trans_array(2, ",", Id,Name, login_ip, login_time) as (Id,Name,login_ip,login_time) from mf_fun_array_test_t;
--返回结果如下，已经对key，id和name进行分组拆解。
+------------+------------+------------+------------+
| id         | name       | login_ip   | login_time |
+------------+------------+------------+------------+
| 1          | Tom        | 192.168.100.1 | 20211101010101 |
| 1          | Tom        | 192.168.100.2 | 20211101010102 |
| 2          | Jerry      | 192.168.100.3 | 20211101010103 |
| 2          | Jerry      | 192.168.100.4 | 20211101010104 |
+------------+------------+------------+------------+
```

TRANS_COLS

- 使用限制
  - 所有作为 `key` 的列必须处在前面，而要转置的列必须放在后面。
  - 在一个 `select` 中只能有一个UDTF，不可以再出现其他的列。

- 命令格式

```
trans_cols (<num_keys>, <key1>,<key2>,…,<col1>, <col2>,<col3>) as (<idx>, <key1>,<key2>,…,<col1>, <col2>)
```

- 命令说明

  将一行数据转为多行数据的UDTF，将不同的列拆分为不同的行。

- 参数说明
  - num_keys：必填。BIGINT类型常量，值必须 `>=0` 。在转为多行时作为转置key的列的个数。

- keys：必填。转置时作为key的列， 个数由num_keys指定。如果num_keys指定所有的列都作为key（即num_keys等于所有列的个数），则只返回一行。
- idx：必填。转换后的行号。
- cols：必填。 要转为行的列。

- 返回值说明

  返回转置后的行，新的列名由 `as` 指定。输出的第一列是转置的下标，下标从1开始。作为key的列类型保持不变，其余所有的列与原来的数据类型一致。

- 示例

  例如表 `t_table` 中的数据如下。

```
+----------+----------+------------+
| Login_id | Login_ip1 | Login_ip2 |
+----------+----------+------------+
| wangwangA | 192.168.0.1 | 192.168.0.2 |
+----------+----------+------------+
--执行SQL。
select trans_cols(1, login_id, login_ip1, login_ip2) as (idx, login_id, login_ip) from t_table;
--返回结果如下。
idx    login_id    login_ip
1    wangwangA    192.168.0.1
2    wangwangA    192.168.0.2
```

UNBASE64

- 命令格式

```
binary unbase64(string <str>)
```

- 命令说明

  将BASE64编码格式字符串str转换为二进制表示格式。

- 参数说明

  str：必填。STRING类型。待转换BASE64编码格式字符串。

- 返回值说明

  返回BINARY类型。输入参数为NULL时，返回结果为NULL。

- 示例
  - 示例1：将字符串 `YWxpYmFiYQ==` 转换为二进制表示值。命令示例如下：

```
--返回alibaba。
select unbase64('YWxpYmFiYQ==');
```

  - 示例2：输入参数为NULL。命令示例如下：

```
--返回NULL。
select unbase64(null);
```

UNIQUE_ID

- 命令格式

```
string unique_id()
```

- 命令说明

  返回一个随机的唯一ID，格式示例为 `29347a88-1e57-41ae-bb68-a9edbdd9****_1` 。该函数的运行效率高于UUID，且返回的ID较长，相较于UUID多一个下划线（_）和一个数字编号，例如 `_1` 。

UUID

- 命令格式

```
string uuid()
```

- 命令说明

  返回一个随机ID，格式示例为 `29347a88-1e57-41ae-bb68-a9edbdd9****` 。

> **说明** UUID返回的是一个随机的全局ID，其重复的概率很小。

3.9.10. 内建函数典型案例

3.9.10.1. ROUND函数精度问题案例

ROUND函数支持根据您指定的小数位数对输入的数值进行四舍五入。本文为您介绍ROUND函数在使用过程存在的精度问题，并提供解决措施及示例供参考。

案例

在SQL代码中使用ROUND函数进行四舍五入时，返回结果与预期的四舍五入结果有偏差。命令示例如下。

```
select round(0.25375,4);
```

按照函数的逻辑实现，返回结果应该为0.2538，但实际返回结果为0.2537，未进位。

产生原因

对于手工输入的数值，MaxCompute会自动将数值转换为DOUBLE类型，因为转换DOUBLE类型数值时无法保证精度，所以ROUND函数处理结果会产生偏差。案例中的0.25375的DOUBLE类型表示结果为0.2536999999...，所以未进位。

当通过ROUND函数处理表中的数据时，会读取到表字段的实际数据类型，不会存在此问题。

解决措施

如果需要保证手工输入数据的精度，可以通过CAST函数将手工输入的数据转换为DECIMAL类型，DECIMAL类型不会存在精度问题。命令示例如下。

```
select round(cast(0.25375 as decimal), 4);
```

返回结果为0.2538。

3.9.10.2. 实现GROUP_CONCAT函数能力案例

GROUP_CONCAT为MySQL提供的函数，用于实现在GROUP BY产生的每一个分组中将指定字段的值连接起来。本文为您介绍如何通过MaxCompute的内建函数实现GROUP_CONCAT函数的能力。

案例

假设，现有一张名为price_total的表，表中包含商品名称（name）、价格（price）和商家ID（saleid）三列，具体数据如下。

```
+--------+------------+------------+
| name   | price      | saleid     |
+--------+------------+------------+
| bag    | 50         | 1          |
| sugar  | 20         | 3          |
| noodle | 2          | 4          |
| potato | 5          | 6          |
| bag    | 100        | 2          |
| sugar  | 10         | 4          |
| potato | 4          | 3          |
| sugar  | 50         | 7          |
| noodle | 2          | 5          |
| noodle | 5          | 1          |
+--------+------------+------------+
```

对于该表中的数据，现需要对所有商品按照商品名称（name）进行分组，实现如下需求：

- 需求一：将同组的价格（price）进行合并，不去重。MySQL的GROUP_CONCAT函数实现代码如下。

  ```
  select name, group_concat(price) from price_total group by name;
  ```

- 需求二：将同组的价格（price）进行合并，并去重。MySQL的GROUP_CONCAT函数实现代码如下。

  ```
  select name, group_concat(distinct price) from price_total group by name;
  ```

- 需求三：将同组的价格（price）进行合并且排序，不去重。MySQL的GROUP_CONCAT函数实现代码如下。

  ```
  select name, group_concat(price order by price desc) from price_total group by name;
  ```

- 需求四：将同组的价格（price）进行合并且排序，并去重。MySQL的GROUP_CONCAT函数实现代码如下。

  ```
  select name, group_concat(distinct price order by price desc) from price_total group by name;
  ```

- 需求五：将同组的价格（price）、商家ID（saleid）进行合并。MySQL的GROUP_CONCAT函数实现代码如下。

  ```
  select name, group_concat(concat_ws(':', price, saleid)) from price_total group by name;
  ```

解决措施

在MaxCompute中，未提供GROUP_CONCAT函数，但您可以通过WM_CONCAT函数实现案例中的需求。

但需要注意的是WM_CONCAT函数并不等价于GROUP_CONCAT函数，二者的能力支持情况如下。

函数能力	WM_CONCAT	GROUP_CONCAT	说明
去重（DISTINCT）	✅	✅	无
分隔符	✅	✅	分隔符用于连接待合并的值。二者的使用区别如下： • WM_CONCAT不能省略分隔符。 • GROUP_CONCAT可以省略分隔符，省略时将英文逗号（,）做为分隔符。
排序	❌	✅	WM_CONCAT暂不支持对合并的值进行排序，如果涉及排序需求，您可以通过如下方式解决： • 先对待合并字段进行排序，然后再合并。 • 自定义UDF。更多自定义UDF信息，请参见MaxCompute UDF。
合并多列	✅	✅	WM_CONCAT和GROUP_CONCAT函数都属于合并行的函数，可以与合并列的函数CONCAT或CONCAT_WS配合使用实现分组合并多列值。

基于上表所列区别，通过WM_CONCAT函数可实现案例中的如下需求：

- 需求一：将同组的价格（price）进行合并，不去重。

```
select name, wm_concat(',', price) as price_new from price_total group by name;
```

返回结果如下。

```
+--------+-----------+
| name   | price_new |
+--------+-----------+
| bag    | 50,100    |
| noodle | 2,2,5     |
| potato | 5,4       |
| sugar  | 20,10,50  |
+--------+-----------+
```

- 需求二：将同组的价格（price）进行合并，并去重。

```
select name, wm_concat(distinct ',', price) as price_new from price_total group by name;
```

返回结果如下。

```
+--------+-----------+
| name   | price_new |
+--------+-----------+
| bag    | 100,50    |
| noodle | 2,5       |
| potato | 4,5       |
| sugar  | 10,20,50  |
+--------+-----------+
```

- 需求三：将同组的价格（price）进行合并且排序，不去重。

```
--解除order by必须带limit的限制。
set odps.sql.validate.orderby.limit=false;
select name, wm_concat(',', price) as price_new from (select name, price from price_total order by name, price desc) group by
name;
```

返回结果如下。

```
+--------+-----------+
| name   | price_new |
+--------+-----------+
| bag    | 100,50    |
| noodle | 5,2,2     |
| potato | 5,4       |
| sugar  | 50,20,10  |
+--------+-----------+
```

- 需求四：将同组的价格（price）进行合并且排序，并去重。

```
--解除order by必须带limit的限制。
set odps.sql.validate.orderby.limit=false;
select name, wm_concat(',', price) as price_new from (select distinct name, price from price_total order by name, price asc) g
roup by name;
```

返回结果如下。

```
+--------+-----------+
| name   | price_new |
+--------+-----------+
| bag    | 50,100    |
| noodle | 2,5       |
| potato | 4,5       |
| sugar  | 10,20,50  |
+--------+-----------+
```

- 需求五：将同组的价格（price）、商家ID（saleid）进行合并。

```
select name, wm_concat(',', concat_ws(':',price,saleid)) as price_new from price_total group by name;
--等效于如下语句。
select name, wm_concat(',', concat(price,':',saleid)) from price_total group by name;
```

返回结果如下。

```
+--------+-----------------+
| name   | price_new       |
+--------+-----------------+
| bag    | 50:1,100:2      |
| noodle | 2:4,2:5,5:1     |
| potato | 5:6,4:3         |
| sugar  | 20:3,10:4,50:7  |
+--------+-----------------+
```

3.9.11. 内建函数常见错误码

本文为您介绍使用MaxCompute内建函数过程中常见的错误码，帮助您了解错误码的产生原因，并提供对应的解决措施。

常见内建函数错误码如下：

- 报错信息：Semantic analysis exception - X type is not enabled in current mode
- 报错信息：Invalid number of arguments - function Y needs m parameters, actually have n
- 报错信息：Invalid argument type- invalid type X of argument m for function Y
- 报错信息：Semantic analysis exception - function or view Y cannot be resolved

报错信息：Semantic analysis exception - X type is not enabled in current mode

- 问题现象

  调用内建函数时，详细报错信息示例如下。

```
FAILED: ODPS-0130071:[1,27] Semantic analysis exception - TIMESTAMP type is not enabled in current mode. Please set odps.sql.t
ype.system.odps2=true to use it.
```

- 产生原因

  SQL语句中使用的内建函数涉及2.0数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP或BINARY），但项目未开启2.0数据类型开关，导致MaxCompute处理失败。

- 解决措施

  您可以在MaxCompute客户端，通过如下方式之一，开启2.0数据类型开关：

  - Session级别：您需要在SQL语句前加上 `set odps.sql.type.system.odps2=true;` 语句，与SQL语句一起提交执行。
  - Project级别：项目所有者（Project Owner）可以根据需要对Project进行设置，等待10~15分钟后才会生效。命令如下。

    ```
    setproject odps.sql.type.system.odps2=true;
    ```

    命令生效后，重新执行SQL语句即可。

    更多 `setproject` 信息，请参见项目空间操作。更多开启Project级别2.0数据类型开关的注意事项，请参见数据类型版本说明。

报错信息：Invalid number of arguments - function Y needs m parameters, actually have n

- 问题现象

  调用内建函数时，详细报错信息示例如下。

```
FAILED: ODPS-0130221:[1,8] Invalid number of arguments - function from_utc_timestamp needs 2 parameters, actually have 1
```

- 产生原因

  SQL语句中使用的内建函数的实际输入参数个数不满足函数语法要求，导致MaxCompute处理失败。

- 解决措施

  检查内建函数的输入参数数量，补足或减少参数，确保满足函数语法要求后，重新执行SQL语句即可。

报错信息：Invalid argument type- invalid type X of argument m for function Y

- 问题现象

  调用内建函数时，详细报错信息示例如下。

```
FAILED: ODPS-0130121:[1,18] Invalid argument type - invalid type STRING of argument 1 for function all_match, expect ARRAY<T>
```

- 产生原因

  SQL语句中使用的内建函数的实际输入参数类型不满足函数语法要求，导致MaxCompute处理失败。

- 解决措施

  根据函数语法要求修改输入参数，确保数据类型满足函数语法要求后，重新执行SQL语句即可。

报错信息：Semantic analysis exception - function or view Y cannot be resolved

- 问题现象

  调用内建函数时，详细报错信息示例如下。

```
FAILED: ODPS-0130071:[1,8] Semantic analysis exception - function or view 'row_number' cannot be resolved
```

- 产生原因

  SQL语句中使用的内建函数名称有误或者未携带参数，导致MaxCompute处理失败。

- 解决措施

  检查函数名称和参数名称，并根据函数语法要求添加输入参数，确保名称及语法格式全部正确后，重新执行SQL语句即可。

3.9.12. 内建函数常见问题

本文为您介绍MaxCompute内建函数的常见问题。

问题类别	常见问题
日期函数	• MaxCompute是否支持将2010/1/3格式转换为2010-01-03？ • 如何将UNIX时间转为日期值？ • 如何获取系统当前时间？ • 使用YEAR、QUARTER、MONTH和DAY日期函数时，报错cannot be resolved，如何解决？ • 执行TO_DATE函数时，报错没有分钟部分，如何解决？
数学函数	使用ROUND函数对DOUBLE类型数据四舍五入，为何结果存在偏差？
窗口函数	MaxCompute可以通过哪个函数设置自增序列？
聚合函数	如何连接相同字段？
字符串函数	• MaxCompute是否支持MD5函数？ • 如何实现对固定长度字符串，不足部分前面补零？ • MySQL支持的SUBSTRING_INDEX函数在MaxCompute中支持吗？ • REGEXP_COUNT函数的参数pattern是否支持嵌入查询语句？ • MaxCompute是否支持Oracle的to_char(数据, FM9999.00)用法？
复杂类型函数	• 如何将符合某条件的字段都拼到一个JSON中？ • 如何将一个JSON字符串中的每个Key当做一个字段？ • 如何将JSON字符串转换为ARRAY格式？
其他函数	• MySQL的IFNULL函数对应MaxCompute的哪个函数？ • 如何实现一行转多行？ • 使用COALESCE函数时，报错Expression not in GROUP BY key，如何解决？ • 在使用IFNULL函数时，报错Invalid function，如何解决？

问题类别	常见问题
隐式转换	使用MaxCompute内建函数时，隐式类型转换报错，如何解决？

MaxCompute是否支持将2010/1/3格式转换为2010-01-03？

如果日期为2010/01/03格式，可以通过 `to_char(to_date('2010/01/03', 'yyyy/mm/dd'), 'yyyy-mm-dd')` 函数进行转换，请参见TO_DATE和TO_CHAR。

如果日期为2010/1/3格式，您需要自行编写UDF函数实现日期转换。编写UDF指导请参见MaxCompute UDF概述。

如何将UNIX时间转为日期值？

您可以使用FROM_UNIXTIME函数将数字型的UNIX时间转为日期值，请参见FROM_UNIXTIME。

如何获取系统当前时间？

您可以使用GETDATE函数获取系统当前时间，请参见GETDATE。

使用YEAR、QUARTER、MONTH和DAY日期函数时，报错cannot be resolved，如何解决？

- 问题现象

  使用YEAR、QUARTER、MONTH和DAY日期函数时，返回报错如下。

  ```
  FAILED: ODPS-0130071:[1,8] Semantic analysis exception - function or view 'year' cannot be resolved
  ```

- 产生原因

  YEAR、QUARTER、MONTH和DAY为MaxCompute 2.0扩展函数，需要打开2.0数据类型开关。

- 解决措施

  您需要在SQL语句前增加set语句 `set odps.sql.type.system.odps2 = true;` ，打开2.0新数据类型开关。

执行TO_DATE函数时，报错没有分钟部分，如何解决？

- 问题现象

  执行SQL语句 `to_date('2016-07-18 18:18:18', 'yyyy-MM-dd HH:mm:ss')` 时，返回报错如下。

  ```
  FAILED: ODPS-0121095:Invalid arguments - format string has second part, but doesn't have minute part : yyyy-MM-dd HH:mm:ss
  ```

- 产生原因

  TO_DATE函数的第二个参数格式有误。 `mm` 和 `MM` 都表示月份，分钟需要使用 `mi` 。

- 解决措施

  修改SQL语句为 `to_date('2016-07-18 18:18:18', 'yyyy-MM-dd HH:mi:ss')` 。

使用ROUND函数对DOUBLE类型数据四舍五入，为何结果存在偏差？

使用 `round` 函数对DOUBLE类型的数据进行四舍五入，发现4.515四舍五入结果为4.51。SQL示例如下。

```
select round(4.515, 2),round(125.315, 2);
```

DOUBLE类型是8字节双精度浮点数，存在一定的精度差。 `4.515` 的DOUBLE类型表示结果为 `4.514999999...` ，因此四舍五入时被计算为4.51。

MaxCompute可以通过哪个函数设置自增序列？

您可以使用ROW_NUMBER函数设置自增序列，请参见ROW_NUMBER。

如何连接相同字段？

您可以使用WM_CONCAT函数连接相同字段，请参见WM_CONCAT。

MaxCompute是否支持MD5函数？

支持，请参见MD5。

如何实现对固定长度字符串，不足部分前面补零？

您可以使用LPAD函数，请参见LPAD。

MySQL支持的SUBSTRING_INDEX函数在MaxCompute中支持吗？

支持，请参见SUBSTRING_INDEX。

REGEXP_COUNT函数的参数pattern是否支持嵌入查询语句？

不支持，使用方法请参见REGEXP_COUNT。

MaxCompute是否支持Oracle的 `to_char(数据, FM9999.00)` 用法？

不支持。如果只涉及调整数字显示格式，您可以使用FORMAT_NUMBER函数实现。命令示例如下。

```
--返回12,332.123。
select format_number(12332.123456, '#,###,###,###.###');
```

在使用IFNULL函数时，报错Invalid function，如何解决？

- 问题现象

  在如下SQL语句中使用IFNULL函数。

  ```
  select a.id as id > , ifnull(concat('phs\xxx', a.insy, '\xxxb\xxx', ifnull()))
  ```

  返回报错如下。

  ```
  Semantic analysis exception - Invalid function : line 1:41 'ifnull'
  ```

- 产生原因

  MaxCompute没有提供 `ifnull` 函数导致报错。

- 解决措施

  您需要使用 `case when` 表达式或 `coalesce` 命令，请参见CASE WHEN表达式或COALESCE。

如何将符合某条件的字段都拼到一个JSON中？

MaxCompute SQL通过过滤条件，例如 `like` 等，查询得到满足指定条件的数据，然后将这些数据通过ARRAY或MAP函数构造为MAP、ARRAY复杂类型数据，然后再用TO_JSON函数聚合。

如何将一个JSON字符串中的每个Key当做一个字段？

您可以使用GET_JSON_OBJECT函数来提取字段。

如何将JSON字符串转换为ARRAY格式？

您可以使用FROM_JSON函数进行转换，例如 `select from_json(<col_name>, "array<bigint>");` 。

MySQL的IFNULL函数对应MaxCompute的哪个函数？

MySQL的IFNULL函数对应MaxCompute的NVL函数。MaxCompute内建函数与Hive、MySQL及Oracle函数的对照关系请参见与Hive、MySQL、Oracle内建函数对照表。

如何实现一行转多行？

您可以使用TRANS_COLS函数来将一行数据转为多行数据。

使用COALESCE函数时，报错Expression not in GROUP BY key，如何解决？

- 问题现象

  使用COALESCE函数时，只要超过一个Expression，就会返回如下报错。

  ```
  FAILED: ODPS-0130071:Semantic analysis exception - Expression not in GROUP BY key : line 8:9 "$.table"
  ```

  报错SQL如下。

  ```
  select
  md5(concat(aid,bid)) as id
  ,aid
  , bid
  , sum(amountdue) as amountdue
  , coalesce(
  sum(regexp_count(get_json_object(extended_x, '$.table.tableParties'), '{')),
  decode(get_json_object(extended_x, '$.table'), null, 0, 1)
  ) as tableparty
  , decode(sum(headcount),null,0,sum(headcount) ) as headcount
  , 'a' as pt
  from e_orders
  where pt='20170425'
  group by aid, bid;
  ```

- 产生原因

  GROUP BY后面缺少分组字段，因此报错。

- 解决措施

  如下表达式的返回值实际上是字段，需要把整个表达式写在GROUP BY后面。

```
coalesce(
sum(regexp_count(get_json_object(extended_x, '$.table.tableParties'), '{')),
decode(get_json_object(extended_x, '$.table'), null, 0, 1)
) as tableparty
, decode(sum(headcount),null,0,sum(headcount) ) as headcount
```

使用MaxCompute内建函数时，隐式类型转换报错，如何解决？

当MaxCompute项目打开2.0新数据类型开关（`odps.sql.type.system.odps2=true`）时，如下隐式类型转换会被禁用，存在精度损失或报错的风险：

- STRING->BIGINT
- STRING->DATETIME
- DOUBLE->BIGINT
- DECIMAL->DOUBLE
- DECIMAL->BIGINT

此时，您可以通过CAST函数执行强制转换解决该问题或者关闭2.0新数据类型开关（`odps.sql.type.system.odps2=false`）。

3.10. MaxCompute UDF

3.10.1. MaxCompute UDF概述

MaxCompute UDF（User Defined Function）即用户自定义函数。MaxCompute产品已经提供多种类型内建函数来满足您的业务需求，当内建函数无法支撑业务实现时，您可以自行编写代码逻辑来创建自定义函数满足多样化需求。本文为您介绍MaxCompute支持的自定义函数类型、应用场景、开发流程及使用说明等信息。

背景信息

广义的UDF定义是自定义标量函数（UDF）、自定义表值函数（UDTF）及自定义聚合函数（UDAF）三种类型的自定义函数的集合。狭义的UDF仅代表用户自定义标量函数。MaxCompute UDF支持的自定义函数类型如下。

自定义函数类型	名称	应用场景
UDF	User Defined Scalar Function。用户自定义标量函数。	适用于一进一出业务场景。即其输入与输出是一对一的关系，读入一行数据，输出一个值。
UDTF	User Defined Table Valued Function。用户自定义表值函数，又称表格UDF。	适用于一进多出业务场景。即其输入与输出是一对多的关系，读入一行数据，输出多个值可视为一张表。
UDAF	User Defined Aggregation Function。用户自定义聚合函数。	适用于多进一出业务场景。即其输入与输出是多对一的关系，将多条输入记录聚合成一个输出值。

除上述自定义函数外，MaxCompute还提供如下针对特殊场景的能力支撑。

自定义函数类型	应用场景
代码嵌入式UDF	当您需要简化MaxCompute UDF操作步骤，并希望能直接查看代码实现逻辑时，您可以直接将Java或Python代码嵌入SQL脚本。
SQL语言定义函数	当您的代码中存在很多相似部分时，可以通过SQL自定义函数实现，提高代码复用率的同时还可以简化操作流程。
开源地理空间UDF	支持在MaxCompute中使用Hive地理空间函数分析空间数据。

使用限制

不支持通过自定义函数访问外网。如果您需要通过自定义函数访问外网，请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

注意事项

使用自定义函数时，您需要注意：

- 在性能上，自定义函数的性能低于内建函数，建议您优先使用内建函数实现相同逻辑的业务需求。
- 在SQL语句中使用自定义函数时，如果计算的数据量过大并且存在倾斜，会导致作业占用的内存超出默认分配的内存。此时，您可以在Session级别设置`set odps.sql.udf.joiner.jvm.memory=xxxx;`属性来解决此问题。更多自定义函数常见问题，请参见MaxCompute UDF常见问题。
- 当自定义函数的名称与内建函数的名称相同时，自定义函数会覆盖同名的内建函数。例如，MaxCompute中存在一个名称为CONCAT的自定义函数，则系统默认会调用自定义的CONCAT，不会调用内建函数CONCAT。如果您希望调用内建函数，需要在内建函数前增加`::`符号，例如`select ::concat('ab', 'c');`。

开发流程

自定义函数（UDF\UDTF\UDAF）的开发流程如下。代码嵌入式UDF、SQL语言定义函数、开源地理空间UDF的开发流程与此不同，请以对应文档描述为准。

- 当您使用Java代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档
①	可选	当您使用Maven编写代码时，需要先在Pom文件中添加代码相关SDK依赖，确保后续编写的代码可编译成功。例如开发自定义函数需要添加的SDK依赖为： <dependency> <groupId>com.aliyun.odps</groupId> <artifactId>odps-sdk-udf</artifactId> <version>0.29.10-public</version> </dependency> 您可以从Maven库中搜索 odps-sdk-udf ，获取SDK依赖版本信息。	IntelliJ IDEA（Maven）	无
②	必选	根据业务需求，编写自定义函数代码。	IntelliJ IDEA（Maven）、MaxCompute Studio	开发Java UDF
③	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		
④	必选	调试自定义函数代码，确保本地运行成功后打包为JAR包。		
⑤	必选	将JAR包作为资源上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	○ MaxCompute客户端 ■ 添加资源 ■ 注册函数 ○ MaxCompute Studio 打包、上传及注册 ○ DataWorks ■ 创建JAR资源 ■ 注册MaxCompute函数
⑥	必选	基于上传的JAR包资源创建自定义函数。		
⑦	可选	在查询数据代码中调用自定义函数。		无

- 当您使用Python代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档
①	必选	根据业务需求，编写自定义函数代码。	MaxCompute Studio	开发Python UDF
②	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		
③	必选	将Python文件或依赖的资源（例如文件资源、表资源、第三方包）上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	○ MaxCompute客户端 ■ 添加资源 ■ 注册函数 ○ MaxCompute Studio 上传及注册 ○ DataWorks 创建Python资源并注册函数
④	必选	基于上传的Python文件或依赖的资源创建自定义函数。		
⑤	可选	在查询数据代码中调用自定义函数。		无

使用说明

自定义函数的使用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例： `select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

MaxCompute SDK

MaxCompute提供的SDK信息如下。各SDK中包含的Package，以及Package中的Class详细信息，请参见MaxCompute SDK。

SDK名称	描述
odps-sdk-core	提供操作MaxCompute基本资源的类。
odps-sdk-commons	Java Util封装。
odps-sdk-udf	UDF功能的主体接口。
odps-sdk-mapred	MapReduce API。
odps-sdk-graph	Graph API。

3.10.2. UDF

3.10.2.1. UDF概述

MaxCompute支持您通过Java、Python语言编写代码创建UDF，扩展MaxCompute的函数能力，满足个性化业务需求。本文为您介绍UDF类型、使用限制、注意事项、开发流程及使用说明等信息。

背景信息

UDF适用于一进一出业务场景。即其输入与输出是一对一的关系，读入一行数据，输出一个值。MaxCompute支持您通过如下两种代码语言编写UDF。

UDF类型	说明
Java UDF	使用Java语言编写UDF逻辑实现代码。更多信息，请参见Java UDF。
Python UDF	使用Python语言编写UDF逻辑实现代码，支持Python 2和Python 3： • Python 2 UDF：Python语言版本为2.7。更多信息，请参见Python 2 UDF。 • Python 3 UDF：Python语言版本为CPython-3.7.3。更多信息，请参见Python 3 UDF。

使用限制

不支持通过自定义函数访问外网。如果您需要通过自定义函数访问外网，请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

注意事项

使用自定义函数时，您需要注意：

- 在性能上，自定义函数的性能低于内建函数，建议您优先使用内建函数实现相同逻辑的业务需求。
- 在SQL语句中使用自定义函数时，如果计算的数据量过大并且存在倾斜，会导致作业占用的内存超出默认分配的内存。此时，您可以在Session级别设置 `set odps.sql.udf.joiner.jvm.memory=xxxx;` 属性来解决此问题。更多自定义函数常见问题，请参见MaxCompute UDF常见问题。
- 当自定义函数的名称与内建函数的名称相同时，自定义函数会覆盖同名的内建函数。例如，MaxCompute中存在一个名称为CONCAT的自定义函数，则系统默认会调用自定义的CONCAT，不会调用内建函数CONCAT。如果您希望调用内建函数，需要在内建函数前增加 `::` 符号，例如 `select ::concat('ab', 'c');` 。

开发流程

UDF的开发流程如下。

- 当您使用Java代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档

序号	可选/必选步骤	功能介绍	工具平台	相关文档

①	可选	当您使用Maven编写代码时，需要先在Pom文件中添加代码相关SDK依赖，确保后续编写的代码可编译成功。例如开发自定义函数需要添加的SDK依赖为：
		`<dependency>`
		`<groupId>com.aliyun.odps</groupId>`
		`<artifactId>odps-sdk-udf</artifactId>`
		`<version>0.29.10-public</version>`
		`</dependency>`
		您可以从Maven库中搜索 `odps-sdk-udf` ，获取SDK依赖版本信息。
			IntelliJ IDEA（Maven）	无

②	必选	根据业务需求，编写自定义函数代码。	IntelliJ IDEA（Maven）、MaxCompute Studio	开发Java UDF
③	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		
④	必选	调试自定义函数代码，确保本地运行成功后打包为JAR包。		
⑤	必选	将JAR包作为资源上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	○ MaxCompute客户端 ■ 添加资源 ■ 注册函数 ○ MaxCompute Studio 打包、上传及注册 ○ DataWorks ■ 创建JAR资源 ■ 注册MaxCompute函数
⑥	必选	基于上传的JAR包资源创建自定义函数。		
⑦	可选	在查询数据代码中调用自定义函数。		无

- 当您使用Python代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档
①	必选	根据业务需求，编写自定义函数代码。	MaxCompute Studio	开发Python UDF
②	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		
③	必选	将Python文件或依赖的资源（例如文件资源、表资源、第三方包）上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	○ MaxCompute客户端 ■ 添加资源 ■ 注册函数 ○ MaxCompute Studio 上传及注册 ○ DataWorks 创建Python资源并注册函数
④	必选	基于上传的Python文件或依赖的资源创建自定义函数。		
⑤	可选	在查询数据代码中调用自定义函数。		无

使用说明

自定义函数的使用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例： `select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

3.10.2.2. Java UDF

本文为您介绍如何通过Java语言编写UDF。

UDF代码结构

您可以通过IntelliJ IDEA（Maven）或MaxCompute Studio工具使用Java语言编写UDF代码，代码中需要包含如下信息：

- Java包（Package）：可选。

  您可以将定义的Java类打包，为后续查找和使用类提供方便。

- 继承UDF类：必选。

  必需携带的UDF类为 `com.aliyun.odps.udf.UDF` 。当您需要使用其他UDF类或者需要用到复杂数据类型时，请根据MaxCompute SDK添加需要的类。例如STRUCT数据类型对应的UDF类为 `com.aliyun.odps.data.Struct` 。

- `@Resolve` 注解：可选。

  格式为 `@Resolve(<signature>)` ， `signature` 用于定义函数的输入参数和返回值的数据类型。当您需要在UDF中使用STRUCT数据类型时，无法基于 `com.aliyun.odps.data.Struct` 反射分析得到Field Name和Field Type，所以需要用 `@Resolve` 注解来辅助获取。即如果您需要在UDF中使用STRUCT，请在UDF Class中加上 `@Resolve` 注解，注解只会影响参数或返回值中包含com.aliyun.odps.data.Struct的重载。例如 `@Resolve("struct<a:string>,string->string")` 。详细使用示例，请参见复杂数据类型示例。

- 自定义Java类：必选。

  UDF代码的组织单位，定义了实现业务需求的变量及方法。

- `evaluate` 方法：必选。

  非静态的Public方法，位于自定义的Java类中。 `evaluate` 方法的输入参数和返回值的数据类型将作为SQL语句中UDF的函数签名Signature（定义UDF的输入与输出数据类型）。

  您可以在UDF中实现多个 `evaluate` 方法，在调用UDF时，MaxCompute会依据UDF调用的参数类型匹配正确的 `evaluate` 方法。

  编写Java UDF时可以使用Java Type或Java Writable Type，MaxCompute项目支持处理的数据类型与Java数据类型的详细映射关系，请参见数据类型。

- UDF初始化或结束代码：可选。您可以通过 `void setup(ExecutionContext ctx)` 和 `void close()` 分别实现UDF初始化和结束。 `void setup(ExecutionContext ctx)` 方法会在 `evaluate` 方法前调用且仅会调用一次，可以用来初始化一些计算所需要的资源或类的成员对象。 `void close()` 方法会在 `evaluate` 方法结束后调用，可以用来执行一些清理工作，例如关闭文件。

UDF代码示例如下。

- 使用Java Type类型

```
//将定义的Java类组织在org.alidata.odps.udf.examples包中。
package org.alidata.odps.udf.examples;
//继承UDF类。
import com.aliyun.odps.udf.UDF;
//自定义Java类。
public final class Lower extends UDF {
//evaluate方法。其中：String标识输入参数的数据类型，return标识返回值。
    public String evaluate(String s) {
        if (s == null) {
        return null;
    }
        return s.toLowerCase();
  }
}
```

- 使用Java Writable Type类型

```
//将定义的Java类组织在com.aliyun.odps.udf.example包中。
package com.aliyun.odps.udf.example;
//添加Java Writable Type类型必需的类。
import com.aliyun.odps.io.Text;
//继承UDF类。
import com.aliyun.odps.udf.UDF;
//自定义Java类。
public class MyConcat extends UDF {
  private Text ret = new Text();
//evaluate方法。其中：Text标识输入参数的数据类型，return标识返回值。
  public Text evaluate(Text a, Text b) {
      if (a == null || b == null) {
      return null;
    }
      ret.clear();
      ret.append(a.getBytes(), 0, a.getLength());
      ret.append(b.getBytes(), 0, b.getLength());
      return ret;
  }
}
```

MaxCompute还支持直接使用在其兼容的Hive版本上开发的UDF，请参见兼容Hive UDF。

使用限制

不支持通过自定义函数访问外网。如果您需要通过自定义函数访问外网，请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

注意事项

在编写Java UDF时，您需要注意：

- 不同UDF JAR包中不建议存在类名相同但实现逻辑不一样的类。例如UDF1、UDF2分别对应资源JAR包udf1.jar、udf2.jar，两个JAR包里都包含名称为 `com.aliyun.UserFunction.class` 的类但实现逻辑不一样，当同一条SQL语句中同时调用UDF1和UDF2时，MaxCompute会随机加载其中一个类，此时会导致UDF执行结果不符合预期甚至编译失败。
- Java UDF中输入或返回值的数据类型是对象，数据类型首字母必须大写，例如String。
- SQL中的NULL值通过Java中的NULL表示。Java Primitive Type无法表示SQL中的NULL值，不允许使用。

数据类型

在MaxCompute中不同数据类型版本支持的数据类型不同。从MaxCompute 2.0版本开始，扩展了更多的新数据类型，同时还支持ARRAY、MAP、STRUCT等复杂类型。更多MaxCompute数据类型版本信息，请参见数据类型版本说明。

为确保编写Java UDF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute Type	Java Type	Java Writable Type
TINYINT	java.lang.Byte	ByteWritable
SMALLINT	java.lang.Short	ShortWritable
INT	java.lang.Integer	IntWritable
BIGINT	java.lang.Long	LongWritable
FLOAT	java.lang.Float	FloatWritable
DOUBLE	java.lang.Double	DoubleWritable
DECIMAL	java.math.BigDecimal	BigDecimalWritable
BOOLEAN	java.lang.Boolean	BooleanWritable
STRING	java.lang.String	Text
VARCHAR	com.aliyun.odps.data.Varchar	VarcharWritable
BINARY	com.aliyun.odps.data.Binary	BytesWritable
DATETIME	java.util.Date	DatetimeWritable
TIMESTAMP	java.sql.Timestamp	TimestampWritable
INTERVAL_YEAR_MONTH	不涉及	IntervalYearMonthWritable
INTERVAL_DAY_TIME	不涉及	IntervalDayTimeWritable
ARRAY	java.util.List	不涉及
MAP	java.util.Map	不涉及
STRUCT	com.aliyun.odps.data.Struct	不涉及

当您需要在Java UDF中使用复杂数据类型时，使用示例请参见复杂数据类型示例。

> 说明 当MaxCompute项目采用MaxCompute 2.0数据类型版本时，UDF的输入或返回值才可以使用Java Writable Type。

使用说明

按照开发流程，完成Java UDF开发后，您即可通过MaxCompute SQL调用Java UDF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute Studio完整开发及调用Java UDF的操作，请参见使用示例。

兼容Hive UDF

当MaxCompute项目采用2.0数据类型版本时，支持Hive风格的UDF，您可以直接使用在MaxCompute兼容的Hive版本上开发的Hive UDF。

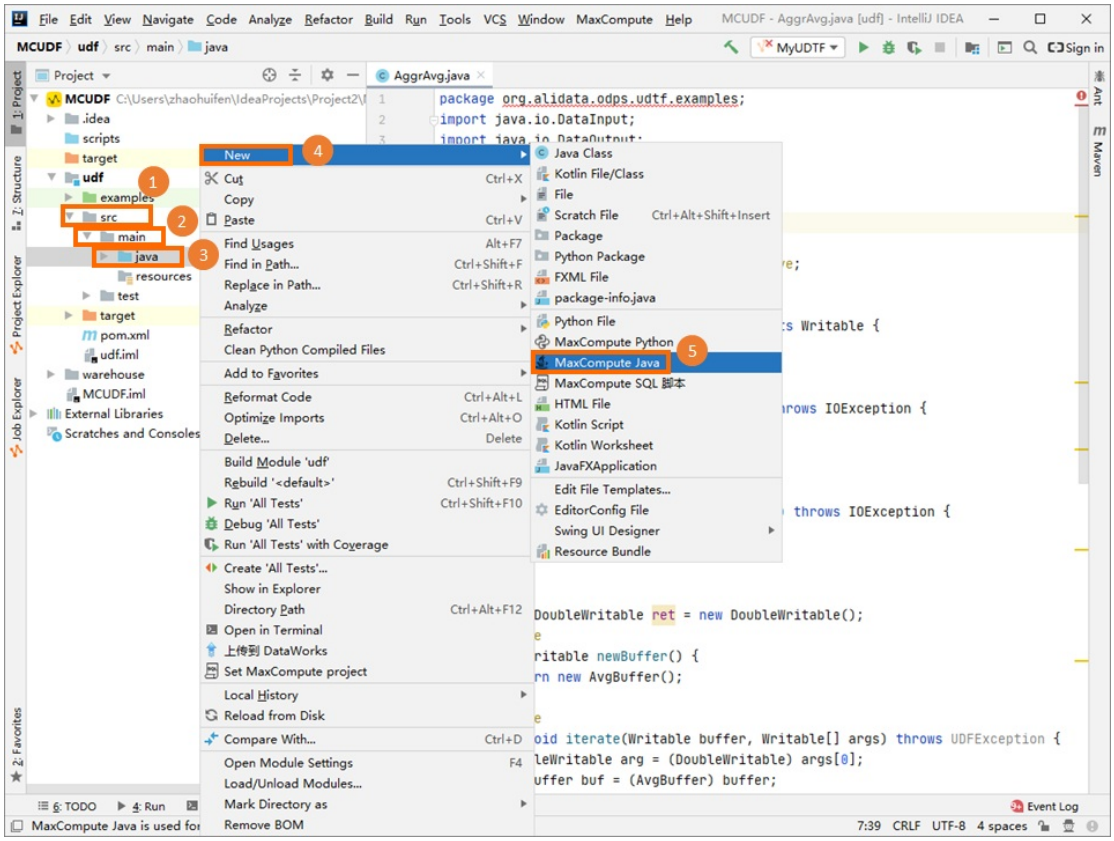
MaxCompute兼容的Hive版本为2.1.0，对应Hadoop版本为2.7.2。如果UDF是在其他版本的Hive或Hadoop上开发的，您需要使用兼容的Hive或Hadoop版本重新编译UDF JAR包。

在MaxCompute上使用Hive UDF的具体案例，请参见兼容Hive Java UDF示例。

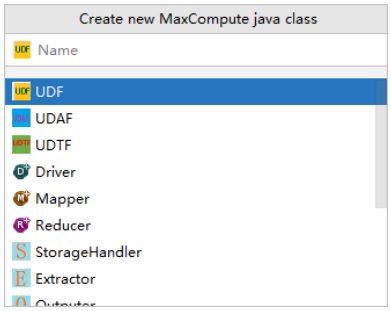
使用示例

以通过MaxCompute Studio开发字符小写转换功能的UDF为例，开发并调用Java UDF的操作步骤如下：

1. 在IntelliJ IDEA上完成如下准备工作：
   i. 安装MaxCompute Studio
   ii. 创建MaxCompute项目连接
   iii. 创建MaxCompute Java Module
2. 编写UDF代码。
   i. 在**Project**区域，右键单击Module的源码目录（即**src > main > java**），选择**new > MaxCompute Java**。

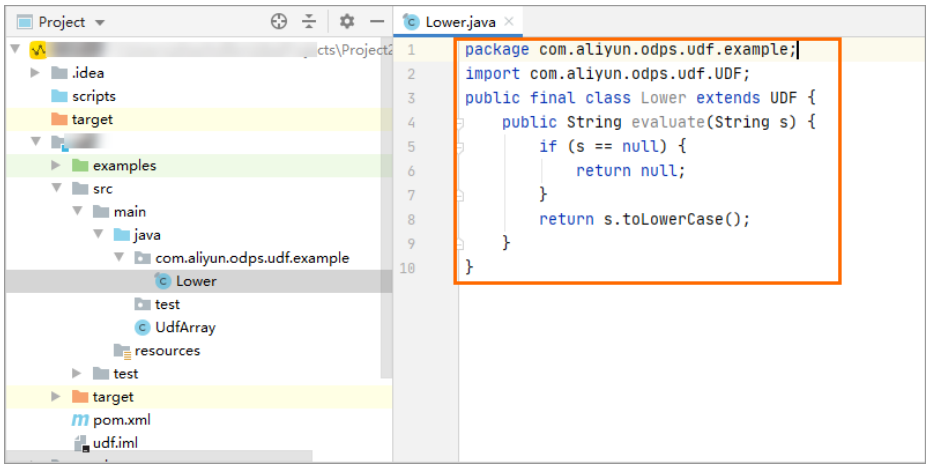


   ii. 在**Create new MaxCompute java class**对话框，单击**UDF**并填写**Name**后，按Enter键。例如Java Class名称为Lower。



Name为创建的MaxCompute Java Class名称。如果还没有创建Package，在此处填写packagename.classname，会自动生成Package。

iii. 在代码编写区域写入如下代码。



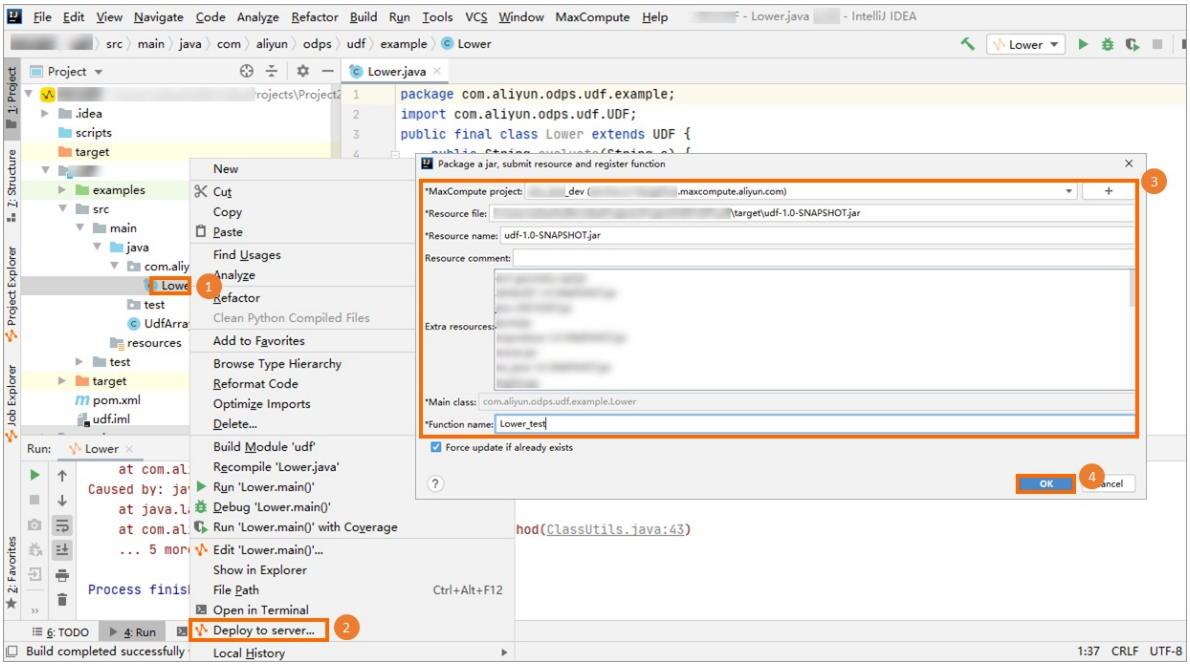
UDF代码示例如下。

```
package <packagename>;
import com.aliyun.odps.udf.UDF;
public final class Lower extends UDF {
    public String evaluate(String s) {
        if (s == null) {
           return null;
        }
           return s.toLowerCase();
    }
}
```

> **说明** 如果需要本地调试Java UDF，请参见开发和调试UDF。

3. 注册MaxCompute UDF。

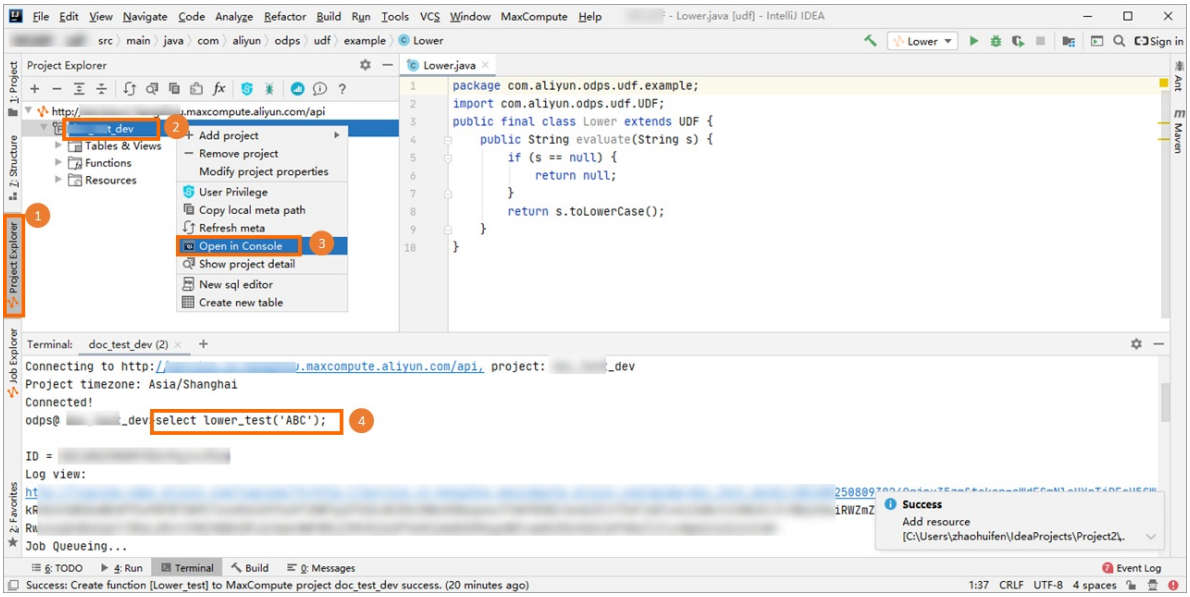
   在UDF Java文件上单击右键，选择**Deploy to server...**，在**Package a jar, submit resource and register function**对话框中配置如下参数后，单击**OK**。



   - **MaxCompute project**：UDF所在的MaxCompute项目名称。由于UDF本身是在连接的MaxCompute项目下编写的，此处保持默认值即可。
   - **Resource file**：UDF依赖的资源文件路径。此处保持默认值即可。
   - **Resource name**：UDF依赖的资源。此处保持默认值即可。
   - **Function name**：注册的函数名称，即后续SQL中调用的UDF名称。例如Lower_test。

4. 在左侧导航栏单击**Project Explore**，在目标MaxCompute项目上单击右键，选择**Open in Console**并在Console区域输入调用UDF的SQL语句，

按Enter键运行即可。



SQL语句示例如下。

```
select lower_test('ABC');
```

返回结果如下。

```
+-----+
| _c0 |
+-----+
| abc |
+-----+
```

3.10.2.3. Python UDF

3.10.2.3.1. Python 2 UDF

MaxCompute使用的Python 2版本为2.7。本文为您介绍如何通过Python 2语言编写UDF。

UDF代码结构

您可以通过MaxCompute Studio工具使用Python 2语言编写UDF代码，代码中需要包含如下信息：

- 编码声明：可选。

  固定声明格式为 `#coding:utf-8` 或 `# -*- coding: utf-8 -*-` ，二者等效。当Python 2代码中出现中文字符时，运行程序会报错，必须在代码头部增加编码声明。

- 导入模块：必选。

  至少要包含 `from odps.udf import annotate` ，导入函数签名模块，MaxCompute才可以识别后续代码中定义的函数签名。当UDF代码中需要引用文件资源或表资源时，需要包含 `from odps.distcache import get_cache_file` （文件资源）或 `from odps.distcache import get_cache_table` （表资源）。

- 函数签名。必选。

  格式为 `@annotate(<signature>)` ， `signature` 用于定义函数的输入参数和返回值的数据类型。更多函数签名信息，请参见函数签名及数据类型。

- 自定义Python类：必选。

  UDF代码的组织单位，定义了实现业务需求的变量及方法。您还可以在代码中引用MaxCompute内置的第三方库或引用文件、表资源。更多信息，请参见第三方库或引用资源。

- `evaluate` 方法：必选。

  位于自定义的Python类中。 `evaluate` 方法定义了输入参数和返回值。一个Python类中只能包含一个 `evaluate` 方法。

UDF代码示例如下。

```
#coding:utf-8
#导入函数签名模块。
from odps.udf import annotate
#函数签名。
@annotate("bigint,bigint->bigint")
#自定义Python类。
class MyPlus(object):
#evaluate方法。
   def evaluate(self, arg0, arg1):
       if None in (arg0, arg1):
           return None
       return arg0 + arg1
```

使用限制

MaxCompute Python 2 UDF使用的Python版本为2.7，并以沙箱模式执行用户代码，即代码是在一个受限的运行环境中执行的。在该环境中，以下行为会被禁止：

- 读写本地文件。
- 启动子进程。
- 启动线程。
- 使用Socket通信。
- 其他系统调用。

基于上述原因，您上传的代码都必须通过标准Python实现。Python标准库中涉及到上述功能的模块或C扩展模块都会被禁止使用。具体标准库的可用模块说明如下：

- 所有基于标准Python实现（不依赖扩展模块）的模块都可用。
- C扩展模块中下列模块可用：
  - array、audioop
  - binascii、bisect
  - cmath、_codecs_cn、_codecs_hk、_codecs_iso2022、_codecs_jp、_codecs_kr、_codecs_tw、_collections、cStringIO
  - datetime
  - _functools、future_builtins、
  - _heapq、_hashlib
  - itertools
  - _json
  - _locale、_lsprof
  - math、_md5、_multibytecodec
  - operator
  - _random
  - _sha256、_sha512、_sha、_struct、strop
  - time
  - unicodedata
  - _weakref
  - cPickle
- 沙箱限制了您的代码最多可向标准输出和标准错误输出写入数据的大小为20 KB，即 `sys.stdout/sys.stderr` 最多能写入20 KB数据，多余的字符会被忽略。

第三方库

MaxCompute的Python 2运行环境中安装了除Python标准库外比较常用的第三方库，作为标准库的补充，例如Numpy。

> **说明** 使用第三方库存在限制，例如禁止本地访问、网络I/O受限，因此第三方库中涉及到相关功能的API也被禁止使用。

函数签名及数据类型

函数签名格式如下。

```
@annotate(<signature>)
```

`signature` 为字符串，用于标识输入参数和返回值的数据类型。执行UDF时，UDF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type'
```

其中：

- `arg_type_list`：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

  `arg_type_list` 还支持星号（*）或为空（''）：

  - 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
  - 当 `arg_type_list` 为空（''）时，表示无输入参数。

- `type`：表示返回值的数据类型。UDF只返回一列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

> **说明** 在编写UDF代码过程中，您可以根据MaxCompute项目的数据类型版本选取合适的数据类型，更多数据类型版本及各版本支持的数据类型信息，请参见数据类型版本说明。

合法的函数签名示例如下。

函数签名示例	说明
`'bigint,double->string'`	输入参数类型为BIGINT、DOUBLE，返回值类型为STRING。
`'*->string'`	输入任意个参数，返回值类型为STRING。
`'->double'`	无输入参数，返回值类型为DOUBLE。
`'array<bigint>->struct<x:string, y:int>'`	输入参数类型为ARRAY<BIGINT>，返回值类型为STRUCT<x:STRING, y:INT>。
`'->map<bigint, string>'`	无输入参数，返回值类型为MAP<BIGINT, STRING>。

为确保编写Python UDF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute SQL Type	Python 2 Type
BIGINT	INT
STRING	STR
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	INT
FLOAT	FLOAT
CHAR	STR
VARCHAR	STR
BINARY	BYTEARRAY
DATE	INT
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT
STRUCT	COLLECTIONS.NAMEDTUPLE

> **说明**
>
> - DATETIME类型对应的Python类型是INT，值为Epoch UTC Time起至今的毫秒数。您可以通过Python标准库中的DATETIME模块处理日期时间类型。
> - `odps.udf.int(value,[silent=True])` 增加了参数 `silent`。当 `silent` 为True时，如果 `value` 无法转为INT，则会返回None（不会返回异常）。
> - NULL值对应Python的None。

引用资源

Python UDF可以通过 `odps.distcache` 模块引用资源，支持引用文件资源和表资源。

- `odps.distcache.get_cache_file(resource_name)` ：返回指定文件资源的内容。
  - `resource_name` 为STRING类型，对应当前MaxCompute项目中已存在的文件资源名。如果文件资源名非法或者没有相应的文件资源，会返回异常。

    > **说明** 使用UDF访问资源，在创建UDF时需要声明引用的资源，否则会报错。

  - 返回值为File-like对象。在使用完此对象后，您需要调用 `close` 方法释放打开的资源文件。

  引用文件资源示例如下。

```
from odps.udf import annotate
from odps.distcache import get_cache_file
@annotate('bigint->string')
class DistCacheExample(object):
def __init__(self):
    cache_file = get_cache_file('test_distcache.txt')
    kv = {}
    for line in cache_file:
        line = line.strip()
        if not line:
            continue
        k, v = line.split()
        kv[int(k)] = v
    cache_file.close()
    self.kv = kv
def evaluate(self, arg):
    return self.kv.get(arg)
```

- `odps.distcache.get_cache_table(resource_name)` ：返回指定表资源的内容。
  - `resource_name` 支持STRING类型，对应当前MaxCompute项目中已存在的表资源名。如果表资源名非法或者没有相应的表资源，会返回异常。
  - 返回值为GENERATOR类型，调用者以遍历方式获取表的内容，每次遍历可得到以数组形式存在的表中的一条记录。

  引用表资源示例如下。

```
from odps.udf import annotate
from odps.distcache import get_cache_table
@annotate('->string')
class DistCacheTableExample(object):
    def __init__(self):
        self.records = list(get_cache_table('udf_test'))
        self.counter = 0
        self.ln = len(self.records)
    def evaluate(self):
        if self.counter > self.ln - 1:
            return None
        ret = self.records[self.counter]
        self.counter += 1
        return str(ret)
```

使用说明

按照开发流程，完成Python 2 UDF开发后，您即可通过MaxCompute SQL调用Python 2 UDF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute Studio完整开发及调用Python 2 UDF的操作，请参见开发Python UDF。

3.10.2.3.2. Python 3 UDF

Python官方即将停止维护Python 2，MaxCompute已支持Python 3，对应版本为CPython-3.7.3。本文为您介绍如何通过Python 3语言编写UDF。

UDF代码结构

您可以通过MaxCompute Studio工具使用Python 3语言编写UDF代码，代码中需要包含如下信息：

- 导入模块：必选。

  至少要包含 `from odps.udf import annotate` ，导入函数签名模块，MaxCompute才可以识别后续代码中定义的函数签名。当UDF代码中需要引用文件资源或表资源时，需要包含 `from odps.distcache import get_cache_file` （文件资源）或 `from odps.distcache import get_cache_table` （表资源）。

- 函数签名：必选。

  格式为 `@annotate(<signature>)` ， `signature` 用于定义函数的输入参数和返回值的数据类型。更多函数签名信息，请参见函数签名与数据类型。

- 自定义Python类：必选。

  UDF代码的组织单位，定义了实现业务需求的变量及方法。您还可以在代码中引用MaxCompute内置的第三方库或引用文件、表资源。更多信息，请参见第三方库或引用资源。

- `evaluate` 方法：必选。

  位于自定义的Python类中。 `evaluate` 方法定义了输入参数和返回值。一个Python类中只能包含一个 `evaluate` 方法。

UDF代码示例如下。

```
#导入函数签名模块。
from odps.udf import annotate
#函数签名。
@annotate("bigint,bigint->bigint")
#自定义Python类。
class MyPlus(object):
#evaluate方法。
   def evaluate(self, arg0, arg1):
       if None in (arg0, arg1):
           return None
       return arg0 + arg1
```

> **说明** Python 2 UDF与Python 3 UDF区别在于底层Python语言版本不一致，请您根据对应版本语言支持的能力编写UDF。

使用限制

Python 3与Python 2不兼容。在您使用Python 3之前，需要考虑兼容性问题，在一个SQL中不允许同时使用Python 3和Python 2。

Python 2 UDF迁移

Python 2官方即将停止维护，建议您根据项目类型执行迁移操作：

- 全新项目：新MaxCompute项目，或第一次使用Python语言编写UDF的MaxCompute项目。建议所有的Python UDF都直接使用Python 3语言编写。
- 存量项目：创建了大量Python 2 UDF的MaxCompute项目。请您谨慎开启Python 3。如果您计划逐步将所有Python 2 UDF迁移为Python 3 UDF，推荐方法如下：
  - 新作业和新UDF：使用Python 3语言编写，在Session级别开启Python 3。开启Python 3方法，请参见开启Python 3。
  - Python 2 UDF：改写Python 2 UDF，使其可以同时兼容Python 2和Python 3。改写方法请参见将Python 2代码移植到Python 3。

    > **说明** 如果您需要编写公共UDF，并为多个MaxCompute项目授权UDF的操作权限，建议UDF同时兼容Python 2和Python 3。

开启Python 3

MaxCompute默认使用Python 2，如果您要使用Python 3，可以在Session级别设置如下属性开启Python 3，并与SQL语句一起提交执行。

```
set odps.sql.python.version=cp37;
```

第三方库

MaxCompute内置的Python 3运行环境中未安装第三方库Numpy。如果您需要使用Numpy的UDF，请手动上传Numpy的WHEEL包。从PyPI或镜像下载Numpy包时，包的文件名为*numpy-<版本号>-cp37-cp37m-manylinux1_x86_64.whl*。上传包的操作请参见资源操作或Python UDF使用第三方包。

Python 3支持的标准库列表请参见Python 3标准库。

函数签名与数据类型

函数签名格式如下。

```
@annotate(<signature>)
```

`signature` 为字符串，用于标识输入参数和返回值的数据类型。执行UDF时，UDF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type'
```

其中：

- `arg_type_list` ：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

`arg_type_list` 还支持星号（*）或为空（''）：

- 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
- 当 `arg_type_list` 为空（''）时，表示无输入参数。

- `type`：表示返回值的数据类型。UDF只返回一列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

> **说明** 在编写UDF代码过程中，您可以根据MaxCompute项目的数据类型版本选取合适的数据类型，更多数据类型版本及各版本支持的数据类型信息，请参见数据类型版本说明。

合法的函数签名示例如下。

函数签名示例	说明
`'bigint,double->string'`	输入参数类型为BIGINT、DOUBLE，返回值类型为STRING。
`'*->string'`	输入任意个参数，返回值类型为STRING。
`'->double'`	无输入参数，返回值类型为DOUBLE。
`'array<bigint>->struct<x:string, y:int>'`	输入参数类型为ARRAY<BIGINT>，返回值类型为STRUCT<x:STRING, y:INT>。
`'->map<bigint, string>'`	无输入参数，返回值类型为MAP<BIGINT, STRING>。

为确保编写Python UDF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute SQL Type	Python 3 Type
BIGINT	INT
STRING	UNICODE
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	DATETIME.DATETIME
FLOAT	FLOAT
CHAR	UNICODE
VARCHAR	UNICODE
BINARY	BYTES
DATE	DATETIME.DATE
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT
STRUCT	COLLECTIONS.NAMEDTUPLE

引用资源

Python UDF可以通过 `odps.distcache` 模块引用资源，支持引用文件资源和表资源。

- `odps.distcache.get_cache_file(resource_name, mode)`：以指定模式 `mode` 返回指定文件资源的内容。
  - `resource_name` 支持STRING类型，对应当前MaxCompute项目中已存在的表资源名。如果表资源名非法或者没有相应的表资源，会返回异常。
  - `mode` 支持STRING类型，默认值为 `'t'`。当 `mode` 为 `'t'` 时以文本格式打开文件，当 `mode` 为 `'b'` 时以二进制格式打开文件。
  - 返回值为File-like对象。在使用完此对象后，您需要调用 `close` 方法释放打开的资源文件。

  引用文件资源示例如下。

```
from odps.udf import annotate
from odps.distcache import get_cache_file
@annotate('bigint->string')
class DistCacheExample(object):
def __init__(self):
    cache_file = get_cache_file('test_distcache.txt')
    kv = {}
    for line in cache_file:
        line = line.strip()
        if not line:
            continue
        k, v = line.split()
        kv[int(k)] = v
    cache_file.close()
    self.kv = kv
def evaluate(self, arg):
    return self.kv.get(arg)
```

- `odps.distcache.get_cache_table(resource_name)`：返回指定资源表的内容。
  - `resource_name` 对应当前MaxCompute项目中已存在的表资源名。如果表资源名非法或者没有相应的表资源，会返回异常。支持读取表中BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、FLOAT、CHAR、VARCHAR、BINARY、DATE、DECIMAL、ARRAY、MAP和STRUCT类型数据。
  - 返回值为Generator类型，调用者通过遍历获取表的内容，每次遍历得到的是以数组形式存在的表中的一条记录。

引用表资源示例如下。

```
from odps.udf import annotate
from odps.distcache import get_cache_table
@annotate('->string')
class DistCacheTableExample(object):
    def __init__(self):
        self.records = list(get_cache_table('udf_test'))
        self.counter = 0
        self.ln = len(self.records)
    def evaluate(self):
        if self.counter > self.ln - 1:
            return None
        ret = self.records[self.counter]
        self.counter += 1
        return str(ret)
```

使用说明

按照开发流程，完成Python 3 UDF开发后，您即可通过MaxCompute SQL调用Python 3 UDF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;`。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute客户端完整开发及调用Python 3 UDF的操作，请参见Python UDF使用示例。

3.10.2.4. 通过UDF访问VPC网络资源

MaxCompute默认不支持通过UDF访问VPC网络。如果您的UDF涉及访问VPC网络中的资源时，需要先创建MaxCompute与目标VPC网络间的网络连接，才可以直接通过UDF访问VPC网络中的资源。本文为您介绍如何通过UDF访问VPC网络资源。

前提条件

请确认您已满足如下条件：

- 已编写UDF代码并注册函数。

  更多编写UDF及注册函数操作信息，请参见Java UDF或Python UDF。

- 已安装MaxCompute客户端。

  更多安装及配置MaxCompute客户端信息，请参见安装并配置MaxCompute客户端。

背景信息

通过UDF访问VPC网络资源时，需要通过专有网络连接方案创建MaxCompute与目标VPC网络间的网络连接。创建网络连接后，在使用SQL命令调用UDF前，需要在SQL语句前增加 `set odps.session.networklink=<networklink_name>;` 属性与SQL语句一起提交执行。`networklink_name` 为创建的网络连接名称。

使用限制

由于专有网络连接方案存在地域限制，仅华北2（北京）、华东2（上海）、华北3（张家口）、华东1（杭州）、华南1（深圳）地域支持通过UDF访问VPC网络资源。

步骤一：创建网络连接

登录MaxCompute控制台，创建网络连接。详细操作指导请参见专有网络连接方案。

步骤二：调用UDF访问VPC网络资源

1. 登录MaxCompute客户端。
2. 执行SQL命令调用UDF。

   假设已通过UDF注册的函数名称为my_ping，用于判断IP是否可以ping通。命令示例如下。

   ```
   --设置网络连接名称，即基于专有网络连接方案配置的网连接名称，仅本Session有效。
   set odps.session.networklink=testLink;
   --执行SQL调用UDF访问VPC网络资源。
   select my_ping('123.56.xxx.xxx');
   ```

   执行结果返回True，表示网络已连通，UDF可正常调用。如果返回报错，请重新确认网络连接信息配置是否正确。

3.10.2.5. Java UDF使用示例

3.10.2.5.1. 兼容Hive Java UDF示例

本文以在MaxCompute客户端操作为例，为您介绍如何使用在MaxCompute兼容的Hive版本上开发的Hive UDF。

前提条件

已安装MaxCompute客户端。更多安装操作，请参见安装并配置MaxCompute客户端。

注意事项

使用兼容的Hive UDF时，您需要注意：

- 在MaxCompute上使用 `add jar` 命令添加Hive UDF的资源时，您需要指定所有JAR包，MaxCompute无法自动将所有JAR包加入Classpath。
- 调用Hive UDF时，需要在SQL语句前添加 `set odps.sql.hive.compatible=true;` 语句，与SQL语句一起提交执行。
- Java UDF在分布式环境中运行时，请注意MaxCompute的Java沙箱限制。

Hive UDF代码示例

Hive UDF代码如下。

```
package com.aliyun.odps.compiler.hive;
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
public class Collect extends GenericUDF {
  @Override
  public ObjectInspector initialize(ObjectInspector[] objectInspectors) throws UDFArgumentException {
    if (objectInspectors.length == 0) {
      throw new UDFArgumentException("Collect: input args should >= 1");
    }
    for (int i = 1; i < objectInspectors.length; i++) {
      if (objectInspectors[i] != objectInspectors[0]) {
        throw new UDFArgumentException("Collect: input oi should be the same for all args");
      }
    }
    return ObjectInspectorFactory.getStandardListObjectInspector(objectInspectors[0]);
  }
  @Override
  public Object evaluate(DeferredObject[] deferredObjects) throws HiveException {
    List<Object> objectList = new ArrayList<>(deferredObjects.length);
    for (DeferredObject deferredObject : deferredObjects) {
      objectList.add(deferredObject.get());
    }
    return objectList;
  }
  @Override
  public String getDisplayString(String[] strings) {
    return "Collect";
  }
}
```

该UDF代码示例可以将任意类型、数量的参数打包成ARRAY输出。假设Hive UDF对应的JAR包名称为test.jar。

操作步骤

1. 登录MaxCompute客户端。
2. 将Hive UDF代码示例通过Hive平台编译为JAR包，执行如下命令将Hive UDF JAR包添加为MaxCompute资源。

   ```
   --添加资源。
   add jar test.jar;
   ```

   更多添加资源信息，请参见添加资源。
3. 执行如下命令注册UDF函数。

   ```
   --注册函数。
   create function hive_collect as 'com.aliyun.odps.compiler.hive.Collect' using 'test.jar';
   ```

   更多注册函数信息，请参见注册函数。
4. 执行如下SQL语句调用新建的UDF函数。

   ```
   --设置MaxCompute项目的模式为Hive兼容模式。
   set odps.sql.hive.compatible=true;
   --调用UDF函数。
   select hive_collect(4y, 5y, 6y);
   ```

   返回结果如下。

   ```
   +------+
   | _c0  |
   +------+
   | [4, 5, 6] |
   +------+
   ```

3.10.2.5.2. 复杂数据类型示例

本文以在MaxCompute Studio操作为例，为您介绍如何在Java UDF中使用复杂数据类型。

前提条件

已安装MaxCompute Studio，并连接至MaxCompute项目，创建了MaxCompute Java Module。

更多操作信息，请参见安装MaxCompute Studio、管理项目连接和创建MaxCompute Java Module。

UDF代码示例

如下代码中，定义了3个重载的evaluate方法。其中：

- 第一个用ARRAY作为参数，ARRAY对应java.util.List。
- 第二个用MAP作为参数，MAP对应java.util.Map。
- 第三个用STRUCT作为参数，STRUCT对应com.aliyun.odps.data.Struct。

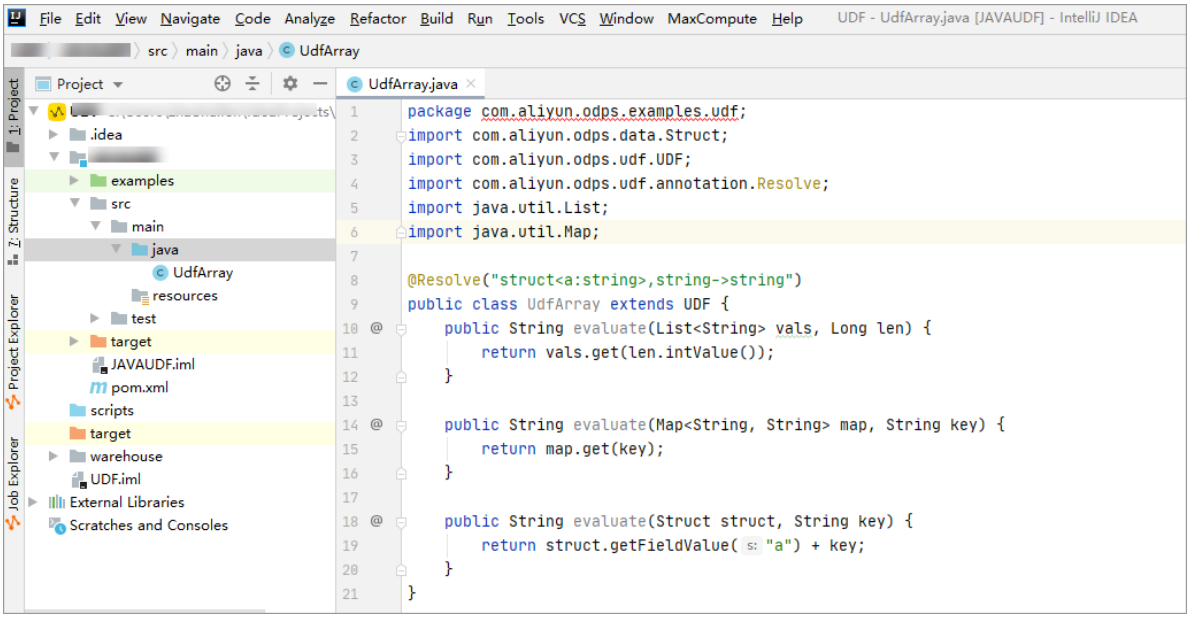
> **说明** com.aliyun.odps.data.Struct无法通过反射分析获取到field name和field type，需要辅助使用 `@Resolve annotation` ，即如果您需要在UDF中使用STRUCT，要求在UDF class上也标注上 `@Resolve` 注解，该注解只会影响参数或返回值中包含com.aliyun.odps.data.Struct的重载。

```
import com.aliyun.odps.data.Struct;
import com.aliyun.odps.udf.UDF;
import com.aliyun.odps.udf.annotation.Resolve;
import java.util.List;
import java.util.Map;
@Resolve("struct<a:string>,string->string")
public class UdfArray extends UDF {
    //接收两个参数，第一个参数对应ARRAY复杂数据类型，第二个参数对应要获取的元素的index，目的是要取出位于index位置的元素。
    public String evaluate(List<String> vals, Long index) {
        return vals.get(index.intValue());
    }
    //接收两个参数，第一个参数对应MAP复杂数据类型，第二个参数对应要取出的Key，目的是要取出Key对应的值。
    public String evaluate(Map<String, String> map, String key) {
        return map.get(key);
    }
    //接收两个参数，第一个参数对应STRUCT复杂数据类型，第二个参数为一个Key值，目的是要取出STRUCT中成员变量a对应的值，并在其后增加Key值以STRING格式返回。
    public String evaluate(Struct struct, String key) {
        return struct.getFieldValue("a") + key;
    }
}
```

操作步骤

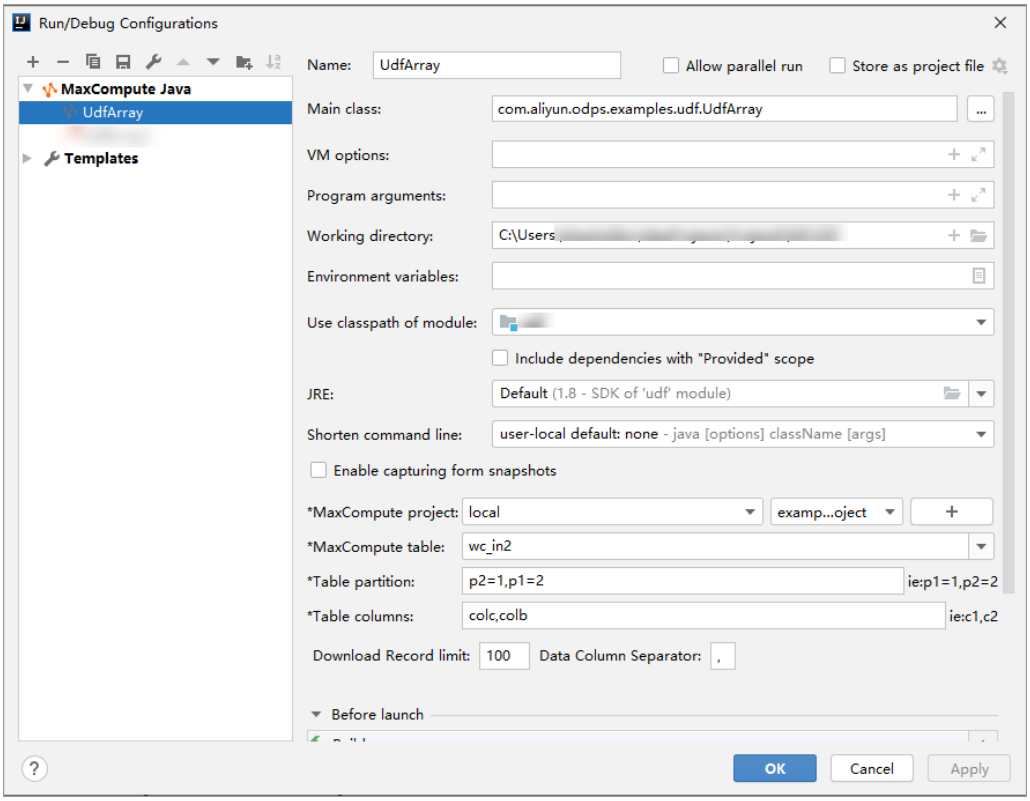
1. 在MaxCompute Studio中新建UDF类型的Java程序。例如Java Class名称为 `UdfArray` ，程序代码为UDF代码示例中的代码。

   更多创建UDF操作，请参见编写UDF。



2. 在本地运行调试UDF，确保代码可以运行成功。

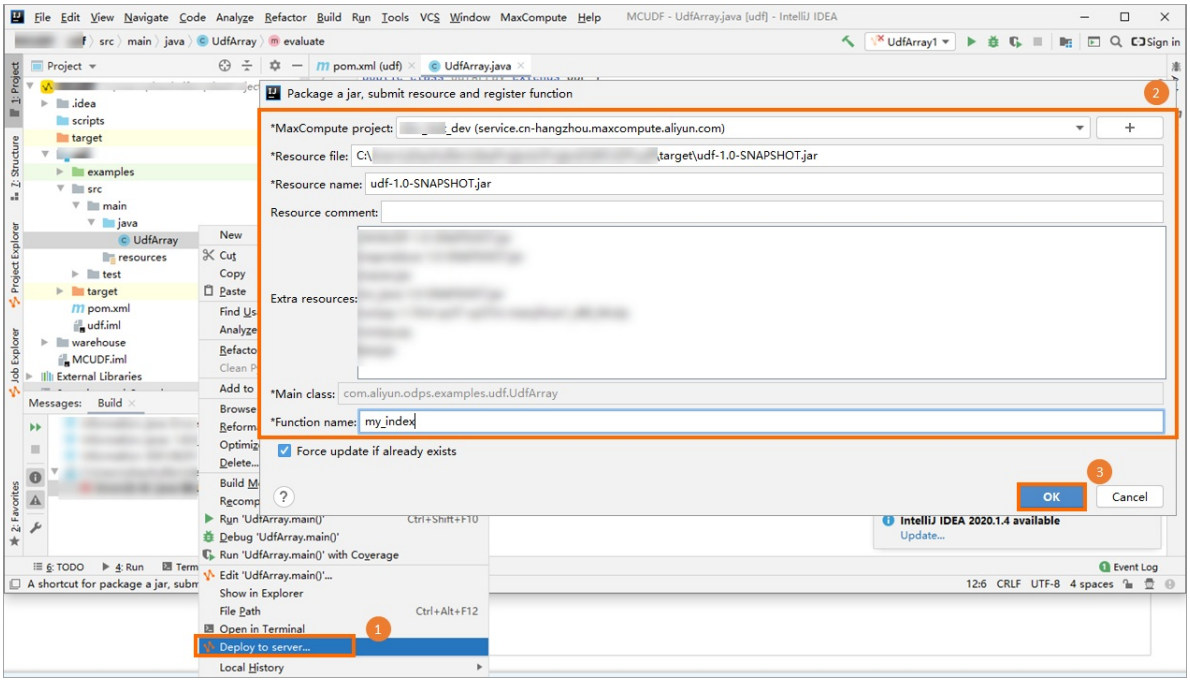
   更多调试操作，请参见通过本地运行调试UDF。



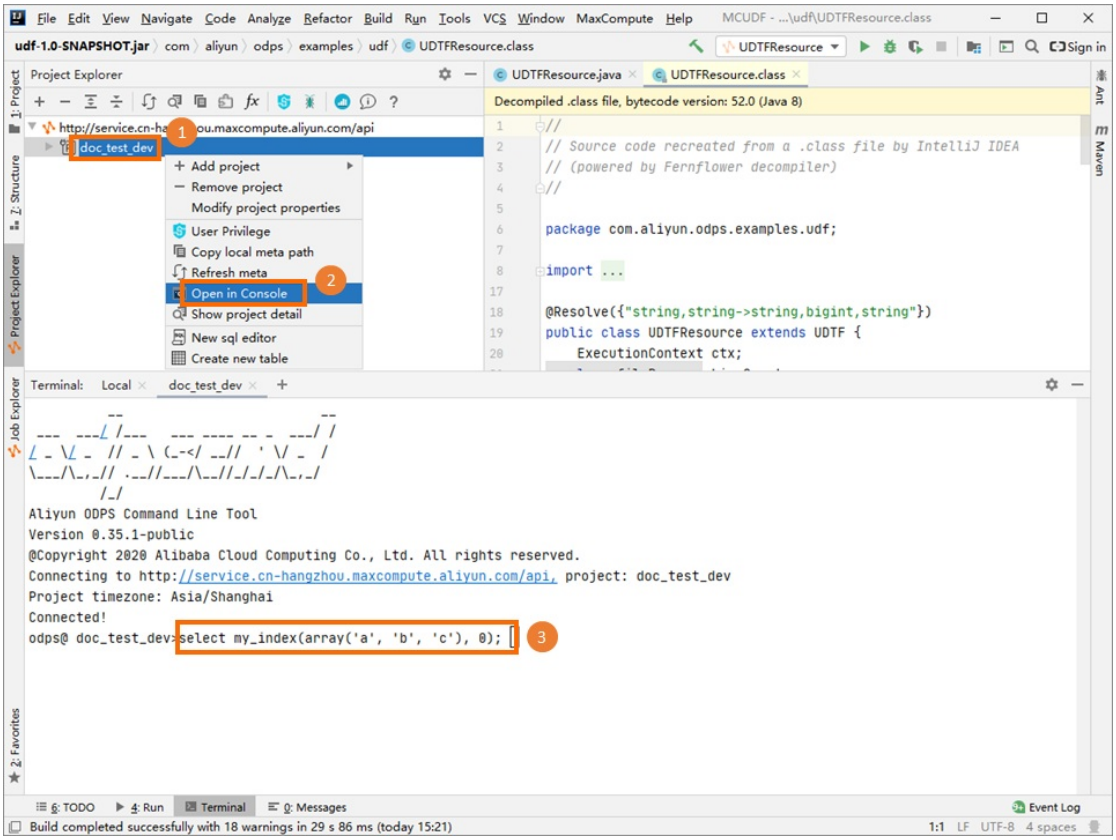
> **说明** 运行参数可参照图示数据填写。

3. 将创建的UDF打包为JAR包，上传至MaxCompute项目并注册函数。例如函数名称为 `my_index` 。

   更多打包操作，请参见操作步骤。



4. 在MaxCompute Studio的左侧导航栏，单击**Project Explorer**，在目标MaxCompute项目上单击右键，启动MaxCompute客户端，并执行SQL命令调用新创建的UDF。



命令示例如下。

```
select my_index(array('a', 'b', 'c'), 0); --返回a。
select my_index(map('key_a','val_a', 'key_b', 'val_b'), 'key_b'); --返回val_b。
select my_index(named_struct('a', 'hello'), 'world'); --返回hello world。
```

3.10.2.5.3. JSON字符串获取示例

本文为您介绍如何通过Java UDF实现根据Path路径获取JSON字符串。

JSON字符串获取UDF

```
UDFGetJsonObject(String jsonString, String pathString)
```

- 函数功能

  解析JSON的字符串json_string，返回路径指定的内容。如果输入的JSON字符串无效，则返回NULL。

- 参数说明
  - json_string：JSON字符串。
  - pathString：路径。其中 `$` 代表根节点，`.` 取子节点，`[]` 取数组下标。取值仅支持字母、数字、下划线。

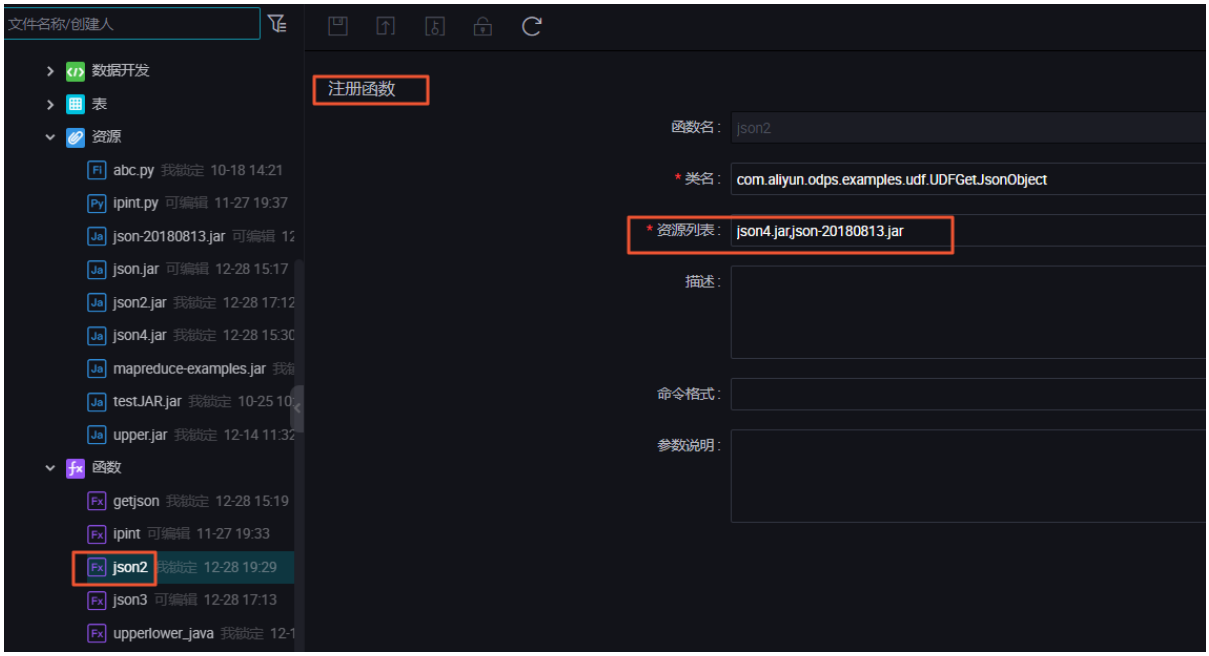
UDF使用示例

- 资源上传

  在MaxCompute公共云运行时，由于沙箱等限制，需要把*org.json*包和UDF JAR包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*org.json*包。

- 注册函数

*UDFGetJsonObject.java*测试成功后，需要注册函数。本示例中，*json4.jar*是UDF打包后生成的JAR，而 `json-20180813.jar` 是 `org.json` 包。



- 使用示例

  成功注册UDF后，执行如下命令测试。

```
select UDFGetJsonObject('{"grade":"一年级","persons":[{"age":"a1","name":"n1"},{"age":"a2","name":"n2"}]}', '$.persons[0].age')
;
```

  执行结果为 `a1` 。

UDF代码示例

```
package com.aliyun.odps.examples.udf; //package名称，可以根据您的情况定义。
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
//阿里云UDF。
import com.aliyun.odps.udf.UDF;
public class UDFGetJsonObject extends UDF {
  //声明私有不可变Pattern正则表达式模式。
  private final Pattern patternKey = Pattern.compile("^([a-zA-Z0-9_\\-]+).*");
  private final Pattern patternIndex = Pattern.compile("\\[([0-9]+|\\*)\\]");
  // 使用链接哈希映射的LRU表缓存。
  // 定义HashCache静态内部子类，继承LinkedHashMap。
  static class HashCache<K, V> extends LinkedHashMap<K, V> {
    private static final int CACHE_SIZE = 16;
    private static final int INIT_SIZE = 32;
    // 声明私有常量HashMap的LOAD_FACTOR装载因子。
    private static final float LOAD_FACTOR = 0.6f;
    HashCache() {
      super(INIT_SIZE, LOAD_FACTOR);
    }
    private static final long serialVersionUID = 1;
    // 重载LinkedHashMap方法，用于判断是否超过Cache大小并清理缓存。
    @Override
    protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
      return size() > CACHE_SIZE;
    }
  }
  //使用HashCache构造方法创建Map类型缓存区域。
  static Map<String, Object> extractObjectCache = new HashCache<String, Object>();
  static Map<String, String[]> pathExprCache = new HashCache<String, String[]>();
```

```
static Map<String, ArrayList<String>> indexListCache = new HashCache<String, ArrayList<String>>();
static Map<String, String> mKeyGroup1Cache = new HashCache<String, String>();
static Map<String, Boolean> mKeyMatchesCache = new HashCache<String, Boolean>();
String result = new String();
public UDFGetJsonObject() {
}
/**
 * Extract json object from a json string based on json path specified, and
 * return json string of the extracted json object. It will return null if the
 * input json string is invalid.
 *
 * A limited version of JSONPath supported: $ : Root object . : Child operator
 * [] : Subscript operator for array * : Wildcard for []
 *
 * Syntax not supported that's worth noticing: '' : Zero length string as key
 * .. : Recursive descent &amp;#064; : Current object/element () : Script
 * expression ?() : Filter (script) expression. [,] : Union operator
 * [start:end:step] : array slice operator
 *
 * @param jsonString
 *          the json string.
 * @param pathString
 *          the json path expression.
 * @return json string or null when an error happens.
 */
public String evaluate(String jsonString, String pathString) {
  // 创建evaluate方法，与HIVE一样，MaxCompute的UDF通常使用evaluate方法，jsonString作为输入的JSON字符串，pathString作为输入的路径。
  // 判断如果输入的JSON字符串路径为空则返回null。
  if (jsonString == null || jsonString.equals("") || pathString == null
      || pathString.equals("")) {
    return null;
  }
  try {
    // Cache pathExpr
    // 获取string数组类型的输入路径。
    String[] pathExpr = pathExprCache.get(pathString);
    if (pathExpr == null) {
      // 用“.”作为分隔，获取输入路径的string数组pathExpr，-1表示如果最后几位是分隔符也会继续切割。
      pathExpr = pathString.split("\\.", -1);
      // 使用put方法将pathString、pathExpr缓存到pathExprCache。
      pathExprCache.put(pathString, pathExpr);
    }
    // 如果path首位不为$，则返回null。
    if (!pathExpr[0].equalsIgnoreCase("$")) {
      return null;
    }
    // Cache extractObject
    Object extractObject = extractObjectCache.get(jsonString);
    if (extractObject == null) {
      extractObject = new JSONObject(jsonString);
      // 使用put方法缓存JSON字符串内容。
      extractObjectCache.put(jsonString, extractObject);
    }
    for (int i = 1; i < pathExpr.length; i++) {
      // 根据路径参数pathExpr使用extract方法取出JSON字符串。
      extractObject = extract(extractObject, pathExpr[i]);
    }
    // 将获取的object类型的JSON字符串以string类型输出。
    return extractObject.toString();
  } catch (Exception e) {
    // 发生异常时返回null。
    return null;
  }
}
private Object extract(Object json, String path) throws JSONException {
  // Cache patternkey.matcher(path).matches()
  Matcher mKey = null;
  Boolean mKeyMatches = mKeyMatchesCache.get(path);
  if (mKeyMatches == null) {
    // 如果缓存中不存在path。
    mKey = patternKey.matcher(path);
    // 如果路径可以匹配到JSON字符串，返回True，如果无法匹配，返回False。
    mKeyMatches = mKey.matches() ? Boolean.TRUE : Boolean.FALSE;
    mKeyMatchesCache.put(path, mKeyMatches);
  }
```

```
    if (!mKeyMatches.booleanValue()) {
      return null;
    }
    // Cache mkey.group(1)
    // 根据path获取对应的JSON。
    String mKeyGroup1 = mKeyGroup1Cache.get(path);
    if (mKeyGroup1 == null) {
      // 判断缓存中是否已存在JSON。
      if (mKey == null) {
        mKey = patternKey.matcher(path);
      }
      mKeyGroup1 = mKey.group(1);
      // 根据path缓存对应的JSON。
      mKeyGroup1Cache.put(path, mKeyGroup1);
    }
    // 获取JSON。
    json = extract_json_withkey(json, mKeyGroup1);
    // Cache indexList
    // 根据获取对应的数组类型JSON。
    ArrayList<String> indexList = indexListCache.get(path);
    if (indexList == null) {
      // 判断缓存中是否已存在数组类型JSON。
      Matcher mIndex = patternIndex.matcher(path);
      indexList = new ArrayList<String>();
      while (mIndex.find()) {
        indexList.add(mIndex.group(1));
      }
      indexListCache.put(path, indexList);
    }
    if (indexList.size() > 0) {
      // 根据path缓存对应数组类型的JSON。
      json = extract_json_withindex(json, indexList);
    }
    return json;
  }
  // 创建数组类型JSON对象。
  ArrayList<Object> jsonList = new ArrayList<Object>();
  // 创建获取数组类型JSON的方法。
  private Object extract_json_withindex(Object json, ArrayList<String> indexList)
      throws JSONException {
    jsonList.clear();
    jsonList.add(json);
    Iterator<String> itr = indexList.iterator();
    while (itr.hasNext()) {
      String index = itr.next();
      ArrayList<Object> tmp_jsonList = new ArrayList<Object>();
      // 如果path路径中包含通配符，则获取所有通配符匹配的JSON。
      if (index.equalsIgnoreCase("*")) {
        for (int i = 0; i < (jsonList).size(); i++) {
          try {
            JSONArray array = (JSONArray) (jsonList).get(i);
            for (int j = 0; j < array.length(); j++) {
              tmp_jsonList.add(array.get(j));
            }
          } catch (Exception e) {
            continue;
          }
        }
        jsonList = tmp_jsonList;
      } else {
        // 如果不存在通配符，则遍历JSON。
        for (int i = 0; i < (jsonList).size(); i++) {
          try {
            tmp_jsonList.add(((JSONArray) (jsonList).get(i)).get(Integer
                .parseInt(index)));
          } catch (ClassCastException e) {
            continue;
          } catch (JSONException e) {
            return null;
          }
          jsonList = tmp_jsonList;
        }
      }
    }
    return (jsonList.size() > 1) ? new JSONArray(jsonList) : jsonList.get(0);
```

```
  }
  // 创建获取普通JSON的方法。
  private Object extract_json_withkey(Object json, String path)
      throws JSONException {
    if (json.getClass() == org.json.JSONArray.class) {
      JSONArray jsonArray = new JSONArray();
      for (int i = 0; i < ((JSONArray) json).length(); i++) {
        Object josn_elem = ((JSONArray) json).get(i);
        try {
          Object json_obj = ((JSONObject) josn_elem).get(path);
          if (json_obj.getClass() == org.json.JSONArray.class) {
            for (int j = 0; j < ((JSONArray) json_obj).length(); j++) {
              jsonArray.put(((JSONArray) json_obj).get(j));
            }
          } else {
            jsonArray.put(json_obj);
          }
        } catch (Exception e) {
          continue;
        }
      }
      return (jsonArray.length() == 0) ? null : jsonArray;
    } else {
      return ((JSONObject) json).get(path);
    }
  }
}
```

3.10.2.5.4. JSON字符串值判断示例

本文为您介绍如何通过Java UDF实现根据Path路径获取JSON字符串并判断指定值是否存在。

JSON字符串值判断UDF

```
UDFGetJsonObject(String jsonString, String pathString)
```

- 函数功能

  解析jsonString字符串，获取路径pathString指定的值，与JSON字符串获取示例中的get_json_object类似。它还可用于判断获取的值中是否包含特定的值。

- 参数说明
  - jsonString：JSON字符串。
  - pathString：路径。其中 `$` 代表根节点， `.` 取子节点， `[]` 取数组下标。pathString只支持字母、数字、下划线。

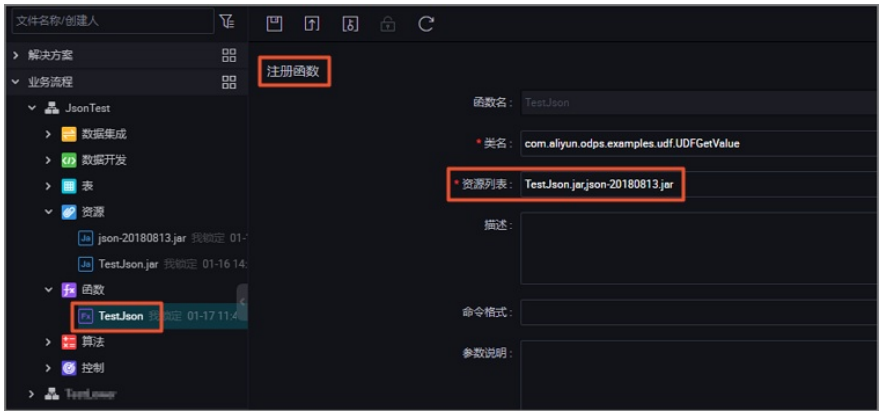
UDF使用示例

- 资源上传

  在MaxCompute公共云运行时，由于沙箱等限制，需要把*org.json*包和UDF Jar包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*org.json*包。

- 注册函数

  *TestJson.jar*是UDF打包后生成的Jar，而*json-20180813.jar*是 `org.json` 包。



- 使用示例

成功注册UDF后，执行以下命令：

```
select TestJson('{"LogA\":[{"id":"11562,20508"}, {"id":"11563,20509"}]}', "LogA[0]");
```

执行结果为 `{"id":"11562,20508"}` 。

```
select bi_udf:bi_get_value('{"LogA\":[{"id":"11562,20508"}, {"id":"11563,20509"}]}', "LogA[0].id", "20508");
```

执行结果为 `true` 。

```
select bi_udf:bi_get_value('{"LogA\":[{"id":"11562,20508"}, {"id":"11563,20509"}]}', "LogA[0].id", "20508", "&");
```

执行结果为 `true` 。

UDF代码示例

```
package com.aliyun.odps.examples.udf; //Package名称，可以根据您的情况定义。
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF; //阿里云UDF。
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class UDFGetValue extends UDF {
    //声明私有不可变Pattern正则表达式模式。
    private final Pattern patternKey = Pattern.compile("^([a-zA-Z0-9_\\-]+).*");
    private final Pattern patternIndex = Pattern.compile("\\[([0-9]+|\\*)\\]");
    //使用HashCache构造方法创建Map类型缓存区域。
    private static Map<String, Object> extractObjectCache = new HashCache<String, Object>();
    private static Map<String, String[]> pathExprCache = new HashCache<String, String[]>();
    private static Map<String, ArrayList<String>> indexListCache = new HashCache<String, ArrayList<String>>();
    private static Map<String, String> mKeyGroup1Cache = new HashCache<String, String>();
    private static Map<String, Boolean> mKeyMatchesCache = new HashCache<String, Boolean>();
    private Text result = new Text();
    private ArrayList<Object> jsonList = new ArrayList<Object>();
    //LRU表缓存，即使用链接散列映射的LRU缓存。
    //定义HashCache静态内部子类，继承LinkedHashMap。
    private static class HashCache<K, V> extends LinkedHashMap<K, V> {
        private static final int CACHE_SIZE = 16;
        private static final int INIT_SIZE = 32;
        //声明私有常量HashMap的LOAD_FACTOR装载因子。
        private static final float LOAD_FACTOR = 0.6f;
        private static final long serialVersionUID = 1;
        HashCache() {
            super(INIT_SIZE, LOAD_FACTOR);
        }
        //重载LinkedHashMap方法，用于判断是否超过Cache大小并清理缓存。
        @Override
        protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
            return size() > CACHE_SIZE;
        }
    }
    public UDFGetValue() {
    }
    /**
     * @param jsonStr 传入的JSON字符串。
     * @param path Json路径。
     * @return 返回对应的值。
     */
    public String evaluate(String jsonStr, String path) {
        //创建evaluate方法，与HIVE一样，MaxCompute的UDF通常使用evaluate方法，jsonString作为输入的JSON字符串，pathString作为输入的路径。
        //判断如果输入的JSON字符串路径为空则返回Null。
        if (jsonStr == null || jsonStr == "" || path == null || path == "") {
            return null;
        }
        String jResult = evaluateJson(jsonStr, path);
        if (jResult == null || "".equals(jResult))
            return null;
        result.set(jResult);
        return result.toString();
    }
```

```
}
/**
 * @param jsonStr 传入的JSON字符串。
 * @param path Json路径。
 * @param value。
 * @return 判断JSON字符串中，指定的path中，是否包含有value（完全包含）,返回"true" 或者"false"。
 */
public String evaluate(String jsonStr, String path, String value) {
    if (jsonStr == null || jsonStr == "" || path == null
            || path == "" || value == null || "".equals(value)) {
        return null;
    }
    String jResult = evaluateJson(jsonStr, path);
    if (jResult == null || "".equals(jResult)) {
        result.set("false");
        return result.toString();
    }
    String[] sV = value.split(",");
    String[] dV = jResult.split(",");
    int sameCount = 0;
    for (int i = 0; i < sV.length; i++) {
        for (int j = 0; j < dV.length; j++) {
            if (sV[i].equals(dV[j])) {
                sameCount++;
                break;
            }
        }
    }
    if (sameCount == sV.length) {
        result.set("true");
    } else {
        result.set("false");
    }
    return result.toString();
}
/**
 * @param jsonStr 传入的JSON字符串。
 * @param path JSON路径。
 * @param value。
 * @param flag 可以取值"|"或者"&","|"表示value中任意一个存在即可，"&"表示value中的值都需要包含。
 * @return 判断获取的JSON字符串中，是否包含有value，返回"true" 或者"false"。
 */
public String evaluate(String jsonStr, String path, String value, String flag) {
    if ("&".equals(flag))
        return evaluate(jsonStr, path, value);
    if (jsonStr == null || jsonStr.equals("") || path == null || path.equals("")
            || value == null || "".equals(value) || flag == null
            || !"|".equals(flag)) {
        result.set("false");
        return result.toString();
    }
    String jResult = evaluateJson(jsonStr, path);
    if (jResult == null || "".equals(jResult)) {
        result.set("false");
        return result.toString();
    }
    String[] sV = value.split(",");
    String[] dV = jResult.split(",");
    for (int i = 0; i < sV.length; i++) {
        for (int j = 0; j < dV.length; j++) {
            if (sV[i].equals(dV[j])) {
                result.set("true");
                return result.toString();
            }
        }
    }
    result.set("false");
    return result.toString();
}
/**
 * Extract json object from a json string based on json path specified, and
 * return json string of the extracted json object. It will return null if
 * the input json string is invalid.
 *
 * A limited version of JSONPath supported: $ : Root object . : Child
 * operator [] : Subscript operator for array * : Wildcard for []
```

```
     *
     * Syntax not supported that's worth noticing: '' : Zero length string as
     * key .. : Recursive descent &amp;#064; : Current object/element () :
     * Script expression ?() : Filter (script) expression. [,] : Union operator
     * [start:end:step] : array slice operator
     *
     * @param jsonString the json string.
     * @param pathString the json path expression.
     * @return json string or null when an error happens.
     */
    private String evaluateJson(String jsonString, String pathString) {
        try {
            pathString = "$." + pathString;
            // Cache pathExpr
            //获取string数组类型的输入路径。
            String[] pathExpr = pathExprCache.get(pathString);
            if (pathExpr == null) {
                //用“.”作为分隔符，获取输入路径的string数组pathExpr，-1表示如果最后几位是分隔符也会继续分割。
                pathExpr = pathString.split("\\.", -1);
                //使用put方法将pathString、pathExpr缓存到pathExprCache。
                pathExprCache.put(pathString, pathExpr);
            }
            //如果path首位不为$，则返回null。
            if (!pathExpr[0].equalsIgnoreCase("$")) {
                return null;
            }
            // Cache extractObject
            Object extractObject = extractObjectCache.get(jsonString);
            if (extractObject == null) {
                extractObject = new JSONObject(jsonString);
                Object put = extractObjectCache.put(jsonString, extractObject);
            }
            for (int i = 1; i < pathExpr.length; i++) {
                //根据路径参数pathExpr使用extract方法取出JSON字符串。
                extractObject = extract(extractObject, pathExpr[i]);
            }
            //将获取的object类型的JSON字符串以string类型输出。
            return extractObject.toString();
        } catch (Exception e) {
            //发生异常时返回null。
            return null;
        }
    }
    private Object extract(Object json, String path) throws JSONException {
        // Cache patternkey.matcher(path).matches()
        Matcher mKey = null;
        Boolean mKeyMatches = mKeyMatchesCache.get(path);
        //如果缓存中不存在path，代码如下。
        if (mKeyMatches == null) {
            mKey = patternKey.matcher(path);
            //如果路径可以匹配到JSON字符串，返回True，如果无法匹配，返回False。
            mKeyMatches = mKey.matches() ? Boolean.TRUE : Boolean.FALSE;
            mKeyMatchesCache.put(path, mKeyMatches);
        }
        if (!mKeyMatches.booleanValue()) {
            return null;
        }
        // Cache mkey.group(1)
        //根据path获取对应的JSON。
        String mKeyGroup1 = mKeyGroup1Cache.get(path);
        //判断缓存中是否已存在JSON。
        if (mKeyGroup1 == null) {
            if (mKey == null) {
                mKey = patternKey.matcher(path);
            }
            mKeyGroup1 = mKey.group(1);
            // 根据path缓存对应的JSON。
            mKeyGroup1Cache.put(path, mKeyGroup1);
        }
        //获取JSON。
        json = extract_json_withkey(json, mKeyGroup1);
        // Cache indexList
        //获取对应的数组类型JSON。
        ArrayList<String> indexList = indexListCache.get(path);
        //判断缓存中是否已存在数组类型JSON。
```

```
        if (indexList == null) {
            Matcher mIndex = patternIndex.matcher(path);
            indexList = new ArrayList<String>();
            while (mIndex.find()) {
                indexList.add(mIndex.group(1));
            }
            indexListCache.put(path, indexList);
        }
        if (indexList.size() > 0) {
            //根据path缓存对应数组类型的JSON。
            json = extract_json_withindex(json, indexList);
        }
        return json;
    }
    private Object extract_json_withindex(Object json, ArrayList<String> indexList) throws JSONException {
        jsonList.clear();
        jsonList.add(json);
        Iterator<String> itr = indexList.iterator();
        while (itr.hasNext()) {
            String index = itr.next();
            //创建数组类型JSON对象。
            ArrayList<Object> tmp_jsonList = new ArrayList<Object>();
            //创建获取数组类型JSON的方法。
            if (index.equalsIgnoreCase("*")) {
                for (int i = 0; i < (jsonList).size(); i++) {
                    try {
                        JSONArray array = (JSONArray) (jsonList).get(i);
                        for (int j = 0; j < array.length(); j++) {
                            tmp_jsonList.add(array.get(j));
                        }
                    } catch (Exception e) {
                        continue;
                    }
                }
                jsonList = tmp_jsonList;
              //如果不存在通配符，则遍历JSON。
            } else {
                for (int i = 0; i < (jsonList).size(); i++) {
                    try {
                        tmp_jsonList.add(((JSONArray) (jsonList).get(i))
                                .get(Integer.parseInt(index)));
                    } catch (ClassCastException e) {
                        continue;
                    } catch (JSONException e) {
                        return null;
                    }
                    jsonList = tmp_jsonList;
                }
            }
        }
        return (jsonList.size() > 1) ? new JSONArray(jsonList) : jsonList
                .get(0);
    }
    //创建获取普通JSON的方法。
    private Object extract_json_withkey(Object json, String path) throws JSONException {
        if (json.getClass() == JSONArray.class) {
            JSONArray jsonArray = new JSONArray();
            for (int i = 0; i < ((JSONArray) json).length(); i++) {
                Object josn_elem = ((JSONArray) json).get(i);
                try {
                    Object json_obj = ((JSONObject) josn_elem).get(path);
                    if (json_obj.getClass() == JSONArray.class) {
                        for (int j = 0; j < ((JSONArray) json_obj).length(); j++) {
                            jsonArray.put(((JSONArray) json_obj).get(j));
                        }
                    } else {
                        jsonArray.put(json_obj);
                    }
                } catch (Exception e) {
                    continue;
                }
            }
            return (jsonArray.length() == 0) ? null : jsonArray;
        } else {
            return ((JSONObject) json).get(path);
```

```
        }
    }
}
```

3.10.2.5.5. 不同类型数据转换JSON STRING类型示例

本文为您介绍如何通过Java UDF实现不同类型数据转换为JSON STRING类型。

在您的数据使用过程中，常常需要将不同类型的数据转换为JSON STRING类型。例如，您的数据是ARRAY类型，由于Tunnel上传下载目前不支持复杂数据类型，可能需要先将ARRAY类型数据转换为JSON STRING类型，再进行下载。

转换JSONStringUDF

```
toJSONString()
```

- 函数功能：将MaxCompute对象（字段）转换为JSON字符串，包括必要的字符串转义。您可以与 `PutJsonValues` 配合使用。
- 参数说明：输入为任意类型的参数，但不支持DATETIME类型。

UDF使用示例

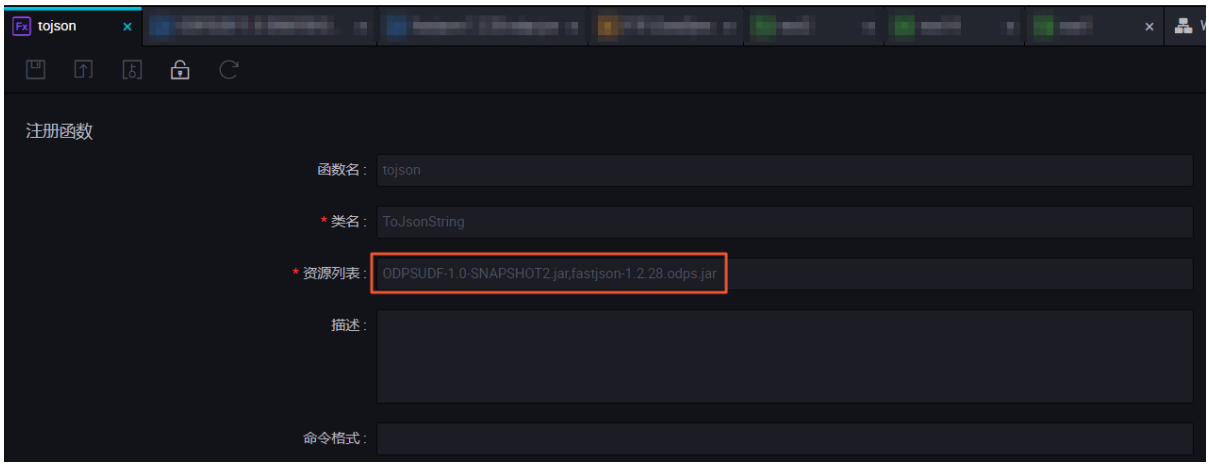
- 资源上传

  在MaxCompute公共云运行时，由于沙箱等限制，您需要把*fastjson.JSON*包和UDF JAR包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*fastjson.JSON*包。

- 注册函数

  *ToJsonStrin.java*通过测试后，将其注册为函数。

  本例中*ODPSUDF-1.0-SNAPSHOT2.jar*是UDF打包后生成的Jar，而*fastjson-1.2.28.odps.jar*是*fastjson.JSON*包。



- 使用示例

```
SELECT tojson(array(2.0,3.0,4.0));
```

> **说明** `tojson` 是您使用DataWorks或Intellij IDEA注册的函数名称。

运行结果如下。

```
[2.0,3.0,4.0]
```

UDF代码示例

```
import com.aliyun.odps.udf.UDF; //阿里云UDF。
import java.math.BigDecimal;
import com.alibaba.fastjson.JSON;  //引入fastjson.JSON类。
import java.util.Arrays;
import java.util.List;
public class ToJsonString extends UDF {
//下列代码调用fastjson.JSON的toJSONString方法，将各种类型的数据转换为JSON string，您可以根据您的需要增加更多的数据类型。
    public String evaluate(String obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(Long obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(Double obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(Boolean obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(BigDecimal obj) {
        return JSON.toJSONString(obj);
    }
//MaxCompute中Array类型数据在JAVA UDF中类型为List，例如Array<double>类型可以在UDF中通过List<Double>进行匹配。
    public String evaluate(List<Double> obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(double[] obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(int[] obj) {
        return JSON.toJSONString(obj);
    }
}
```

单元测试

```
//下列代码为测试代码，用于测试数据转换是否生效，您在实际使用过程中可注释掉。
    public static void main(String[] args) {
        //System.out.println(new ToJsonString().evaluate(null));//java不能调用，但MaxCompute可以。
        System.out.println(new ToJsonString().evaluate("中文\t"));
        System.out.println(new ToJsonString().evaluate(123L));
        System.out.println(new ToJsonString().evaluate(1.123));
        System.out.println(new ToJsonString().evaluate(true));
        //System.out.println(new ToJsonString().evaluate(new Date()));//udf/udaf不支持datetime类型。
        System.out.println(new ToJsonString().evaluate(BigDecimal.TEN));
        System.out.println(new ToJsonString().evaluate(Arrays.asList(1.0,2.0,3.0,4.0)));
        double[] args1 = {1,2,3,4};
        System.out.println(new ToJsonString().evaluate(args1));
       }
    }
```

3.10.2.5.6. JSON字符串增加键值示例

本文为您介绍如何通过Java UDF实现向基础JSON字符串中增加（覆盖）一个或多个键值对（支持嵌套JSON）。

JSON字符串增加键值UDF说明

```
PutJsonValues(String jsonBase,String Key1,String Value1,String Key2,String Value2...)
```

- 函数功能：向基础JSON串中增加（覆盖）一个或多个键值对支持嵌套JSON，接受奇数个参数。可以和不同类型数据转换JSON STRING类型示例中的`ToJsonObeject`配合使用（请先完成`ToJsonObeject`函数的注册）。
- 参数说明：
  - jsonBase：基础JSON串。
  - Key1：增加的Key1值，STRING类型。
  - Value1：增加的Key1对应的Value1值，STRING类型。
  - Key2：增加的Key2值，STRING类型。
  - Value2：增加的Key2对应的Value2值，STRING类型。

UDF使用示例

- 资源上传

  实际在MaxCompute公共云运行时，由于沙箱等限制，需要把*fastjson.JSON*包和UDF JAR包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*fastjson.JSON*包。

- 注册函数

  *PutJsonValues.java*测试通过后，将其注册函数。

  本例中*PutJsonValues.jar*是UDF打包后生成的JAR包，*ODPSUDF-1.0-SNAPSHOT2.jar*是上例 `ToJsonString` 打包后生成的JAR包，而*fastjson-1.2.28.odps.jar*是*fastjson.JSON*包。



> 说明
> - 本例的操作步骤详情请参见创建MaxCompute资源。
> - 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册三个步骤。您可以使用一键发布功能一次性完成这些步骤（MaxCompute Studio会依次执行 `mvn clean package` 、上传JAR包和注册UDF这三个步骤）。
> - 通过客户端使用常用命令进行资源上传操作，请参见资源操作。

- 使用示例

  成功注册UDF后，执行以下命令。

```
SELECT tojson(NULL)
,  tojson(concat('123', chr(3)))
,  tojson(123)
,  tojson(12.3)
,  tojson(true)
,  tojson(CAST(123  AS  DECIMAL))
,  putJsonValuesTest('{}', 'a', '123', 'b', tojson('abc'), 'c', '{"c1":567}')
;
```

运行结果：

```
null
"123\u0003"
123
12.3
true
123
{"a":123,"b":"abc","c":{"c1":567}}
```

> 说明
> - `null` 代表NULL。 `a` 的值是数值没有引号。 `b` 的值是字符串，需要使用 `tojson` 转义为JSON字符串。 `{"c1":567}` 本身已经是一个合法的JSON元素，因此可以直接传入。
> - `putJsonValues` 、 `tojson` 是您使用DataWorks或IntelliJ IDEA注册的函数名称。

UDF代码示例

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject; //引入fastjson.JSON类。
import com.aliyun.odps.udf.UDF;//阿里云UDF。
public class PutJsonValues extends UDF {
//MaxCompute中String类型数据与JAVA UDF中类型一致，String... keyValues为可变形参，表示参数个数不定。
    public  PutJsonValues(){
    }
    public  String evaluate(String jsonBase, String... keyValues) {
        if (jsonBase == null) {
            return null;
        }
        JSONObject outputObj = JSON.parseObject(jsonBase);
        for (int i = 1; i < keyValues.length; i = i + 2) {
            String key = keyValues[i - 1];
            String value = keyValues[i];
            if (key == null || value == null) {
                continue;
            }
            Object valueObj = JSON.parse(value);
            outputObj.put(key, valueObj);
        }
        return JSON.toJSONString(outputObj);
    }
}
```

单元测试

```
//下列代码为测试代码，用于测试数据转换是否生效，您在实际使用过程中可注释掉。
public static void main(String[] args) {
    PutJsonValues putJsonValues=new PutJsonValues();
    System.out.println(putJsonValues.hashCode());
    String js=putJsonValues.evaluate("{}", "k", "null");
    System.out.println(js);
    System.out.println(new PutJsonValues().evaluate("{}", "a", "123", "b", "234", "c", "345"));
    System.out.println(new PutJsonValues().evaluate("{}", "k", "\"abc\""));
    System.out.println(new PutJsonValues().evaluate("{}", "k", "{\"kk\":123}"));
}
```

3.10.2.5.7. 使用正则表达式替换字符串示例

本文为您介绍如何通过Java UDF实现使用正则表达式替换字符串。

正则表达式替换字符串UDF

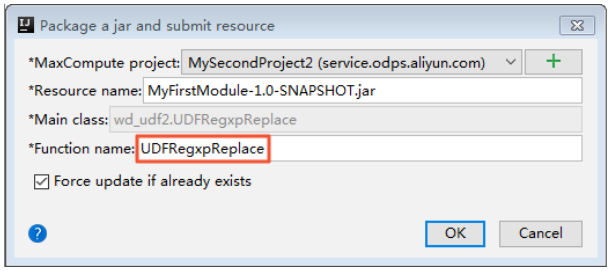
```
String UDFRegxpReplace(String s, String regex, String replacement)
```

- 函数功能：替换满足正则表达式的字符串。与MaxCompute的内建函数REGEXP_REPLACE函数相比，该函数中正则表达式支持变量。
- 参数说明：
  - s：源字符串，STRING类型。
  - regex：正则表达式，STRING类型。
  - replacement：替换字符串，将该字符串通过正则表达式替换源字符串，STRING类型。

UDF使用示例

- 注册函数

  *UDFRegxpReplace.java*测试通过后，将其注册为函数使用。



> **说明** 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行 `mvn clean package` 、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传及注册。

- 使用示例

  成功注册UDF后，执行以下命令。

```
select UDFRegxpReplace("foobar", "oo|ar", "") from dual;
```

  执行结果如下。

```
+-----+
| _c0 |
+-----+
| fb  |
+-----+
```

UDF代码示例

```
// package名称，可以根据您的情况定义。
package com.aliyun.odps.examples.udf;
import com.aliyun.odps.udf.annotation.UdfProperty;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
@UdfProperty(isDeterministic=true)
public class UDFRegxpRplace extends UDF {
    private String lastRegex = new String();
    private Pattern p = null;
    private String lastReplacement = new String();
    private String replacementString = "";
    public String evaluate(String s, String regex, String replacement) {
        if (s == null || regex == null || replacement == null) {
            return null;
        }
        // 如果正则表达式已更改，再次编译正则表达式。
        if (!regex.equals(lastRegex) || p == null) {
            lastRegex = regex;
            p = Pattern.compile(regex.toString());
        }
        Matcher m = p.matcher(s.toString());
        // 如果替换更改，再次执行toString。
        if (!replacement.equals(lastReplacement)) {
            lastReplacement = replacement;
            replacementString = replacement.toString();
        }
        StringBuffer sb = new StringBuffer();
        while (m.find()) {
            m.appendReplacement(sb, replacementString);
        }
        m.appendTail(sb);
        return sb.toString();
    }
}
```

3.10.2.5.8. 获取字符串（不含分隔符）Value示例

本文为您介绍如何获取不包含分隔符的字符串中指定Key对应的Value值。

获取字符串（不含分隔符）value的UDF说明

```
UDFKeyValue(String str, String splitor1, String splitor2, String key)
```

- 函数功能：从字符串中获得指定key的Value值。

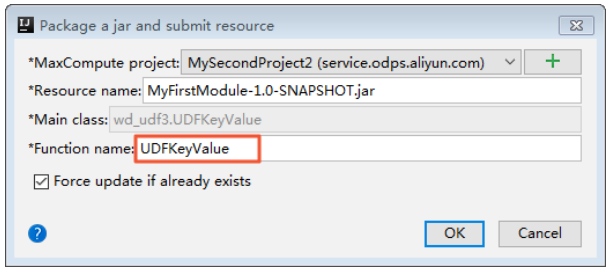
> **说明** 该UDF不适用于字符串本身包含分隔符的情况，如果需要处理该情况请使用UDFKeyValueEx。

- 参数说明
  - str：字符串。
  - splitor1：通过splitor1分割出 `key:value` 对。splitor1默认为 `;` 。
  - splitor2：对分割出来的 `Key:Value` 对，使用splitor2进行分割。splitor2默认为 `:` 。

UDF使用示例

1. 注册函数

   *UDFKeyValue.java*测试通过后，将其注册函数。

   > **说明** 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行 `mvn clean package` 、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传及注册。

2. 使用示例

   成功注册UDF后，执行以下命令：

   - 示例一

     ```
     select UDFKeyValue('a:1;b:2;','a');
     ```

     运行结果如下。

     ```
     +-----+
     | _c0 |
     +-----+
     | 1   |
     +-----+
     ```

   - 示例二

     ```
     select UDFKeyValue('a:1;b:2;','\;',':','a');
     ```

     运行结果如下。

     ```
     +-----+
     | _c0 |
     +-----+
     | 1   |
     +-----+
     ```

UDF代码示例

```
//package名称，可以根据您的情况定义。
package com.aliyun.odps.examples.udf;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
public class UDFKeyValue extends UDF{
        // 存储所有键：值对。
    private Map<String, String> mKeyValueache = new ConcurrentHashMap<String, String>();
    private Text result = new Text();
    public UDFKeyValue() {
    }
    public String evaluate(String str, String keyname){
        // 默认split1=; split2=:。
        return evaluate(str, ";", ":", keyname);
    }
    public String evaluate(String str, String split1, String split2, String keyname) {
        try {
                        // 通过splitor1分割出key:value对。
            if (str == null || "".equals(str))
                return null;
            String[] values1 = str.split(split1);
            mKeyValueache.clear();
            int i = 0;
                        // 对分割出来的key:value对，使用splitor2进行分割。
            while (i < values1.length) {
                storeKeyValue(values1[i], split2);
                i++;
            }
            String resultValue = getKeyValue(keyname);
            if (resultValue == null)
                return null;
            result.set(new Text(resultValue));
                        // 获取结果值。
            return result.toString();
        } catch (Exception e) {
            return null;
        }
    }
    private boolean storeKeyValue(String keyValues, String split) {
        if (keyValues == null || "".equals(keyValues))
            return false;
        if (mKeyValueache == null)
            mKeyValueache = new ConcurrentHashMap<String, String>();
        String[] keyValueArr = keyValues.split(split);
        if (keyValueArr.length == 2) {
            mKeyValueache.put(keyValueArr[0], keyValueArr[1]);
            return true;
        }
        return false;
    }
    private String getKeyValue(String keyName) {
        if (keyName == null ||
            "".equals(keyName) ||
            mKeyValueache == null ||
            mKeyValueache.size() == 0)
            return null;
        return mKeyValueache.get(keyName);
    }
}
```

3.10.2.5.9. 获取字符串（含有分隔符）Value示例

本文为您介绍如何通过Java UDF获取含有二级分隔符的字符串中指定Key对应的Value值。

获取字符串（含有分隔符）Value的UDF

```
UDFKeyValueEx(String str, String split1, String split2, String keyname)
```

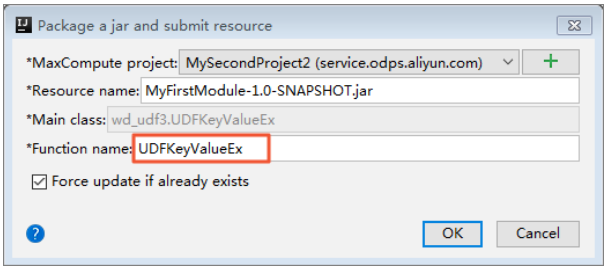
- 函数功能：取出Key所对应的Value值。先按一级分隔符进行分割，然后按二级分隔符进行分隔，再取出Key所对应的Value值。与UDFKeyValue主要的不同在于，该UDF适用于Value值本身含有二级分隔的情况。
- 参数说明：

- `str`：字符串。
- `split1`：一级分隔符，为正则表达式。默认一级分隔符为`;`。
- `split2`：二级分隔符，非正则表达式。默认二级分隔符为`:`。
- `keyname`：要获取的Key值。

UDF使用示例

1. 注册函数

   *UDFKeyValueEx.java*测试通过后，将其注册函数。

   > **说明**
   > - MaxCompute Studio提供了一键发布功能，即依次执行`mvn clean package`、上传JAR和注册UDF这三个步骤。您也可以使用DataWorks上传JAR包，详情请参见创建MaxCompute资源。
   > - 通过客户端使用常用命令进行资源上传操作，详情请参见资源操作。

2. 使用示例

   成功注册UDF后，执行以下命令：

   - 示例一

     ```
     select UDFKeyValueEx('a:b;c:d', ';', ':', 'a');
     ```

     运行结果如下。

     ```
     +-----+
     | _c0 |
     +-----+
     | b   |
     +-----+
     ```

   - 示例二

     ```
     select UDFKeyValueEx('a:b:c;c:d', ';', ':', 'a:b');
     ```

     运行结果如下。

     ```
     +-----+
     | _c0 |
     +-----+
     | c   |
     +-----+
     ```

UDF代码示例

```
// package名称，可以根据您的情况定义。
package com.aliyun.odps.examples.udf;
import com.taobao.bi.odps.udf.UDF;
public class UDFKeyValueEx extends UDF {
    public String evaluate(String str, String keyname) {
                // 默认分隔符split1=; split2=:。
        return evaluate(str, ";", ":", keyname);
    }
    public String evaluate(String str, String split1, String split2, String keyname) {
        try {
            if (str==null || split1==null || split2==null || keyname==null){
                return null;
            }
                        // 使用二级分隔符。
            String keySplit = keyname+split2;
                        // 遍历字符串。
            for (String subStr : str.split(split1)){
                if (subStr.startsWith(keySplit)){
                    if (keySplit.length() < subStr.length()){
                                                //返回结果值。
                        return subStr.substring(keySplit.length());
                    }else{
                        return null;
                    }
                }
            }
            // 没有Key对应的value。
            return null;
        } catch (Exception e) {
            return null;
        }
    }
}
```

3.10.2.5.10. 获取指定日期格式的时间示例

本文为您介绍如何通过evaluate()方法获取指定格式时间。

获取指定时间UDF说明

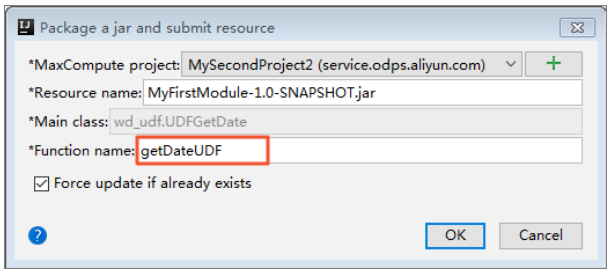
```
String UDFGetDate(String date, Long days)
String UDFGetDate(String date, Long months)
```

- 函数功能：获取指定间隔的时间。返回日期格式为yyyyMMdd、yyyy-MM-dd。
- 参数说明：
  - date：表示时间。格式为yyyyMMddHHmmss、yyyyMMdd、yyyy-MM-dd、yyyy-MM-dd HH:mm:ss。
  - days：表示间隔的天数。值可以为正或者负。
  - months：表示间隔月份。如果值为2，则表示求指定月份向后加两个月的月份；如果值为-2，则表示求指定月份向前减两个月的月份。

UDF使用示例

- 注册函数

  *UDFGetDate.java*测试通过后，将其注册为函数使用。



> **说明** 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行 `mvn clean package` 、上传JAR和注册UDF这三个步骤）。详情请参见打包、上传及注册。

- 使用示例

成功注册UDF后，执行以下命令：

- 示例一

```
select getDateUDF(20140503,-2);
```

运行结果如下。

```
+------+
| _c0 |
+------+
| 20140501 |
+------+
```

- 示例二

```
select getDateUDF("20100405",2,10);
```

运行结果如下。

```
+------+
+------+
| _c0 |
+------+
| 2010-06-10 |
+------+
```

- 示例三

```
select getDateUDF("20100405",2,'first');
```

运行结果如下。

```
+------+
| _c0 |
+------+
| 2010-06-01 |
+------+
---+
```

- 示例四

```
select getDateUDF("20100405",-2,"10");
```

运行结果如下。

```
+------+
| _c0 |
+------+
| 2010-02-10 |
+------+---+
```

UDF代码示例

```
package com.aliyun.odps.examples.udf; // package名称，可以根据您的情况定义。
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
import com.aliyun.odps.udf.annotation.UdfProperty;
@UdfProperty(isDeterministic=true)
public class UDFGetDate extends UDF{
  public String evaluate(String riqi, Long days) {
        try {
                        // 判断日期。
            String year = riqi.substring(0, 2);
            if (!"18".equals(year) && !"19".equals(year) && !"20".equals(year)
                    && !"21".equals(year)) {
                return "err";
            }
            String date = "";
            if (riqi.endsWith(".0")) {
                riqi = riqi.substring(0, riqi.length() - 2);
            }
            if (riqi.indexOf("-") >= 0) {
```

```
            riqi = riqi.substring(0, 10);
            riqi = riqi.replaceAll("-", "");
        } else {
            riqi = riqi.substring(0, 8);
        }
        try {
                            // 对指定日期进行处理：（+/-间隔天数）。
            date = getDay(riqi, days.intValue());
        } catch (Exception e) {
            return "err";
        }
        return date;
    } catch (Exception e) {
        return "err";
    }
}
public String evaluate(String riqi, Long n, String flag) {
    try {
                    // 判断日期。
        String year = riqi.substring(0, 2);
        if (!"18".equals(year) && !"19".equals(year) && !"20".equals(year)
                && !"21".equals(year)) {
            return "err";
        }
    } catch (Exception e) {
        return "err";
    }
    try {
        String date = "";
        if (riqi.endsWith(".0")) {
            riqi = riqi.substring(0, riqi.length() - 2);
        }
        if (riqi.indexOf("-") >= 0) {
            riqi = riqi.substring(0, 10);
            riqi = riqi.replaceAll("-", "");
        } else {
            riqi = riqi.substring(0, 8);
        }
        try {
                            // 如果flag为first，则日期为获取月份和当月的第一天。
            if ("first".equals(flag)) {
                date = getMonthStartDate(riqi, n.intValue());
            } else if ("last".equals(flag)) {
                                // 如果flag为last，则日期为获取月份和当月的最后一天。
                date = getMonthEndDate(riqi, n.intValue());
            } else {
                date = getMonthDate(riqi, n.intValue(), flag);
            }
        } catch (Exception e) {
            return "err";
        }
        return date;
    } catch (Exception e) {
        return "err";
    }
}
public String evaluate(Long n) {
    if (null == n)
        return "err";
    return getNmonthAgo(n.intValue());
}
private String getMonthDate(String riqi, int n, String flag) {
    Calendar ca = Calendar.getInstance();
    Date dtBegin = new Date();
    try {
        dtBegin = new SimpleDateFormat("yyyyMMdd").parse(riqi);
    } catch (ParseException e1) {
        e1.printStackTrace();
    }
    int day = Integer.parseInt(flag);
    ca.setTime(dtBegin);
    ca.add(Calendar.MONTH, n);
    ca.set(Calendar.HOUR_OF_DAY, 0);
    ca.set(Calendar.MINUTE, 0);
    ca.set(Calendar.SECOND, 0);
    ca.set(Calendar.DAY_OF_MONTH, day);
```

```
        ca.set(Calendar.DAY_OF_MONTH, day);
        Date firstDate = ca.getTime();
        return ymdFormat(firstDate);
    }
    private String getNmonthAgo(int n) {
        Calendar ca = Calendar.getInstance();
        Date dtBegin = new Date();
        ca.setTime(dtBegin);
        ca.add(Calendar.MONTH, n);
        Date firstDate = ca.getTime();
        return ymdhmsFormat(firstDate);
    }
        // 对指定日期进行处理：（+/-间隔天数）。
    private String getDay(String strBeginDate, int n) {
        Date dtBegin = new Date();
        try {
            dtBegin = new SimpleDateFormat("yyyyMMdd").parse(strBeginDate);
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
        Calendar cld = Calendar.getInstance();
        cld.setTime(dtBegin);
        int day = cld.get(Calendar.DAY_OF_YEAR);
        String strYear = strBeginDate.substring(0, 4);
        String strInputPath = "";
        String strDate = "";
        try {
            int nDays = 1;
            cld.setTime(dtBegin);
            cld.set(Calendar.DAY_OF_YEAR, day + n);
            Date dt = cld.getTime();
            strDate = new SimpleDateFormat("yyyyMMdd").format(dt);
            strYear = strDate.substring(0, 4);
            strInputPath = strDate;
        } catch (NumberFormatException e) {
            e.printStackTrace();
        }
        return strInputPath;
    }
        // 对指定日期进行设置flag处理。
    private String getMonthStartDate(String riqi, int n) {
        Calendar ca = Calendar.getInstance();
        Date dtBegin = new Date();
        try {
            dtBegin = new SimpleDateFormat("yyyyMMdd").parse(riqi);
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
        ca.setTime(dtBegin);
        ca.add(Calendar.MONTH, n);
        ca.set(Calendar.HOUR_OF_DAY, 0);
        ca.set(Calendar.MINUTE, 0);
        ca.set(Calendar.SECOND, 0);
        ca.set(Calendar.DAY_OF_MONTH, 1);
        Date firstDate = ca.getTime();
        return ymdFormat(firstDate);
    }
    private String getMonthEndDate(String riqi, int n) {
        Calendar ca = Calendar.getInstance();
        Date dtBegin = new Date();
        try {
            dtBegin = new SimpleDateFormat("yyyyMMdd").parse(riqi);
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
        ca.setTime(dtBegin);
        ca.add(Calendar.MONTH, n);
        ca.set(Calendar.HOUR_OF_DAY, 23);
        ca.set(Calendar.MINUTE, 59);
        ca.set(Calendar.SECOND, 59);
        ca.set(Calendar.DAY_OF_MONTH, 1);
        ca.add(Calendar.MONTH, 1);
        ca.add(Calendar.DAY_OF_MONTH, -1);
        Date lastDate = new Date(ca.getTime().getTime());
        return ymdFormat(lastDate);
    }
```

```
    private String ymdFormat(Date date) {
        if (date == null) {
            return "";
        }
                // 获取指定日期格式。
        DateFormat ymdFormat = new SimpleDateFormat("yyyy-MM-dd");
        return ymdFormat.format(date);
    }
    private String ymdhmsFormat(Date date) {
        if (date == null) {
            return "";
        }
                // 获取指定日期格式。
        DateFormat ymdFormat = new SimpleDateFormat("yyyyMMddHHmmss");
        return ymdFormat.format(date);
    }
}
```

3.10.2.5.11. 取余函数示例

本文为您介绍如何通过Java UDF实现不同类型数值相除获取余数。

取余函数UDF说明

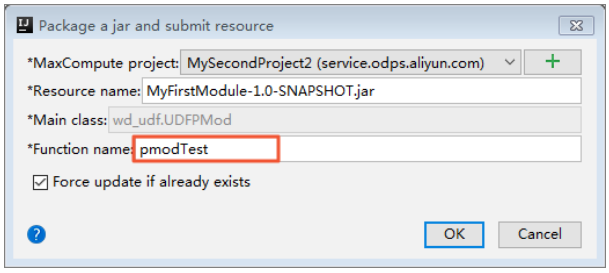
```
Integer UDFPMod(INTEGER a, INTEGER b)
Long UDFPMod(Long a, Long b)
Double UDFPMod(Double a, Double b)
```

- 函数功能：返回a除以b的余数。
- 参数说明：
  - a：被除数，支持INTEGER、Long以及Double类型。
  - b：除数，支持INTEGER、Long以及Double类型。

UDF使用示例

- 注册函数

  *UDFPMod.java*测试通过后，将其注册为函数使用。



  > **说明** 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行 `mvn clean package` 、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传及注册。

- 使用示例

  成功注册UDF后，执行以下命令：

  - 示例一

    ```
    select pmodTest(-9L,-4L);
    ```

    运行结果如下。

    ```
    +------+
    | _c0 |
    +------+
    | -1  |
    +------+
    ```

- 示例二

```
select pmodTest(9.0,-4.0);
```

运行结果如下。

```
+------+
| _c0 |
+------+
| -3.0 |
+------+
```

- 示例三

```
select pmodTest(9,4);
```

运行结果如下。

```
+------+
| _c0 |
+------+
| 1   |
+------+
```

UDF代码示例

```
package com.aliyun.odps.examples.udf;// package名称，可以根据您的情况定义。
import com.aliyun.odps.udf.UDF;
public class UDFPMod extends UDF {
    // 使用Long类型的数据。
    public String evaluate(Long  a,  Long  b)  {
        if  ((a  == null) || (b == null) || b==0L)  {
            return  null;
        }
        Long d=((a % b) + b) % b;
        return d.toString();
    }
    // 使用Double类型的数据。
    public String evaluate(Double a, Double b)  {
        if  ((a == null) || (b == null) || b==0.0)  {
            return  null;
        }
        Double d=((a % b) + b) % b;
        return d.toString() ;
    }
    // 使用Integer类型的数据。
    public String evaluate(Integer a, Integer b)  {
        if  ((a == null) || (b == null) || b==0)  {
            return  null;
        }
        Integer d=((a % b) + b) % b;
        return d.toString();
    }
}
```

> **说明** 示例中为返回值类型做了 `toString` 处理，对测试结果无影响。

UDF单元测试

```
package com.aliyun.odps.examples.udf// package名称，可以根据您的情况定义。
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestUDFPMod {
    private UDFPMod udf = new UDFPMod();
    @Test
    public void test_null(){
        assertEquals(null, udf.evaluate(null, 1L));
        assertEquals(null, udf.evaluate(1L, null));
        assertEquals(null, udf.evaluate((Long)null, null));
        assertEquals(null, udf.evaluate(null, 1.0));
        assertEquals(null, udf.evaluate(1.0, null));
        assertEquals(null, udf.evaluate((Double)null, null));
    }
    @Test
    public void test_正常情况(){
        assertEquals(1L, udf.evaluate(9L, 4L));
        assertEquals(3L, udf.evaluate(-9L, 4L));
        assertEquals(-1L, udf.evaluate(-9L, -4L));
        assertEquals(-3L, udf.evaluate(9L, -4L));
        // 原来抛异常，现返回null。
        assertEquals(null, udf.evaluate(9L, 0L));
        assertEquals(1.0, udf.evaluate(9.0, 4.0));
        assertEquals(3.0, udf.evaluate(-9.0, 4.0));
        assertEquals(-1.0, udf.evaluate(-9.0, -4.0));
        assertEquals(-3.0, udf.evaluate(9.0, -4.0));
        // 原来返回Nan，现返回null。
        assertEquals(null, udf.evaluate(9.0, 0.0));
    }
}
```

3.10.2.5.12. 获取URL中指定位置的字符示例

本文为您介绍如何使用indexOf()、substring()、lastIndexOf()等字符串处理方法获取URL中指定位置的字符。

获取URL中指定位置的字符UDF说明

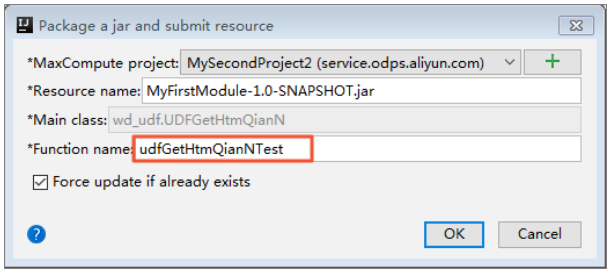
```
String UDFGetHtmQianN(String url, Long n)
```

- 函数功能：获取URL中指定位置的字符。
- 参数说明：
  - url：STRING类型的URL。
  - n：LONG类型长度值。

UDF使用示例

- 注册函数

  *UDFGetHtmQianN.java*测试通过后，将其注册为函数使用。



> **说明** 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行 `mvn clean package` 、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传及注册。

- 使用示例

  成功注册UDF后，执行以下命令：

- 示例一

```
select udfGetHtmQianNTest("http://www.taobao.com", 1);
```

运行结果如下。

```
+------+
| _c0 |
+------+
|     |
+------+
```

> **说明** 这里的返回结果不为NULL。

- 示例二

```
select udfGetHtmQianNTest("http://www.taobao.com/a.htm", 1);
```

运行结果如下。

```
+------+
| _c0 |
+------+
| a    |
+------+
```

- 示例三

```
select udfGetHtmQianNTest("http://www.taobao.com/a-b-c-d.htm", 3);
```

运行结果如下。

```
+------+
| _c0 |
+------+
| b    |
+------+
```

UDF代码示例

```
package com.aliyun.odps.examples.udf;// package名称，可以根据您的情况定义。
import com.aliyun.odps.udf.UDF;
public class UDFGetHtmQianN extends UDF {
    public String evaluate(String url, Long n) {
        try {
            //是取url字符串中第一次出现.htm时.的索引位置，数值类型为int。
            int index = url.indexOf(".htm");
            if (index < 0) {
                return "";
            }
            //从0开始，其中不包括index位置的字符。
            String a = url.substring(0, index);
            //返回指定字符/在此字符串中最后一次出现处的索引。
            index = a.lastIndexOf("/");
            // 取得的字符串长度为：a.length() - (index + 1)
            String b = a.substring(index + 1, a.length());
            //用"-"拆分字符串b，获取字符串数组。
            String[] c = b.split("-");
            //此处c.length>=n时，才有返回值。
            if (c.length < n) {
                return "";
            }
            if (n == 0) {
                return "";
            }
            //返回字符串数组中指定下标的字符。
            String d = c[c.length - n.intValue()];
            return d;
        } catch (Exception e) {
            return "err";
        }
    }
}
```

UDF单元测试

```
package com.aliyun.odps.examples.udf// package名称，可以根据您的情况定义。
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestUDFGetHtmQianN {
    private UDFGetHtmQianN udf = new UDFGetHtmQianN();
    @Test
    public void test_url(){
        Long n = 1L;
        assertEquals("err", udf.evaluate(null, n));
        //原来抛异常，现返回null。
        assertEquals("", udf.evaluate("", n));
        assertEquals("", udf.evaluate("http://www.taobao.com", n));
        assertEquals("a", udf.evaluate("http://www.taobao.com/a.htm", n));
        assertEquals("b", udf.evaluate("http://www.taobao.com/a-b.htm", n));
    }
    @Test
    public void test_index(){
        String url = "http://www.taobao.com/a-b-c.htm";
        assertEquals("err", udf.evaluate(url, null));
        //原来抛异常，现返回null。
        assertEquals("err", udf.evaluate(url, -1L));
        assertEquals("", udf.evaluate(url, 0L));
        assertEquals("c", udf.evaluate(url, 1L));
        //原来返回-， 现返回null。
        assertEquals("", udf.evaluate(url, 100L));
    }
}
```

3.10.2.6. Python UDF使用示例

3.10.2.6.1. 复杂数据类型示例

本文以在MaxCompute客户端操作为例，为您介绍如何通过Python UDF处理ARRAY、MAP或STRUCT复杂数据类型。

前提条件

已安装并配置MaxCompute客户端，更多信息请参见安装并配置MaxCompute客户端。

示例参考：`array<bigint>` 类型转换为 `array<datetime>` 类型

1. 将如下代码保存为PY脚本文件，例如array.py，并放置在MaxCompute客户端的 `bin` 目录中。

   ```
   from odps.udf import annotate
   import datetime
   @annotate('array<bigint>->array<datetime>')
   class ArrayExample:
       def evaluate(self, intput_list):
           output_list = list()
           for item in intput_list:
               t = datetime.datetime.fromtimestamp(item)
               output_list.append(t)
           return output_list
   ```

2. 登录MaxCompute客户端。
3. 执行如下命令，将PY脚本文件添加为MaxCompute资源。

   ```
   add py array.py;
   ```

   返回结果如下。

   ```
   OK: Resource 'array.py' have been created.
   ```

   更多添加资源命令信息，请参见添加资源。
4. 执行如下命令，注册Python UDF，即注册函数。

   ```
   create function array_udf as 'array.ArrayExample' using 'array.py';
   ```

   其中：

   - `array_udf` 表示注册的Python UDF名称，即后续在SQL语句中调用的自定义函数名称。
   - `array.ArrayExample` 中，`array` 表示array.py脚本文件的名称，`ArrayExample` 为array.py脚本文件中定义的类。

   返回结果如下。

```
Success: Function 'array_udf' have been created.
```

更多注册函数信息，请参见注册函数。

5. 构造测试数据并调用注册的函数，确保函数可正常处理数据。

```
--创建测试表。
create table array_test(col1 array<bigint>);
--插入数据。
insert into array_test values (array(1554047999)), (array(1554047989));
--调用在SQL语句中调用新注册的函数，处理表数据。
set odps.sql.python.version=cp37;
select array_udf(col1) from array_test;
```

返回结果如下。

```
+------+
| _c0  |
+------+
| [2019-03-31 23:59:49] |
| [2019-03-31 23:59:59] |
+------+
```

示例参考：`map<string,bigint>` 类型转换为 `map<string,datetime>` 类型

1. 将如下代码保存为PY脚本文件，例如map.py，并放置在MaxCompute客户端的 `bin` 目录中。

```
from odps.udf import annotate
import datetime
@annotate('map<string,bigint>->map<string,datetime>')
class MapExample:
    def evaluate(self, intput_dict):
        output_dict = dict()
        for key in intput_dict:
            value = intput_dict[key]
            t = datetime.datetime.fromtimestamp(value)
            output_dict[key] = t
        return output_dict
```

2. 登录MaxCompute客户端。
3. 执行如下命令，将PY脚本文件上传为MaxCompute资源。

```
add py map.py;
```

返回结果如下。

```
OK: Resource 'map.py' have been created.
```

更多添加资源命令信息，请参见添加资源。

4. 执行如下命令，注册Python UDF，即注册函数。

```
create function map_udf as 'map.MapExample' using 'map.py';
```

其中：

- `map_udf` 表示注册的Python UDF名称，即后续在SQL语句中调用的自定义函数名称。
- `map.MapExample` 中，`map` 表示map.py脚本文件的名称，`MapExample` 为map.py脚本文件中定义的类。

返回结果如下。

```
Success: Function 'map_udf' have been created.
```

更多注册函数信息，请参见注册函数。

5. 构造测试数据并调用注册的函数，确保函数可正常处理数据。

```
--创建测试表。
create table map_test(col1 map<string,bigint>);
--插入数据。
insert into map_test values (map('a', 1554047999)), (map('b',1554047989));
--调用在SQL语句中调用新注册的函数，处理表数据。
set odps.sql.python.version=cp37;
select map_udf(col1) from map_test;
```

返回结果如下。

```
+------+
| _c0  |
+------+
| {b:2019-03-31 23:59:49} |
| {a:2019-03-31 23:59:59} |
+------+
```

示例参考：`struct<input_name:string,input_timestamp:bigint>` 类型转换为 `struct<output_name:string,output_time:datetime>` 类型

1. 将如下代码保存为PY脚本文件，例如struct.py，并放置在MaxCompute客户端的 `bin` 目录中。

```
from odps.udf import annotate
import datetime, collections
@annotate('struct<input_name:string,input_timestamp:bigint>->struct<output_name:string,output_time:datetime>')
class StructExample:
    def evaluate(self, intput_namedtuple):
        OutputNamedTuple = collections.namedtuple('output_namedtuple', ['output_name', 'output_time'])
        name_val = intput_namedtuple.input_name
        time_val = datetime.datetime.fromtimestamp(intput_namedtuple.input_timestamp)
        output_namedtuple = OutputNamedTuple(name_val, time_val)
        return output_namedtuple
```

2. 登录MaxCompute客户端。
3. 执行如下命令，将PY脚本文件上传为MaxCompute资源。

```
add py struct.py;
```

返回结果如下。

```
OK: Resource 'struct.py' have been created.
```

更多添加资源命令信息，请参见添加资源。

4. 执行如下命令，注册Python UDF，即注册函数。

```
create function struct_udf as 'struct.StructExample' using 'struct.py';
```

其中：

- `struct_udf` 表示注册的Python UDF名称，即后续在SQL语句中调用的自定义函数名称。
- `struct.StructExample` 中，`struct` 表示struct.py脚本文件的名称，`StructExample` 为struct.py脚本文件中定义的类。

返回结果如下。

```
Success: Function 'struct_udf' have been created.
```

更多注册函数信息，请参见注册函数。

5. 构造测试数据并调用注册的函数，确保函数可正常处理数据。

```
--创建测试表。
create table struct_test(col1 struct<a:string, b:bigint>);
--插入数据。
insert into struct_test values (struct('date1',1554047999)), (struct('date2',1554047989));
--在SQL语句中调用新注册的函数，处理表数据。
set odps.sql.python.version=cp37;
select struct_udf(col1) from struct_test;
```

返回结果如下。

```
+------+
| _c0  |
+------+
| {output_name:date2, output_time:2019-03-31 23:59:49} |
| {output_name:date1, output_time:2019-03-31 23:59:59} |
+------+
```

3.10.2.6.2. 引用文件资源示例

本文以在MaxCompute客户端操作为例，为您介绍如何通过Python UDF引用文件资源。

前提条件

请确认您已完成如下操作：

- 已安装并配置MaxCompute客户端。

更多安装并配置MaxCompute客户端信息，请参见安装并配置MaxCompute客户端。

- 已将待引用的文件添加为MaxCompute项目中的资源。

  本文已添加的文件资源示例为test_distcache.txt，包含的数据如下。

  ```
  1 a
  2 b
  3 c
  4 d
  ```

  更多添加资源操作，请参见添加资源。

UDF代码示例

Python UDF代码如下，实现从引用的文件资源（例如test_distcache.txt）中返回满足要求的数据。

```
from odps.udf import annotate
from odps.distcache import get_cache_file
@annotate('bigint->string')
class DistCacheExample(object):
    def __init__(self):
        cache_file = get_cache_file('test_distcache.txt')
        kv = {}
        for line in cache_file:
            line = line.strip()
            if not line:
                continue
            k, v = line.split()
            kv[int(k)] = v
        cache_file.close()
        self.kv = kv
    def evaluate(self, arg):
        return self.kv.get(arg)
```

操作步骤

1. 将UDF代码示例保存为PY脚本文件（例如file.py），并放置在MaxCompute客户端的 `bin` 目录中。
2. 登录MaxCompute客户端。
3. 执行如下命令，将PY脚本文件上传为MaxCompute资源。

   ```
   add py file.py;
   ```

   返回结果如下。

   ```
   OK: Resource 'file.py' have been created.
   ```

   更多添加资源命令信息，请参见添加资源。
4. 执行如下命令，注册Python UDF，即注册函数。

   ```
   create function file_udf as 'file.DistCacheExample' using 'file.py, test_distcache.txt';
   ```

   其中：

   - `file_udf` 表示注册的Python UDF名称，即后续在SQL语句中调用的自定义函数名称。
   - `file.DistCacheExample` 中，`file` 表示file.py脚本文件的名称，`DistCacheExample` 为file.py脚本文件中定义的类。

   返回结果如下。

   ```
   Success: Function 'file_udf' have been created.
   ```

   更多注册函数信息，请参见注册函数。
5. 构造测试数据并调用注册的函数。

   ```
   --创建测试表。
   create table file_table (arg bigint);
   --插入数据。
   insert into file_table values (1), (4), (15), (123), (7995);
   --在SQL语句中调用新注册的函数，返回文件资源中满足要求的数据。
   select file_udf(arg) from file_table;
   ```

   返回结果如下。

```
+-----+
| _c0 |
+-----+
| a   |
| d   |
| NULL |
| NULL |
| NULL |
+-----+
```

3.10.2.6.3. 引用表资源示例

本文以在MaxCompute客户端操作为例，为您介绍如何通过Python UDF引用表资源。

前提条件

请确认您已完成如下操作：

- 已安装并配置MaxCompute客户端。

  更多安装并配置MaxCompute客户端信息，请参见安装并配置MaxCompute客户端。

- 已将待引用的表添加为MaxCompute项目中的资源。

  本文已添加的表资源示例为udf_test，包含的数据如下。

```
+------------+------+
| col1       | col2 |
+------------+------+
| 1          | a    |
| 2          | b    |
| 4          | c    |
| 5          | d    |
+------------+------+
```

  更多添加资源操作，请参见添加资源。

UDF代码示例

Python UDF代码如下，实现遍历引用的表资源（例如udf_test）中的数据，返回数组格式的数据。

```
from odps.udf import annotate
from odps.distcache import get_cache_table
@annotate('->string')
class DistCacheTableExample(object):
    def __init__(self):
        self.records = list(get_cache_table('udf_test'))
        self.counter = 0
        self.ln = len(self.records)
    def evaluate(self):
        if self.counter > self.ln - 1:
            return None
        ret = self.records[self.counter]
        self.counter += 1
        return str(ret)
```

操作步骤

1. 将UDF代码示例保存为PY脚本文件（例如table.py），并放置在MaxCompute客户端的 `bin` 目录中。
2. 登录MaxCompute客户端。
3. 执行如下命令，将PY脚本文件上传为MaxCompute资源。

   ```
   add py table.py;
   ```

   返回结果如下。

   ```
   OK: Resource 'table.py' have been created.
   ```

   更多添加资源命令信息，请参见添加资源。
4. 执行如下命令，注册Python UDF，即注册函数。

   ```
   create function table_udf as 'table.DistCacheTableExample' using 'table.py,udf_test';
   ```

   其中：

   - `table_udf` 表示注册的Python UDF名称，即后续在SQL语句中调用的自定义函数名称。

- `table.DistCacheTableExample` 中，`table` 表示table.py脚本文件的名称，`DistCacheTableExample` 为table.py脚本文件中定义的类。

返回结果如下。

```
Success: Function 'table_udf' have been created.
```

更多注册函数信息，请参见注册函数。

5. 构造测试数据并调用注册的函数。

```
--创建测试表。
create table table_test (arg bigint);
--插入数据。
insert into table_test values (1), (4), (15), (123), (7995);
--在SQL语句中调用新注册的函数。
select table_udf() from table_test;
```

返回结果如下。

```
+-----+
| _c0 |
+-----+
| (4, 'c') |
| (5, 'd') |
| (1, 'a') |
| (2, 'b') |
| NULL |
+-----+
```

3.10.2.6.4. Python UDF使用第三方包

MaxCompute支持您在Python UDF中引用第三方包，例如Numpy包、需要编译的第三方包或依赖动态链接库的第三方包。本文为您介绍如何通过Python UDF引用第三方包。

背景信息

通过Python UDF使用第三方包支持的场景如下：

- 使用Numpy包（Python 3 UDF）

  您需要修改Numpy包的后缀格式，基于MaxCompute客户端上传Numpy包，并注册函数。函数注册成功后即可通过Python 3 UDF调用。

- 使用需要编译的第三方包

  您需要在与MaxCompute兼容的环境下，对第三方资源包中的setup.py脚本进行编译生成WHEEL包，并修改后缀格式。基于MaxCompute客户端上传包，并注册函数。函数注册成功后即可通过Python UDF调用。推荐使用Linux环境，Windows用户推荐使用Docker。

- 使用依赖动态链接库的第三方包

  您需要基于第三方包的源码编译so链接库，然后编译生成WHEEL包，并修改后缀格式。基于MaxCompute客户端上传包和so链接库文件，并注册函数。函数注册成功后即可通过Python UDF调用。

前提条件

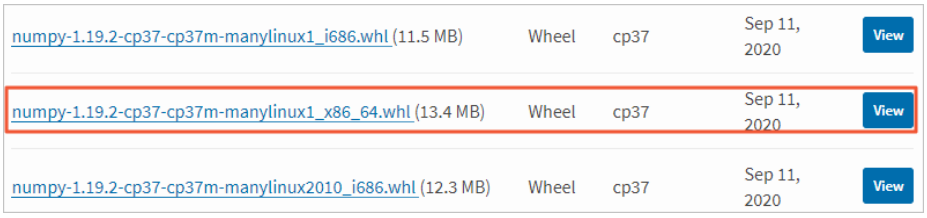
在执行操作前，请确认已完成如下操作：

- 已安装Python环境。推荐使用Python 3。
- 已安装并配置MaxCompute客户端。客户端配置详情请参见安装并配置MaxCompute客户端。
- 如果您通过Python UDF使用需要编译的第三方包，请确认已安装pip、setuptools（通过 `pip install setuptools` 安装）和wheel（通过 `pip install wheel` 安装）。
- 如果您使用的第三方包为GDAL 3.0及以上版本，请确认已安装PROJ 6。
- 如果您通过Docker编译第三方包，请确认已安装Docker，详情请参见Docker安装文档。

使用Numpy包（Python 3 UDF）

您可以通过MaxCompute内置的Python 3环境使用Numpy包。MaxCompute内置的Python 2环境默认安装了Numpy，不需要手动上传Numpy包。通过Python 3 UDF使用Numpy包的步骤如下：

1. 以1.19.2版本为例，在PyPI页面的**Download files**区域，单击文件名后缀为cp37-cp37m-manylinux1_x86_64.whl的Numpy包进行下载。



> **说明** 其它后缀名称的包可能会加载失败。如果您需要选择其他版本，在PyPI页面左上角的Navigation区域，单击Release history即可查看历史版本。

2. 修改下载的Numpy包后缀为ZIP格式。

   例如*numpy-1.19.2-cp37-cp37m-manylinux1_x86_64.zip*。

3. 通过MaxCompute客户端上传Numpy包至MaxCompute项目空间。上传资源详情请参见资源操作。

   命令示例如下：

```
ADD ARCHIVE D:\Downloads\numpy-1.19.2-cp37-cp37m-manylinux1_x86_64.zip -f;
```

4. 编写Python UDF脚本，保存为PY格式文件。

   假设此处保存的脚本名称为import_numpy.py。Python UDF脚本示例如下：

```
from odps.udf import annotate
@annotate("->string")
class TryImport(object): #类名为TryImport。
    def __init__(self):
        import sys
        sys.path.insert(0, 'work/numpy-1.19.2-cp37-cp37m-manylinux1_x86_64.zip') #Numpy包，您只需要替换work/后边的包名即可。
    def evaluate(self):
        import numpy
        return "import succeed"
```

5. 通过MaxCompute客户端将import_numpy.py脚本以资源形式上传至MaxCompute项目空间。

   命令示例如下：

```
ADD PY D:\Desktop\import_numpy.py -f;
```

6. 使用上传的import_numpy.py脚本及Numpy包，通过MaxCompute客户端注册自定义函数。注册函数详情请参见函数操作。

   假设注册的自定义函数名为numpy。命令示例如下：

```
CREATE FUNCTION numpy AS 'import_numpy.TryImport' USING 'doc_test_dev/resources/import_numpy.py,numpy-1.19.2-cp37-cp37m-manyl
inux1_x86_64.zip';
```

> **说明** 注册函数时资源列表里需要加上Numpy包，例如*numpy-1.19.2-cp37-cp37m-manylinux1_x86_64.zip*。

7. 完成注册后您即可编写SQL语句调用新建的自定义函数。执行SQL语句时需要开启 Python3，详情请参见Python 3 UDF。

使用需要编译的第三方包

如果第三方包是PyPI页面中格式为TAR.GZ的压缩包，或从GitHub下载的源码包，这些包解压后的根目录下有时会存在setup.py文件。在使用这种类型的第三方包前，您需要先在与MaxCompute兼容的环境下将setup.py编译生成WHEEL包，然后再执行上传资源及注册函数操作，即可通过Python UDF调用第三方包。上传资源及注册函数操作请参见使用Numpy包（Python 3 UDF）。

> **注意**
> - 由于第三方包是运行在Linux环境的，推荐您使用Linux环境编译第三方包，使用Windows环境编译会存在包不兼容的问题。
> - 如果您使用Windows环境，推荐在Docker的quay.io/pypa/manylinux2010_x86_64镜像容器中使用对应版本的Python（`/opt/python/cp27-cp27m/bin/python` 或 `/opt/python/cp37-cp37m/bin/python3`）编译生成WHEEL包。

以Linux环境为例，确保环境兼容的关注点如下：

- 需要使用兼容的Python版本。在系统的命令行窗口执行如下命令，检查环境Python版本。

```
python -c "import wheel.pep425tags; print(wheel.pep425tags.get_abi_tag())"
```

  - 如果返回值为 `cp27m` 或 `cp37m`，表示Python版本满足兼容性。
  - 如果返回值为 `cp27mu` 或 `cp37mu`，表示Python版本不满足兼容性。您需要在系统的命令行窗口执行 `./configure --enable-unicode=ucs2` 命令配置Python编码格式为UCS2。

- 如果涉及C或C++代码依赖，需要使用兼容的GCC（GNU Compiler Collection）版本。

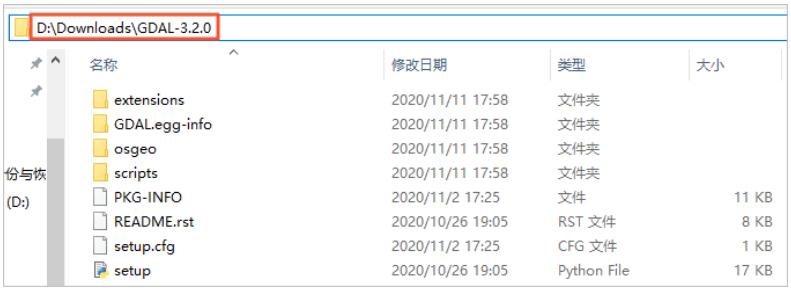
> **说明** 推荐您使用GCC 4.9.2及以下版本，GCC版本高于4.9.2时，编译生成的WHEEL包中的SO类型文件可能与MaxCompute环境不兼容。

确认环境满足兼容性要求后，以Linux系统为例，通过setup.py生成WHEEL包的步骤如下：

1. 将第三方包解压到本地，在系统的命令行窗口，切换路径至setup.py文件所在文件夹。

   例如，下载的包为*GDAL-3.2.0.zip*，解压后setup.py文件所在路径为*D:\Downloads\GDAL-3.2.0*，命令示例如下：

```
cd D:\Downloads\GDAL-3.2.0
```



2. 在系统的命令行窗口，执行如下命令查看返回结果中是否有bdist_wheel。

   命令示例如下：

   ```
   python setup.py --help-command
   ```

   - 如果返回结果中有bdist_wheel，请执行步骤3。
   - 如果返回结果中没有bdist_wheel，您需要修改setup.py脚本，将脚本中的 `from distutils.core import setup` 修改为 `from setuptools import setup` ，然后再执行步骤3。

3. 在系统的命令行窗口，执行如下命令编译生成WHEEL包。

   ```
   python setup.py bdist_wheel
   ```

   > **说明** WHEEL包在dist目录下。

使用依赖动态链接库的第三方包

部分Python第三方包除了依赖Python库，可能还会依赖其它动态链接库的依赖。以GDAL 3.0.4为例，为您介绍如何使用Docker的quay.io/pypa/manylinux2010_x86_64镜像容器，编译相关的so链接库，并编译生成可以在MaxCompute上使用的WHEEL包。基于生成的so链接库文件、WHEEL包或Numpy包，再执行上传资源及注册函数操作，即可通过Python UDF调用第三方包。上传资源及注册函数操作请参见使用Numpy包（Python 3 UDF）。

> **说明** 请确认您已安装Docker后再执行后续步骤。Docker操作详情请参见Docker文档。

通过Python UDF使用依赖so链接库的第三方包的操作步骤如下：

1. 查看依赖项。您可以在PyPI页面的**Dependencies**区域查看依赖项。

   例如GDAL 3.0.4的依赖项如下。

   Dependencies

   - *libgdal (3.0.4 or greater) and header files (gdal-devel)*
   - *numpy (1.0.0 or greater) and header files (numpy-devel) (not explicitly required, but many examples and utilities will not work without it)*

   > **说明** 在图示中，依赖项包括*libgda*和*numpy*。*libgdal*需要在镜像容器中通过编译GDAL源码得到，*numpy*需要在PyPI页面或通过Docker镜像容器下载Numpy包。

2. 下载Numpy包。

   您可以选择如下两种方式之一下载Numpy包：

   - 在PyPI页面的**Download files**区域，单击文件名后缀为*cp37-cp37m-manylinux1_x86_64.whl*的Numpy包进行下载。

     > **说明** 需要注意的是，如果您的Python环境为Python 2，Numpy包需要在PyPI页面左侧的**Navigation**区域，单击**Release history**，选择1.16.6及之前版本，且后缀为*cp27-cp27m-manylinux1_x86_64.whl*的包进行下载。

   - 在Docker的quay.io/pypa/manylinux2010_x86_64镜像容器中，执行 `/opt/python/cp37-cp37m/bin/pip download numpy -d ./` 命令下载Numpy包到当前目录。

3. 编译so链接库。

   i. 下载GDAL 3.0.4源码并解压到本地。

   ii. 使用Docker下载quay.io/pypa/manylinux2010_x86_64镜像容器，进入终端输入模式。

   命令示例如下：

   ```
   docker pull quay.io/pypa/manylinux2010_x86_64
   docker run -it quay.io/pypa/manylinux1_x86_64 /bin/bash
   ```

iii. 上传GDAL 3.0.4源码到镜像容器中。

命令示例如下：

```
docker cp ./gdal-3.0.4 <CONTAINER ID>:/opt/source/
```

CONTAINER ID获取方式请参见docker ps。

4. 在镜像容器中编译GDAL 3.0.4，操作详情请参见BuildingOnUnix。

命令示例如下：

```
# configure选项中需要指定安装PROJ 6的位置。
./configure --prefix=/path/to/install/prefix --with-proj=/path/to/install/proj6/prefix
make
make install
export PATH=/path/to/install/prefix/bin:$PATH
export LD_LIBRARY_PATH=/path/to/install/prefix/lib:$LD_LIBRARY_PATH
export GDAL_DATA=/path/to/install/prefix/share/gdal
# Test
gdalinfo --version
```

编译过程中可能出现如下报错：

- `configure: error: PROJ 6 symbols not found`：GDAL 3.0及以上版本依赖PROJ 6，需要您下载安装PROJ 6。
- `fatal error: zlib.h: No such file or directory`：改用 `yum install zlib-devel` 命令编译。

5. 使用Docker下载命令将两个so链接库（非软链接）下载到本机，在GDAL、PROJ 6安装目录的lib文件夹下获取*libgdal.so*和*libproj.so*。
6. 在镜像容器中制作GDAL WHEEL包。操作详情请参见BuildingOnUnix。

命令示例如下：

```
# 如果需要numpy支持，要先安装numpy。
/opt/python/cp37-cp37m/bin/pip install numpy
# 切换到GDAL源码目录下。
cd swig/python
# 生成WHEEL包，在dist目录下。WHEEL包示例：GDAL-3.0.4-cp37-cp37m-linux_x86_64.whl
/opt/python/cp37-cp37m/bin/python setup.py bdist_wheel
```

7. 基于生成的so链接库文件、WHEEL包或Numpy包，再执行上传资源及注册函数操作，即可实现通过Python UDF使用第三方包。上传资源及注册函数操作请参见使用Numpy包（Python 3 UDF）。

需要注意的是：

- 在上传资源时，您需要以FILE资源形式上传*libgdal.so*和*libproj.so*，以ARCHIVE资源形式上传*numpy-1.19.2-cp37-cp37m-manylinux1_x86_64.zip*和*GDAL-3.0.4-cp37-cp37m-linux_x86_64.zip*。
- 在注册函数时，您需要在函数资源列表中添加*libgdal.so*、*libproj.so*、*numpy-1.19.2-cp37-cp37m-manylinux1_x86_64.zip*和*GDAL-3.0.4-cp37-cp37m-linux_x86_64.zip*。

Python UDF代码示例如下：

```
# coding: utf-8
from odps.udf import annotate
from odps.distcache import get_cache_file
def include_file(file_name):
    import os, sys
    so_file = get_cache_file(file_name, 'b')
    with open(so_file.name, 'rb') as fp:
        content=fp.read()
        so = open(file_name, "wb")
        so.write(content)
        so.flush()
        so.close()
@annotate("->string")
class TryImport(object):
    def __init__(self):
        import sys
        include_file('libgdal.so.26')
        include_file('libproj.so.15')
        sys.path.insert(0, 'work/GDAL-3.0.4-cp37-cp37m-linux_x86_64.zip') #编译后的GDAL包，您只需要替换work/后边的包名即可。
        sys.path.insert(0, 'work/numpy-1.19.2-cp37-cp37m-manylinux1_x86_64.zip') #Numpy包，您只需要替换work/后边的包名即可。
    def evaluate(self):
        from osgeo import gdal
        from osgeo import ogr
        from osgeo import osr
        from osgeo import gdal_array
        from osgeo import gdalconst
        return "import succeed"
```

> **说明** 运行时如果报错找不到*libgdal.so.26*、*libproj.so.15*，您需要修改*libgdal.so*、*libproj.so*为*libgdal.so.26*、*libproj.so.15*。

3.10.3. UDTF

3.10.3.1. UDTF概述

MaxCompute支持您通过Java、Python语言编写代码创建UDTF，扩展MaxCompute的函数能力，满足个性化业务需求。本文为您介绍UDTF类型、使用限制、注意事项、开发流程及使用说明等信息。

背景信息

UDTF为用户自定义表值函数，适用于一进多出业务场景。即其输入与输出是一对多的关系，读入一行数据，输出多个值可视为一张表。MaxCompute支持您通过如下两种代码语言编写UDTF。

UDTF类型	说明
Java UDTF	使用Java语言编写UDTF逻辑实现代码。更多信息，请参见Java UDTF。
Python UDTF	使用Python语言编写UDTF逻辑实现代码，支持Python 2和Python 3： • Python 2 UDTF：Python语言版本为2.7。更多信息，请参见Python 2 UDTF。 • Python 3 UDTF：Python语言版本为CPython-3.7.3。更多信息，请参见Python 3 UDTF。

MaxCompute支持的内建函数中包含部分UDTF函数，例如EXPLODE，更多内建UDTF函数信息，请参见其他函数或复杂类型函数。

使用限制

- 不支持通过自定义函数访问外网。如果您需要通过自定义函数访问外网，请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。
- 在 `select` 语句中使用UDTF时，不允许存在其他列或表达式。**错误示例**如下。

```
--查询语句中同时携带了UDTF和其他列。
select value, user_udtf(key) as mycol ...
```

- UDTF不能嵌套使用。**错误示例**如下。

```
--user_udtf1嵌套了user_udtf2，不允许嵌套。
select user_udtf1(user_udtf2(key)) as mycol...;
```

- 不支持在同一个 `select` 子句中与 `group by` 、 `distribute by` 、 `sort by` 联用。**错误示例**如下。

```
--UDTF不能与group by联用。
select user_udtf(key) as mycol ... group by mycol;
```

注意事项

使用自定义函数时，您需要注意：

- 在性能上，自定义函数的性能低于内建函数，建议您优先使用内建函数实现相同逻辑的业务需求。
- 在SQL语句中使用自定义函数时，如果计算的数据量过大并且存在倾斜，会导致作业占用的内存超出默认分配的内存。此时，您可以在Session级别设置 `set odps.sql.udf.joiner.jvm.memory=xxxx;` 属性来解决此问题。更多自定义函数常见问题，请参见MaxCompute UDF常见问题。
- 当自定义函数的名称与内建函数的名称相同时，自定义函数会覆盖同名的内建函数。例如，MaxCompute中存在一个名称为CONCAT的自定义函数，则系统默认会调用自定义的CONCAT，不会调用内建函数CONCAT。如果您希望调用内建函数，需要在内建函数前增加 `::` 符号，例如 `select ::concat('ab', 'c');` 。

开发流程

UDTF的开发流程如下。

- 当您使用Java代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档

序号	可选/必选步骤	功能介绍	工具平台	相关文档

①	可选	当您使用Maven编写代码时，需要先在Pom文件中添加代码相关SDK依赖，确保后续编写的代码可编译成功。例如开发自定义函数需要添加的SDK依赖为：
		`<dependency>`
		`<groupId>com.aliyun.odps</groupId>`
		`<artifactId>odps-sdk-udf</artifactId>`
		`<version>0.29.10-public</version>`
		`</dependency>`
		您可以从Maven库中搜索 `odps-sdk-udf` ，获取SDK依赖版本信息。
			IntelliJ IDEA（Maven）	无

②	必选	根据业务需求，编写自定义函数代码。	IntelliJ IDEA（Maven）、MaxCompute Studio	开发Java UDF
③	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		
④	必选	调试自定义函数代码，确保本地运行成功后打包为JAR包。		
⑤	必选	将JAR包作为资源上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	○ MaxCompute客户端 ■ 添加资源 ■ 注册函数 ○ MaxCompute Studio 打包、上传及注册 ○ DataWorks ■ 创建JAR资源 ■ 注册MaxCompute函数
⑥	必选	基于上传的JAR包资源创建自定义函数。		
⑦	可选	在查询数据代码中调用自定义函数。		无

- 当您使用Python代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档
①	必选	根据业务需求，编写自定义函数代码。	MaxCompute Studio	开发Python UDF
②	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		
③	必选	将Python文件或依赖的资源（例如文件资源、表资源、第三方包）上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	○ MaxCompute客户端 ■ 添加资源 ■ 注册函数 ○ MaxCompute Studio 上传及注册 ○ DataWorks 创建Python资源并注册函数
④	必选	基于上传的Python文件或依赖的资源创建自定义函数。		
⑤	可选	在查询数据代码中调用自定义函数。		无

使用说明

自定义函数的使用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例： `select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

3.10.3.2. Java UDTF

本文为您介绍如何通过Java语言编写UDTF。

UDTF代码结构

您可以通过IntelliJ IDEA（Maven）或MaxCompute Studio工具使用Java语言编写UDTF代码，代码中需要包含如下信息：

- Java包（Package）：可选。

  您可以将定义的Java类打包，为后续查找和使用类提供方便。

- 继承UDTF类：必选。

  必须携带的UDTF类为 `com.aliyun.odps.udf.UDTF` 、 `com.aliyun.odps.udf.annotation.Resolve` （对应 `@Resolve` 注解）和 `com.aliyun.odps.udf.UDFException` （对应实现Java类的方法）。当您需要使用其他UDTF类或者需要用到复杂数据类型时，请根据MaxCompute SDK添加需要的类。

- 自定义Java类：必选。

  UDTF代码的组织单位，定义了实现业务需求的变量及方法。

- `@Resolve` 注解：必选。

  格式为 `@Resolve(<signature>)` 。 `signature` 为函数签名，用于定义函数的输入参数和返回值的数据类型。UDTF无法通过反射分析获取函数签名，只能通过 `@Resolve` 注解方式获取函数签名，例如 `@Resolve("smallint->varchar(10)")` 。更多 `@Resolve` 注解信息，请参见@Resolve注解。

- 实现Java类的方法：必选。

  Java类实现包含如下4个方法，您可以根据实际需要进行选择。

接口定义	描述
`public void setup(ExecutionContext ctx) throws UDFException`	初始化方法，在UDTF处理输入的数据前，MaxCompute会调用用户自定义的初始化行为。在每个Worker内 `setup` 会被先调用一次。
`public void process(Object[] args) throws UDFException`	SQL中每一条记录都会对应调用一次 `process` ， `process` 的参数为SQL语句中指定的UDTF输入参数。输入参数以 `Object[]` 的形式传入，输出结果通过 `forward` 函数输出。您需要在 `process` 函数内自行调用 `forward` ，以决定输出数据。
`public void close() throws UDFException`	UDTF的结束方法。只会被调用一次，即在处理完最后一条记录之后被调用。
`public void forward(Object …o) throws UDFException`	调用 `forward` 方法输出数据，每调用一次 `forward` 代表输出一条记录。在SQL查询语句中调用UDTF时，可以通过 `as` 子句将 `forward` 输出的结果进行重命名。

  编写Java UDTF时可以使用Java Type或Java Writable Type，MaxCompute项目支持处理的数据类型与Java数据类型的详细映射关系，请参见数据类型。

UDTF代码示例如下。

```
//将定义的Java类组织在org.alidata.odps.udtf.examples包中。
package org.alidata.odps.udtf.examples;
//继承UDTF类。
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDTFCollector;
import com.aliyun.odps.udf.annotation.Resolve;
import com.aliyun.odps.udf.UDFException;
//自定义Java类。
//@Resolve注解。
@Resolve("string,bigint->string,bigint")
public class MyUDTF extends UDTF {
     //实现Java类的方法。
     @Override
     public void process(Object[] args) throws UDFException {
         String a = (String) args[0];
         Long b = (Long) args[1];
         for (String t: a.split("\\s+")) {
         forward(t, b);
       }
     }
   }
```

使用限制

- 不支持通过自定义函数访问外网。如果您需要通过自定义函数访问外网，请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

- 在 `select` 语句中使用UDTF时，不允许存在其他列或表达式。**错误示例**如下。

```
--查询语句中同时携带了UDTF和其他列。
select value, user_udtf(key) as mycol ...
```

- UDTF不能嵌套使用。**错误示例**如下。

```
--user_udtf1嵌套了user_udtf2，不允许嵌套。
select user_udtf1(user_udtf2(key)) as mycol...;
```

- 不支持在同一个 `select` 子句中与 `group by` 、 `distribute by` 、 `sort by` 联用。**错误示例**如下。

```
--UDTF不能与group by联用。
select user_udtf(key) as mycol ... group by mycol;
```

注意事项

在编写Java UDTF时，您需要注意：

- 不同UDTF JAR包中不建议存在类名相同但实现逻辑不一样的类。例如UDTF1、UDTF2分别对应资源JAR包udtf1.jar、udtf2.jar，两个JAR包里都包含名称为 `com.aliyun.UserFunction.class` 的类但实现逻辑不一样，当同一条SQL语句中同时调用UDTF1和UDTF2时，MaxCompute会随机加载其中一个类，此时会导致UDTF执行结果不符合预期甚至编译失败。
- Java UDTF中输入或返回值的数据类型是对象，数据类型首字母必须大写，例如String。
- SQL中的NULL值通过Java中的NULL表示。Java Primitive Type无法表示SQL中的NULL值，不允许使用。

@Resolve注解

`@Resolve` 注解格式如下。

```
@Resolve(<signature>)
```

`signature` 为函数签名字符串，用于标识输入参数和返回值的数据类型。执行UDTF时，UDTF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type_list'
```

其中：

- `type_list` ：表示返回值的数据类型。UDTF可以返回多列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。
- `arg_type_list` ：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

  `arg_type_list` 还支持星号（*）或为空（''）：

  - 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
  - 当 `arg_type_list` 为空（''）时，表示无输入参数。

合法 `@Resolve` 注解示例如下。

@Resolve注解示例	说明
`@Resolve('bigint,boolean->string,datetime')`	输入参数类型为BIGINT、BOOLEAN，返回值类型为STRING、DATETIME。
`@Resolve('*->string, datetime')`	输入任意个参数，返回值类型为STRING、DATETIME。
`@Resolve('->double, bigint, string')`	无输入参数，返回值类型为DOUBLE、BIGINT、STRING。
`@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")`	输入参数类型为ARRAY、STRUCT、MAP，返回值类型为MAP、STRUCT。

数据类型

在MaxCompute中不同数据类型版本支持的数据类型不同。从MaxCompute 2.0版本开始，扩展了更多的新数据类型，同时还支持ARRAY、MAP、STRUCT等复杂类型。更多MaxCompute数据类型版本信息，请参见数据类型版本说明。

为确保编写Java UDTF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute Type	Java Type	Java Writable Type
TINYINT	java.lang.Byte	ByteWritable
SMALLINT	java.lang.Short	ShortWritable

MaxCompute Type	Java Type	Java Writable Type
INT	java.lang.Integer	IntWritable
BIGINT	java.lang.Long	LongWritable
FLOAT	java.lang.Float	FloatWritable
DOUBLE	java.lang.Double	DoubleWritable
DECIMAL	java.math.BigDecimal	BigDecimalWritable
BOOLEAN	java.lang.Boolean	BooleanWritable
STRING	java.lang.String	Text
VARCHAR	com.aliyun.odps.data.Varchar	VarcharWritable
BINARY	com.aliyun.odps.data.Binary	BytesWritable
DATETIME	java.util.Date	DatetimeWritable
TIMESTAMP	java.sql.Timestamp	TimestampWritable
INTERVAL_YEAR_MONTH	不涉及	IntervalYearMonthWritable
INTERVAL_DAY_TIME	不涉及	IntervalDayTimeWritable
ARRAY	java.util.List	不涉及
MAP	java.util.Map	不涉及
STRUCT	com.aliyun.odps.data.Struct	不涉及

> 说明 当MaxCompute项目采用MaxCompute 2.0数据类型版本时，UDTF的输入或返回值才可以使用Java Writable Type。

使用说明

按照开发流程，完成Java UDTF开发后，您即可通过MaxCompute SQL调用Java UDTF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;`。更多跨项目分享信息，请参见基于Package跨项目访问资源。

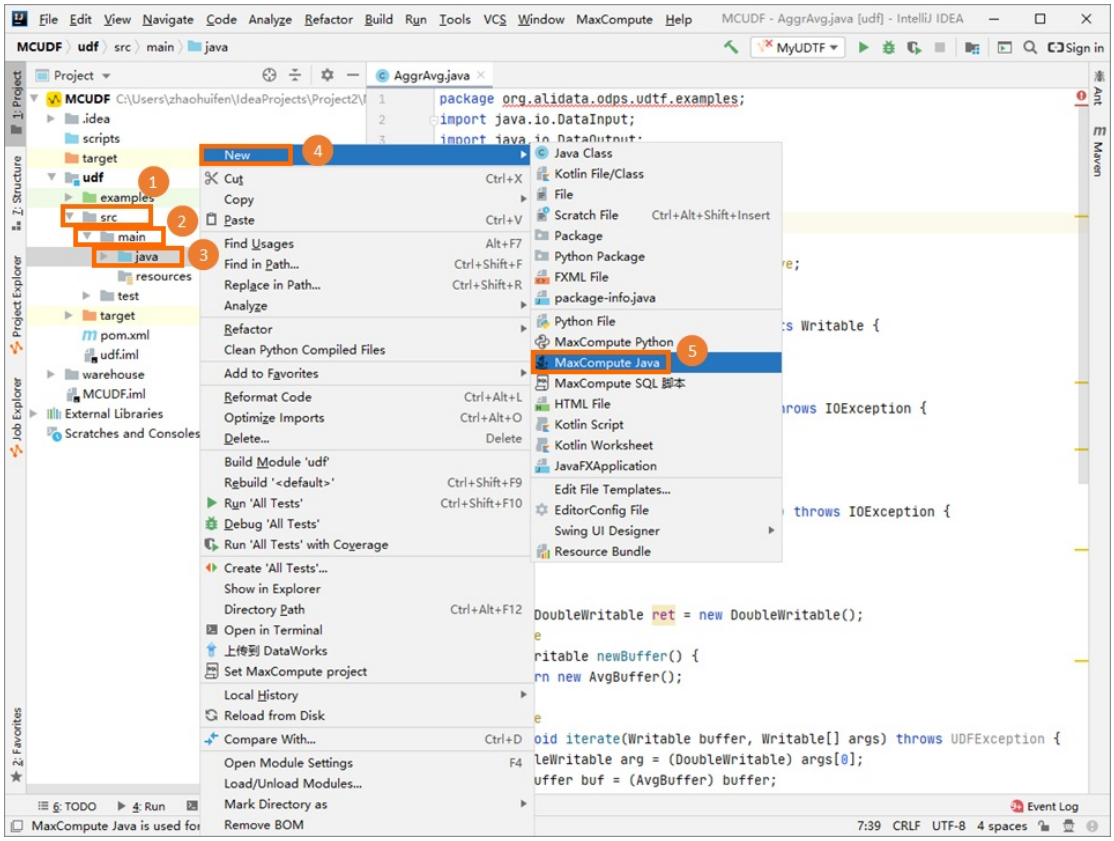
使用MaxCompute Studio完整开发及调用Java UDTF的操作，请参见使用示例。

使用示例

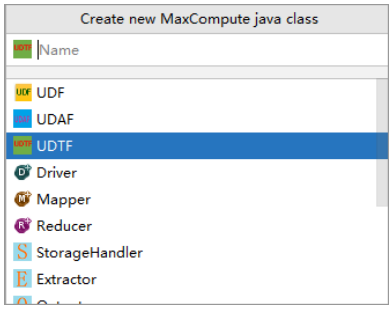
以通过MaxCompute Studio开发UDTF为例，开发并调用Java UDTF的操作步骤如下：

1. 在IntelliJ IDEA上完成如下准备工作：
    i. 安装MaxCompute Studio
    ii. 创建MaxCompute项目连接
    iii. 创建MaxCompute Java Module
2. 编写UDTF代码。

i. 在**Project**区域，右键单击Module的源码目录（即**src > main > java**），选择**new > MaxCompute Java**。

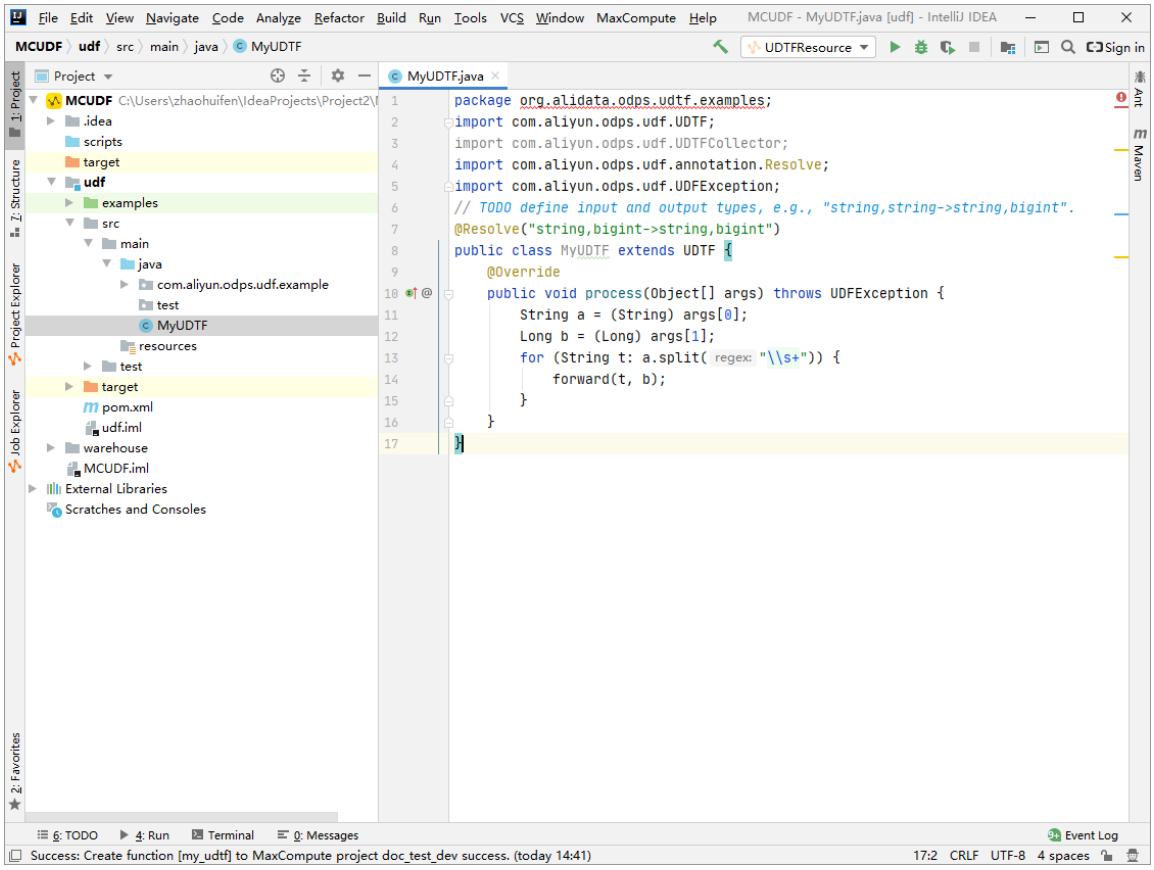


ii. 在**Create new MaxCompute java class**对话框，单击**UDTF**并填写**Name**后，按Enter键。例如Java Class名称为MyUDTF。



Name为创建的MaxCompute Java Class名称。如果还没有创建Package，在此处填写packagename.classname，会自动生成Package。

iii. 在代码编写区域写入如下代码。

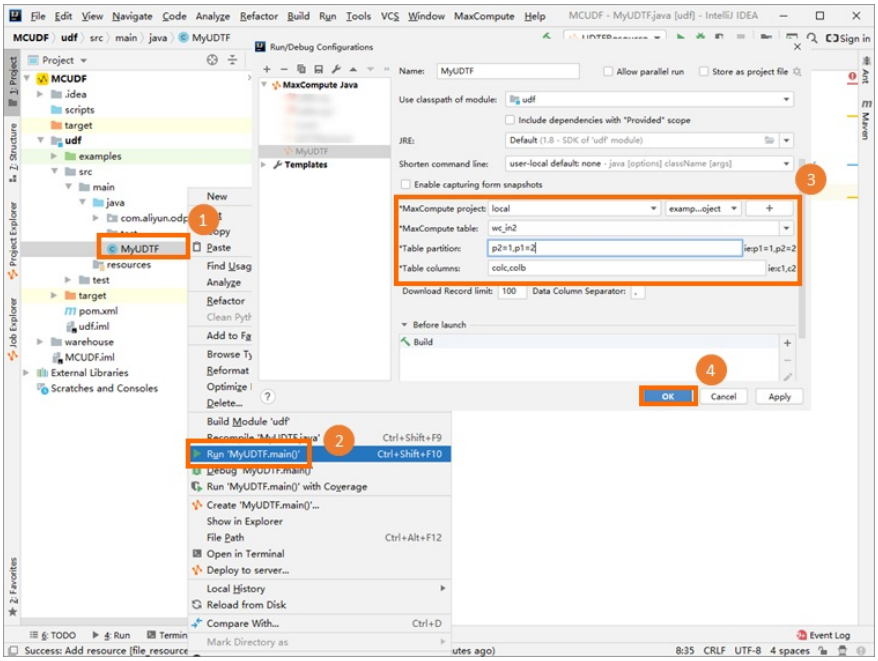


UDTF代码示例如下。

```
package org.alidata.odps.udtf.examples;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDTFCollector;
import com.aliyun.odps.udf.annotation.Resolve;
import com.aliyun.odps.udf.UDFException;
// TODO define input and output types, e.g., "string,string->string,bigint".
   @Resolve("string,bigint->string,bigint")
   public class MyUDTF extends UDTF {
     @Override
     public void process(Object[] args) throws UDFException {
       String a = (String) args[0];
       Long b = (Long) args[1];
       for (String t: a.split("\\s+")) {
         forward(t, b);
       }
     }
   }
```

3. 在本地运行调试UDTF，确保代码可以运行成功。

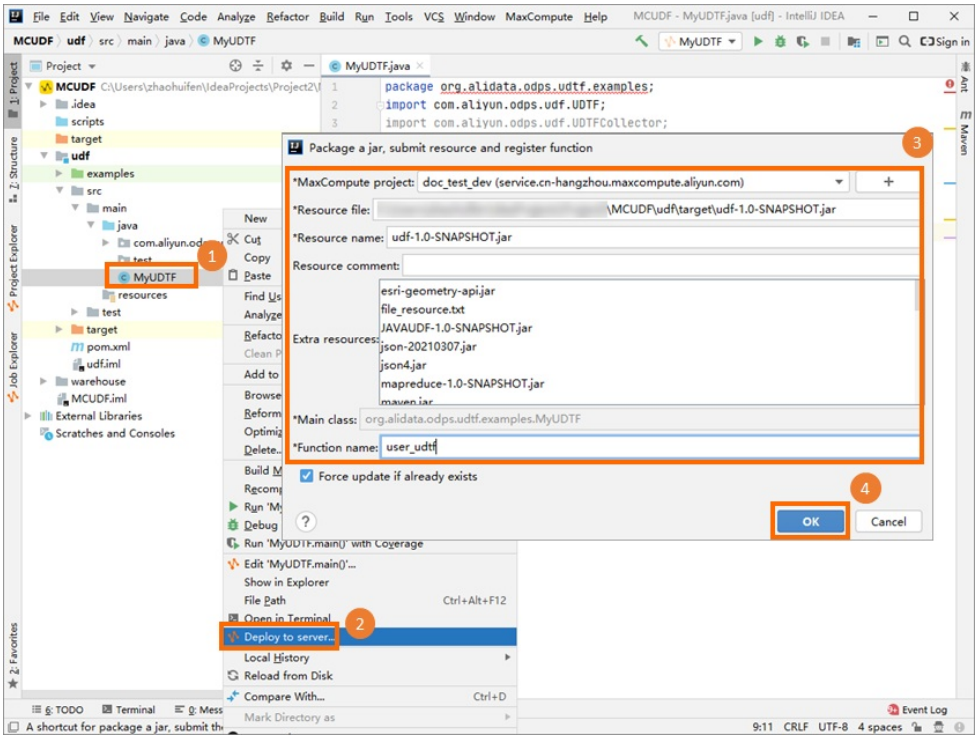
   更多调试操作，请参见通过本地运行调试UDF。



> **说明** 运行参数可参照图示数据填写。

4. 将创建的UDTF打包为JAR包，上传至MaxCompute项目并注册函数。例如函数名称为 `user_udtf` 。

   更多打包操作，请参见操作步骤。



5. 在MaxCompute Studio的左侧导航栏，单击**Project Explorer**，在目标MaxCompute项目上单击右键，启动MaxCompute客户端，并执行SQL命令调用新创建的UDTF。

   假设待查询目标表my_table的数据结构如下。

```
+------------+------------+
| col0       | col1       |
+------------+------------+
| A B        | 1          |
| C D        | 2          |
+------------+------------+
```

执行如下SQL命令调用UDTF。

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
```

返回结果如下。

```
+----+------------+
| c0 | c1         |
+----+------------+
| A  | 1          |
| B  | 1          |
| C  | 2          |
| D  | 2          |
+----+------------+
```

3.10.3.3. Java UDTF使用示例

3.10.3.3.1. Java UDTF读取MaxCompute资源示例

本文为您介绍基于MaxCompute Studio通过Java UDTF读取MaxCompute资源的使用示例。

前提条件

已安装MaxCompute Studio，并连接至MaxCompute项目，创建了MaxCompute Java Module。

更多操作信息，请参见安装MaxCompute Studio、管理项目连接和创建MaxCompute Java Module。

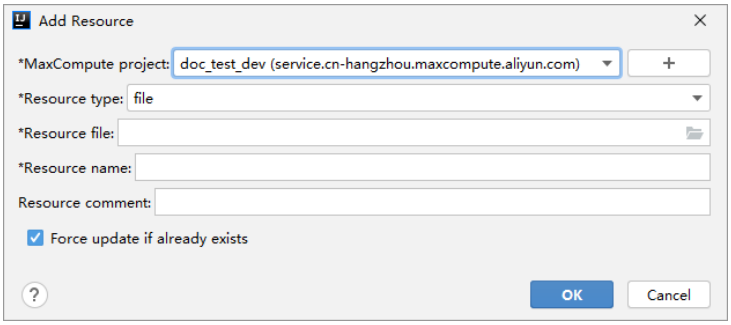
更多MaxCompute资源信息，请参见资源。

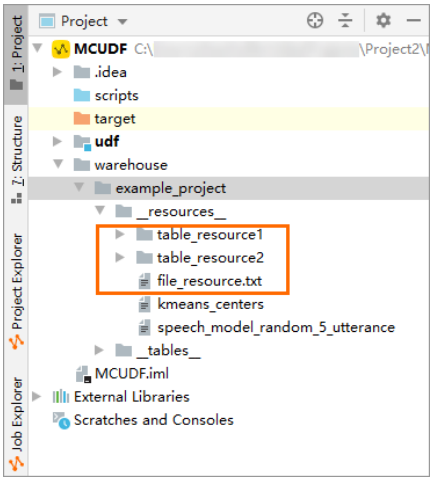
UDTF代码示例

```
package com.aliyun.odps.examples.udf;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Iterator;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.annotation.Resolve;
/**
 * project: example_project
 * table: wc_in2
 * partitions: p2=1,p1=2
 * columns: colc,colb
 */
@Resolve("string,string->string,bigint,string")
public class UDTFResource extends UDTF {
  ExecutionContext ctx;
  long fileResourceLineCount;
  long tableResource1RecordCount;
  long tableResource2RecordCount;
  @Override
  public void setup(ExecutionContext ctx) throws UDFException {
  this.ctx = ctx;
  try {
   InputStream in = ctx.readResourceFileAsStream("file_resource.txt");
   BufferedReader br = new BufferedReader(new InputStreamReader(in));
   String line;
   fileResourceLineCount = 0;
   while ((line = br.readLine()) != null) {
     fileResourceLineCount++;
   }
   br.close();
   Iterator<Object[]> iterator = ctx.readResourceTable("table_resource1").iterator();
   tableResource1RecordCount = 0;
   while (iterator.hasNext()) {
     tableResource1RecordCount++;
     iterator.next();
   }
   iterator = ctx.readResourceTable("table_resource2").iterator();
   tableResource2RecordCount = 0;
   while (iterator.hasNext()) {
     tableResource2RecordCount++;
     iterator.next();
   }
 } catch (IOException e) {
   throw new UDFException(e);
 }
}
   @Override
   public void process(Object[] args) throws UDFException {
     String a = (String) args[0];
     long b = args[1] == null ? 0 : ((String) args[1]).length();
     forward(a, b, "fileResourceLineCount=" + fileResourceLineCount + "|tableResource1RecordCount="
     + tableResource1RecordCount + "|tableResource2RecordCount=" + tableResource2RecordCount);
    }
}
```

操作步骤

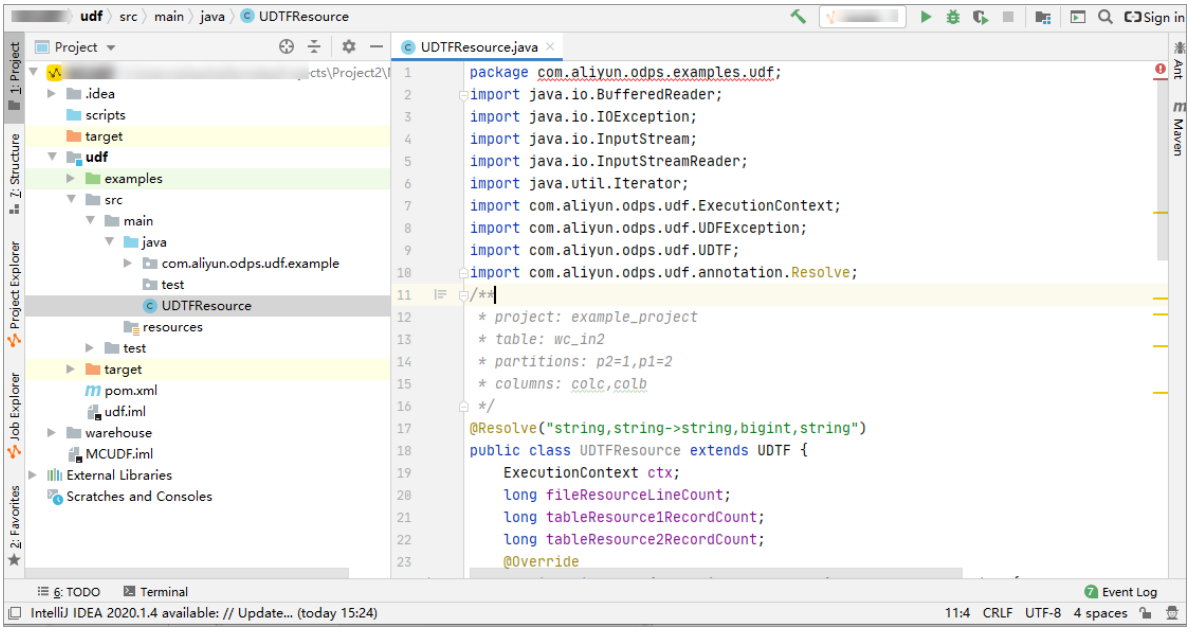
1. 在MaxCompute Studio的顶部菜单栏，选择**MaxCompute > 添加资源**，添加如下资源文件。



资源文件	资源类型	资源内容
file_resource.txt	file	您可以在MaxCompute客户端的如下位置获取示例资源文件。 
table_resource1	table	
table_resource2	table	

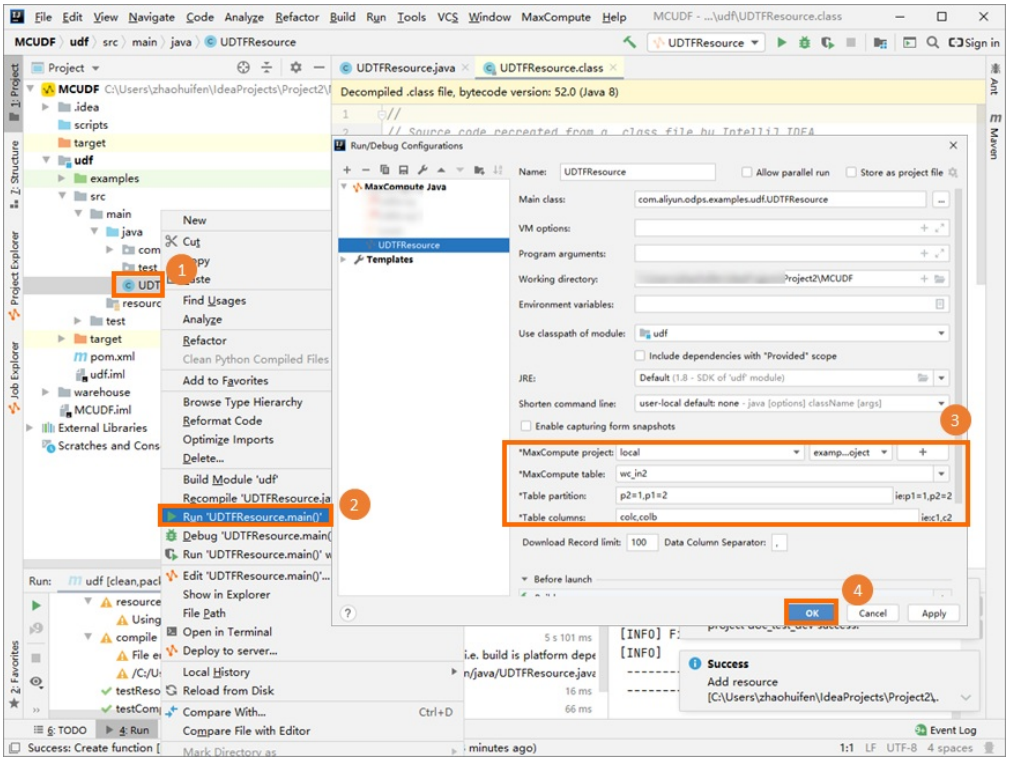
2. 在MaxCompute Studio中新建UDTF类型的Java程序。例如Java Class名称为 `UDTFResource` ，程序代码为UDTF代码示例中的代码。

   更多创建UDTF操作，请参见编写UDF。



3. 在本地运行调试UDTF，确保代码可以运行成功。

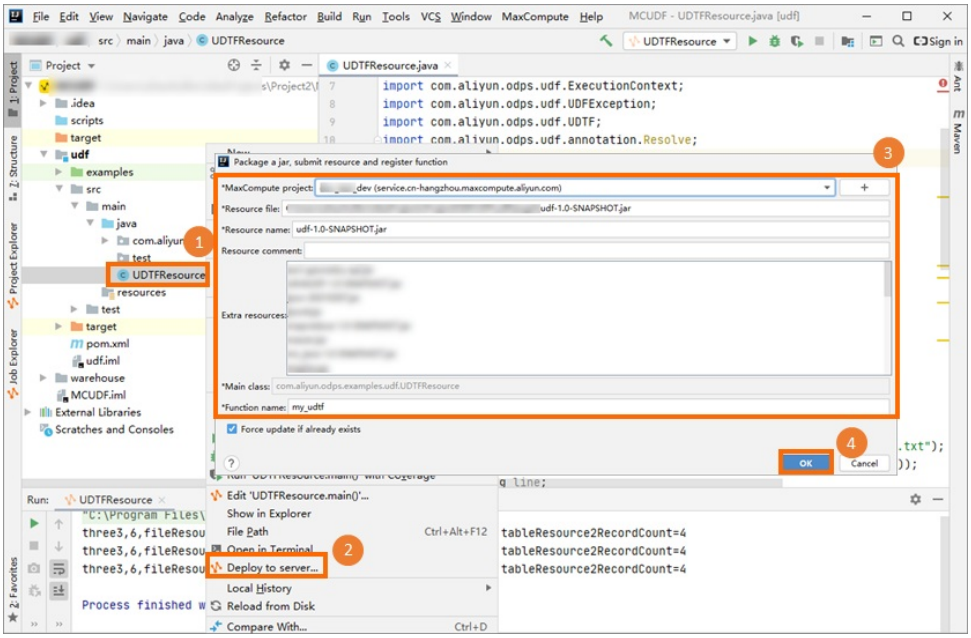
   更多调试操作，请参见通过本地运行调试UDF。



> **说明** 运行参数可参照图示数据填写。

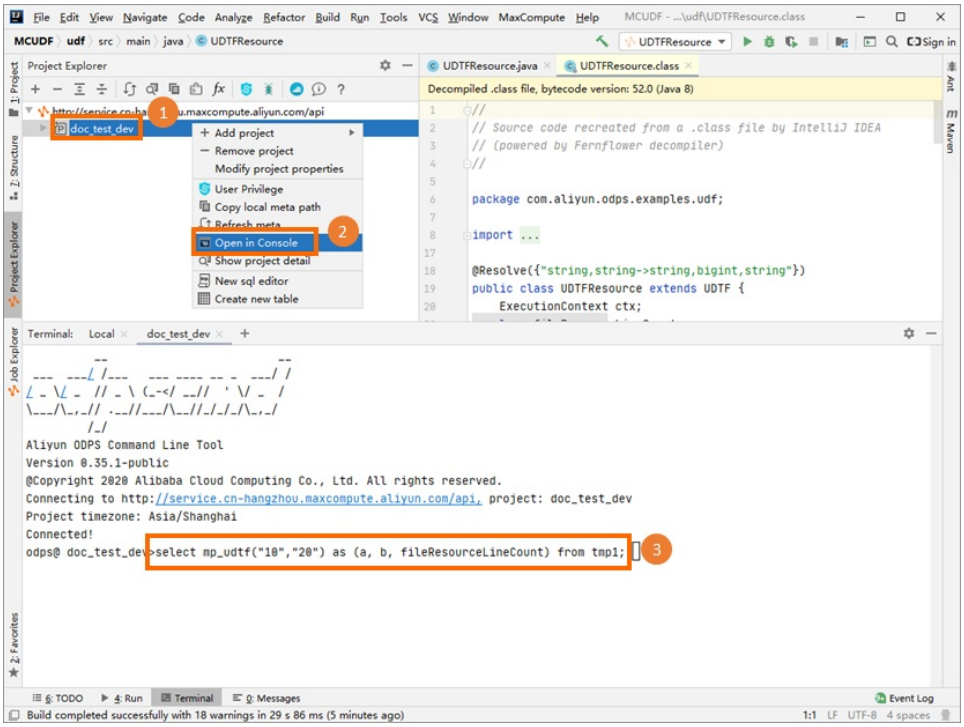
4. 将创建的UDTF打包为JAR包，上传至MaxCompute项目并注册函数。例如函数名称为 `my_udtf` 。

   更多打包操作，请参见操作步骤。



   Extra resources需要选中步骤1中添加的三个资源文件。

5. 在MaxCompute Studio的左侧导航栏，单击**Project Explorer**，在目标MaxCompute项目上单击右键，启动MaxCompute客户端，并执行SQL命令调用新创建的UDTF。



SQL代码示例如下。

```
select my_udtf("10","20") as (a, b, fileResourceLineCount);
```

返回结果如下。

```
+-------+------------+--------+
| a | b      | fileResourceLineCount |
+-------+------------+--------+
| 10    | 2          | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordCount=0 |
| 10    | 2          | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordCount=0 |
+-------+------------+--------+
```

3.10.3.4. Python UDTF

3.10.3.4.1. Python 2 UDTF

MaxCompute使用的Python 2版本为2.7。本文为您介绍如何通过Python 2语言编写UDTF。

UDTF代码结构

您可以通过MaxCompute Studio工具使用Python 2语言编写UDTF代码，代码中需要包含如下信息：

- 编码声明：可选。

  固定声明格式为 `#coding:utf-8` 或 `# -*- coding: utf-8 -*-` ，二者等效。当Python 2代码中出现中文字符时，运行程序会报错，您需要在代码头部增加编码声明。

- 导入模块：必选。

  至少要包含 `from odps.udf import annotate` 和 `from odps.udf import BaseUDTF` 。 `from odps.udf import annotate` 用于导入函数签名模块，MaxCompute才可以识别后续代码中定义的函数签名。 `from odps.udf import BaseUDTF` 为Python UDTF的基类，您需要通过此类在派生类中实现 `process` 或 `close` 等方法。

  当UDTF代码中需要引用文件资源或表资源时，需要包含 `from odps.distcache import get_cache_file` （文件资源）或 `from odps.distcache import get_cache_table` （表资源）。

- 函数签名：可选。

  格式为 `@annotate(<signature>)` ， `signature` 用于定义函数的输入参数和返回值的数据类型。如果不指定函数签名，在SQL中调用UDTF时，可以匹配任意类型的输入参数，但返回值类型无法推导，所有输出参数都将会是STRING类型。更多函数签名信息，请参见函数签名及数据类型。

- 自定义Python类（派生类）：必选。

  UDTF代码的组织单位，定义了实现业务需求的变量及方法。您还可以在代码中引用MaxCompute内置的第三方库或引用文件、表资源。更多信息，请参见第三方库或引用资源。

- 实现Python类的方法：必选。

Python类实现包含如下4个方法，您可以根据实际需要进行选择。

方法定义	描述
`BaseUDTF.init()`	初始化方法。派生类如果需要实现此方法，必须在一开始调用基类的初始化方法 `super(BaseUDTF, self).init()` 。 `init` 方法在整个UDTF生命周期中只会被调用一次，即在处理第一条记录之前。当UDTF需要保存内部状态时，可以通过此方法初始化所有状态。
`BaseUDTF.process([args, ...])`	SQL中每一条记录都会对应调用一次 `process` ， `process` 的参数为SQL语句中指定的UDTF输入参数。
`BaseUDTF.forward([args, ...])`	UDTF的输出方法。此方法由用户代码调用。每调用一次 `forward` ，就会输出一条记录。 `forward` 的参数为SQL语句中指定的UDTF的输出参数。 如果Python代码中未指定函数签名，在调用 `forward` 方法时，必须将所有输出值转换为STRING类型。
`BaseUDTF.close()`	UDTF的结束方法。只会被调用一次，即在处理完最后一条记录之后被调用。

UDTF代码示例如下。

```
#coding:utf-8
#导入函数签名模块及基类。
from odps.udf import annotate
from odps.udf import BaseUDTF
#函数签名。
@annotate('string -> string')
#自定义Python类。
class Explode(BaseUDTF):
#实现Python类的方法。
   def process(self, arg):
       props = arg.split(',')
       for p in props:
           self.forward(p)
```

使用限制

MaxCompute Python 2 UDTF使用的Python版本为2.7，并以沙箱模式执行用户代码，即代码是在一个受限的运行环境中执行的。在该环境中，以下行为会被禁止：

- 读写本地文件。
- 启动子进程。
- 启动线程。
- 使用Socket通信。
- 其他系统调用。

基于上述原因，您上传的代码都必须通过标准Python实现。Python标准库中涉及到上述功能的模块或C扩展模块都会被禁止使用。具体标准库的可用模块说明如下：

- 所有基于标准Python实现（不依赖扩展模块）的模块都可用。
- C扩展模块中下列模块可用：
  - array、audioop
  - binascii、bisect
  - cmath、_codecs_cn、_codecs_hk、_codecs_iso2022、_codecs_jp、_codecs_kr、_codecs_tw、_collections、cStringIO
  - datetime
  - _functools、future_builtins、
  - _heapq、_hashlib
  - itertools
  - _json
  - _locale、_lsprof
  - math、_md5、_multibytecodec
  - operator
  - _random
  - _sha256、_sha512、_sha、_struct、strop
  - time
  - unicodedata
  - _weakref
  - cPickle

- 沙箱限制了您的代码最多可向标准输出和标准错误输出写入数据的大小为20 KB，即 `sys.stdout/sys.stderr` 最多能写入20 KB数据，多余的字符会被忽略。

第三方库

MaxCompute的Python 2运行环境中安装了除Python标准库外比较常用的第三方库，作为标准库的补充，例如Numpy。

> 说明 使用第三方库存在限制，例如禁止本地访问、网络I/O受限，因此第三方库中涉及到相关功能的API也被禁止使用。

函数签名及数据类型

函数签名格式如下。

```
@annotate(<signature>)
```

`signature` 为函数签名字符串，用于标识输入参数和返回值的数据类型。执行UDTF时，UDTF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type_list'
```

其中：

- `type_list` ：表示返回值的数据类型。UDTF可以返回多列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。
- `arg_type_list` ：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

  `arg_type_list` 还支持星号（*）或为空（''）：
  - 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
  - 当 `arg_type_list` 为空（''）时，表示无输入参数。

> 说明 在编写UDTF代码过程中，您可以根据MaxCompute项目的数据类型版本选取合适的数据类型，更多数据类型版本及各版本支持的数据类型信息，请参见数据类型版本说明。

合法的函数签名示例如下。

函数签名示例	说明
`@annotate('bigint,boolean->string,datetime')`	输入参数类型为BIGINT、BOOLEAN，返回值类型为STRING、DATETIME。
`@annotate('*->string, datetime')`	输入任意个参数，返回值类型为STRING、DATETIME。
`@annotate('->double, bigint, string')`	无输入参数，返回值类型为DOUBLE、BIGINT、STRING。
`@annotate("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")`	输入参数类型为ARRAY、STRUCT、MAP，返回值类型为MAP、STRUCT。

为确保编写Python UDTF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute SQL Type	Python 2 Type
BIGINT	INT
STRING	STR
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	INT
FLOAT	FLOAT
CHAR	STR
VARCHAR	STR
BINARY	BYTEARRAY

MaxCompute SQL Type	Python 2 Type
DATE	INT
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT
STRUCT	COLLECTIONS.NAMEDTUPLE

> 说明
> - DATETIME类型对应的Python类型是INT，值为Epoch UTC Time起至今的毫秒数。您可以通过Python标准库中的DATETIME模块处理日期时间类型。
> - `odps.udf.int(value,[silent=True])` 增加了参数 `silent` 。当 `silent` 为True时，如果 `value` 无法转为INT，则会返回None（不会返回异常）。
> - NULL值对应Python的None。

引用资源

Python UDTF可以通过 `odps.distcache` 模块引用资源，支持引用文件资源和表资源。

- `odps.distcache.get_cache_file(resource_name)` ：返回指定文件资源的内容。
  - `resource_name` 为STRING类型，对应当前MaxCompute项目中已存在的文件资源名。如果文件资源名非法或者没有相应的文件资源，会返回异常。

    > 说明 使用UDTF访问资源，在创建UDTF时需要声明引用的资源，否则会报错。

  - 返回值为File-like对象。在使用完此对象后，您需要调用 `close` 方法释放打开的资源文件。
- `odps.distcache.get_cache_table(resource_name)` ：返回指定表资源的内容。
  - `resource_name` 支持STRING类型，对应当前MaxCompute项目中已存在的表资源名。如果表资源名非法或者没有相应的表资源，会返回异常。
  - 返回值为GENERATOR类型，调用者以遍历方式获取表的内容，每次遍历可得到以数组形式存在的表中的一条记录。

引用文件资源和表资源的代码示例如下。

```
# -*- coding: utf-8 -*-
from odps.udf import annotate
from odps.udf import BaseUDTF
from odps.distcache import get_cache_file
from odps.distcache import get_cache_table
@annotate('string -> string, bigint')
class UDTFExample(BaseUDTF):
    """读取资源文件和资源表里的pageid、adid，生成dict
    """
    def __init__(self):
        import json
        cache_file = get_cache_file('test_json.txt')
        self.my_dict = json.load(cache_file)
        cache_file.close()
        records = list(get_cache_table('table_resource1'))
        for record in records:
            self.my_dict[record[0]] = [record[1]]
    """输入pageid，输出pageid以及它对应的所有adid
    """
    def process(self, pageid):
        for adid in self.my_dict[pageid]:
            self.forward(pageid, adid)
```

使用说明

按照开发流程，完成Python 2 UDTF开发后，您即可通过MaxCompute SQL调用Python 2 UDTF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例： `select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute Studio完整开发及调用Python 2 UDTF的操作，请参见开发Python UDF。

3.10.3.4.2. Python 3 UDTF

Python官方即将停止维护Python 2，MaxCompute已支持Python 3，对应版本为CPython-3.7.3。本文为您介绍如何通过Python 3语言编写UDTF。

UDTF代码结构

您可以通过MaxCompute Studio工具使用Python 3语言编写UDTF代码，代码中需要包含如下信息：

- 导入模块：必选。

  至少要包含 `from odps.udf import annotate` 和 `from odps.udf import BaseUDTF` 。 `from odps.udf import annotate` 用于导入函数签名模块，MaxCompute才可以识别后续代码中定义的函数签名。 `from odps.udf import BaseUDTF` 为Python UDTF的基类，您需要通过此类在派生类中实现 `process` 或 `close` 等方法。

  当UDTF代码中需要引用文件资源或表资源时，需要包含 `from odps.distcache import get_cache_file` （文件资源）或 `from odps.distcache import get_cache_table` （表资源）。

- 函数签名：可选。

  格式为 `@annotate(<signature>)` ， `signature` 用于定义函数的输入参数和返回值的数据类型。如果不指定函数签名，在SQL中调用UDTF时，可以匹配任意类型的输入参数，但返回值类型无法推导，所有输出参数都将会是STRING类型。更多函数签名信息，请参见函数签名及数据类型。

- 自定义Python类（派生类）：必选。

  UDTF代码的组织单位，定义了实现业务需求的变量及方法。您还可以在代码中引用MaxCompute内置的第三方库或引用文件、表资源。更多信息，请参见第三方库或引用资源。

- 实现Python类的方法：必选。

  Python类实现包含如下4个方法，您可以根据实际需要进行选择。

方法定义	描述
`BaseUDTF.init()`	初始化方法。派生类如果需要实现此方法，必须在一开始调用基类的初始化方法 `super(BaseUDTF, self).init()` 。 `init` 方法在整个UDTF生命周期中只会被调用一次，即在处理第一条记录之前。当UDTF需要保存内部状态时，可以通过此方法初始化所有状态。
`BaseUDTF.process([args, ...])`	SQL中每一条记录都会对应调用一次 `process` ， `process` 的参数为SQL语句中指定的UDTF输入参数。
`BaseUDTF.forward([args, ...])`	UDTF的输出方法。此方法由用户代码调用。每调用一次 `forward` ，就会输出一条记录。 `forward` 的参数为SQL语句中指定的UDTF的输出参数。 如果Python代码中未指定函数签名，在调用 `forward` 方法时，必须将所有输出值转换为STRING类型。
`BaseUDTF.close()`	UDTF的结束方法。只会被调用一次，即在处理完最后一条记录之后被调用。

UDTF代码示例如下。

```
#导入函数签名模块及基类。
from odps.udf import annotate
from odps.udf import BaseUDTF
#函数签名。
@annotate('string -> string')
#自定义Python类。
class Explode(BaseUDTF):
#实现Python类的方法。
   def process(self, arg):
       props = arg.split(',')
       for p in props:
           self.forward(p)
```

> 说明 Python 2 UDTF与Python 3 UDTF区别在于底层Python语言版本不一致，请您根据对应版本语言支持的能力编写UDTF。

使用限制

Python 3与Python 2不兼容。在您使用Python 3之前，需要考虑兼容性问题，在一个SQL中不允许同时使用Python 3和Python 2。

Python 2 UDTF迁移

Python 2官方即将停止维护，建议您根据项目类型执行迁移操作：

- 全新项目：新MaxCompute项目，或第一次使用Python语言编写UDTF的MaxCompute项目。建议所有的Python UDTF都直接使用Python 3语言编写。
- 存量项目：创建了大量Python 2 UDTF的MaxCompute项目。请您谨慎开启Python 3。如果您计划逐步将所有Python 2 UDTF迁移为Python 3 UDTF，推荐方法如下：

- 新作业和新UDTF：使用Python 3语言编写，在Session级别开启Python 3。开启Python 3方法，请参见开启Python 3。
- Python 2 UDTF：改写Python 2 UDTF，使其可以同时兼容Python 2和Python 3。改写方法请参见将Python 2代码移植到Python 3。

> **说明** 如果您需要编写公共UDTF，并为多个MaxCompute项目授权UDTF的操作权限，建议UDTF同时兼容Python 2和Python 3。

开启Python 3

MaxCompute默认使用Python 2，如果您要使用Python 3，可以在Session级别设置如下属性开启Python 3，并与SQL语句一起提交执行。

```
set odps.sql.python.version=cp37;
```

第三方库

MaxCompute内置的Python 3运行环境中未安装第三方库Numpy。如果您需要使用Numpy的UDTF，请手动上传Numpy的WHEEL包。从PyPI或镜像下载Numpy包时，包的文件名为*numpy-<版本号>-cp37-cp37m-manylinux1_x86_64.whl*。上传包的操作请参见资源操作或Python UDF使用第三方包。

函数签名及数据类型

函数签名格式如下。

```
@annotate(<signature>)
```

`signature` 为函数签名字符串，用于标识输入参数和返回值的数据类型。执行UDTF时，UDTF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type_list'
```

其中：

- `type_list`：表示返回值的数据类型。UDTF可以返回多列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。
- `arg_type_list`：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

  `arg_type_list` 还支持星号（*）或为空（''）：
  - 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
  - 当 `arg_type_list` 为空（''）时，表示无输入参数。

> **说明** 在编写UDTF代码过程中，您可以根据MaxCompute项目的数据类型版本选取合适的数据类型，更多数据类型版本及各版本支持的数据类型信息，请参见数据类型版本说明。

合法的函数签名示例如下。

函数签名示例	说明
`@annotate('bigint,boolean->string,datetime')`	输入参数类型为BIGINT、BOOLEAN，返回值类型为STRING、DATETIME。
`@annotate('*->string, datetime')`	输入任意个参数，返回值类型为STRING、DATETIME。
`@annotate('->double, bigint, string')`	无输入参数，返回值类型为DOUBLE、BIGINT、STRING。
`@annotate("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")`	输入参数类型为ARRAY、STRUCT、MAP，返回值类型为MAP、STRUCT。

为确保编写Python UDTF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute SQL Type	Python 3 Type
BIGINT	INT
STRING	UNICODE
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	DATETIME.DATETIME
FLOAT	FLOAT

MaxCompute SQL Type	Python 3 Type
CHAR	UNICODE
VARCHAR	UNICODE
BINARY	BYTES
DATE	DATETIME.DATE
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT
STRUCT	COLLECTIONS.NAMEDTUPLE

引用资源

Python UDTF可以通过 `odps.distcache` 模块引用资源，支持引用文件资源和表资源。

- `odps.distcache.get_cache_file(resource_name)`：返回指定文件资源的内容。
  - `resource_name` 为STRING类型，对应当前MaxCompute项目中已存在的文件资源名。如果文件资源名非法或者没有相应的文件资源，会返回异常。

    > **说明** 使用UDTF访问资源，在创建UDTF时需要声明引用的资源，否则会报错。

  - 返回值为File-like对象。在使用完此对象后，您需要调用 `close` 方法释放打开的资源文件。
- `odps.distcache.get_cache_table(resource_name)`：返回指定表资源的内容。
  - `resource_name` 支持STRING类型，对应当前MaxCompute项目中已存在的表资源名。如果表资源名非法或者没有相应的表资源，会返回异常。
  - 返回值为GENERATOR类型，调用者以遍历方式获取表的内容，每次遍历可得到以数组形式存在的表中的一条记录。

引用文件资源和表资源的代码示例如下。

```
from odps.udf import annotate
from odps.udf import BaseUDTF
from odps.distcache import get_cache_file
from odps.distcache import get_cache_table
@annotate('string -> string, bigint')
class UDTFExample(BaseUDTF):
    """读取资源文件和资源表里的pageid、adid_list，生成dict
    """
    def __init__(self):
        import json
        cache_file = get_cache_file('test_json.txt')
        self.my_dict = json.load(cache_file)
        cache_file.close()
        records = list(get_cache_table('table_resource1'))
        for record in records:
            self.my_dict[record[0]] = record[1]
    """输入pageid，输出pageid以及它对应的所有adid
    """
    def process(self, pageid):
        for adid in self.my_dict[pageid]:
            self.forward(pageid, adid)
```

使用说明

按照开发流程，完成Python 3 UDTF开发后，您即可通过MaxCompute SQL调用Python 3 UDTF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;`。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute Studio完整开发及调用Python 3 UDTF的操作，请参见开发Python UDF。

3.10.3.5. Python UDTF使用示例

3.10.3.5.1. Python 2 UDTF读取MaxCompute资源示例

本文为您介绍基于MaxCompute客户端通过Python 2 UDTF读取MaxCompute资源的使用示例。

前提条件

已安装MaxCompute客户端。更多安装MaxCompute客户端操作，请参见安装并配置MaxCompute客户端。

UDTF代码示例

```
# -*- coding: utf-8 -*-
from odps.udf import annotate
from odps.udf import BaseUDTF
from odps.distcache import get_cache_file
from odps.distcache import get_cache_table
@annotate('string -> string, bigint')
class UDTFExample(BaseUDTF):
    """读取资源文件和资源表里的pageid、adid，生成dict
    """
    def __init__(self):
        import json
        cache_file = get_cache_file('test_json.txt')
        self.my_dict = json.load(cache_file)
        cache_file.close()
        records = list(get_cache_table('table_resource1'))
        for record in records:
            self.my_dict[record[0]] = [record[1]]
    """输入pageid，输出pageid以及它对应的所有adid
    """
    def process(self, pageid):
        for adid in self.my_dict[pageid]:
            self.forward(pageid, adid)
```

操作步骤

1. 将UDTF代码示例保存为py_udtf_example.py文件，放置于MaxCompute客户端的bin目录中。
2. 登录MaxCompute客户端创建资源表table_resource1和内部表tmp1（后续执行DML操作写入的目标表）并插入数据，准备资源文件test_json.txt并放置于MaxCompute客户端的bin目录中。

   更多登录MaxCompute客户端操作，请参见登录MaxCompute客户端。建表、插入数据命令及资源文件内容示例如下：

   - 创建资源表table_resource1，并插入数据。

     ```
     create table if not exists table_resource1 (pageid string, adid int);
     insert into table table_resource1 values("contact_page2",2),("contact_page3",5);
     ```

   - 创建内部表tmp1，并插入数据。

     ```
     create table if not exists tmp1 (pageid string);
     insert into table tmp1 values ("front_page"),("contact_page1"),("contact_page3");
     ```

   - 资源文件test_json.txt的内容如下。

     ```
     {"front_page":[1, 2, 3], "contact_page1":[3, 4, 5]}
     ```

3. 通过MaxCompute客户端，将py_udtf_example.py文件、test_json.txt和表table_resource1添加为MaxCompute的资源。

   更多添加资源信息，请参见添加资源。命令示例如下。

   ```
   add py py_udtf_example.py;
   add file test_json.txt;
   add table table_resource1 as table_resource1;
   ```

4. 在MaxCompute客户端上创建UDTF函数my_udtf。

   更多创建函数信息，请参见注册函数。命令示例如下。

   ```
   create function my_udtf as 'py_udtf_example.UDTFExample' using 'py_udtf_example.py, test_json.txt, table_resource1';
   ```

5. 在MaxCompute客户端上执行SQL命令调用新创建的UDTF。

   命令示例如下：

   - 示例1：单纯使用UDTF函数运行SQL。

     ```
     select my_udtf(pageid) as (pageid, adid) from tmp1;
     ```

     返回结果如下。

```
+------------+------------+
| pageid     | adid       |
+------------+------------+
| front_page | 1          |
| front_page | 2          |
| front_page | 3          |
| contact_page1 | 3          |
| contact_page1 | 4          |
| contact_page1 | 5          |
| contact_page3 | 5          |
+------------+------------+
```

- 示例2：对示例1中的命令改写，结合Lateral View运行SQL。

```
select pageid, adid from tmp1 lateral view my_udtf(pageid) adTable as udtf_pageid, adid;
```

返回结果如下。

```
+------------+------------+
| pageid     | adid       |
+------------+------------+
| front_page | 1          |
| front_page | 2          |
| front_page | 3          |
| contact_page1 | 3          |
| contact_page1 | 4          |
| contact_page1 | 5          |
| contact_page3 | 5          |
+------------+------------+
```

执行结果的两列分别为tmp1.pageid和UDTF输出的adid。

- 示例3：结合聚合函数和Lateral View运行SQL。

```
--聚合统计每个adid出现的次数。
select adid, count(1) as cnt
    from tmp1 lateral view my_udtf(pageid) adTable as udtf_pageid, adid
group by adid;
```

返回结果如下。

```
+------------+------------+
| adid       | cnt        |
+------------+------------+
| 1          | 1          |
| 2          | 1          |
| 3          | 2          |
| 4          | 1          |
| 5          | 2          |
+------------+------------+
```

3.10.3.5.2. Python 3 UDTF读取MaxCompute资源示例

本文为您介绍基于MaxCompute客户端通过Python 3 UDTF读取MaxCompute资源的使用示例。

前提条件

已安装MaxCompute客户端。更多安装MaxCompute客户端操作，请参见安装并配置MaxCompute客户端。

UDTF代码示例

```
from odps.udf import annotate
from odps.udf import BaseUDTF
from odps.distcache import get_cache_file
from odps.distcache import get_cache_table
@annotate('string -> string, bigint')
class UDTFExample(BaseUDTF):
    """读取资源文件和资源表里的pageid、adid_list，生成dict
    """
    def __init__(self):
        import json
        cache_file = get_cache_file('test_json.txt')
        self.my_dict = json.load(cache_file)
        cache_file.close()
        records = list(get_cache_table('table_resource1'))
        for record in records:
            self.my_dict[record[0]] = record[1]
    """输入pageid，输出pageid以及它对应的所有adid
    """
    def process(self, pageid):
        for adid in self.my_dict[pageid]:
            self.forward(pageid, adid)
```

操作步骤

1. 将UDTF代码示例保存为py_udtf_example.py文件，放置于MaxCompute客户端的bin目录中。
2. 登录MaxCompute客户端创建资源表table_resource1和内部表tmp1（后续执行DML操作写入的目标表）并插入数据，准备资源文件test_json.txt并放置于MaxCompute客户端的bin目录中。

   更多登录MaxCompute客户端操作，请参见登录MaxCompute客户端。建表、插入数据命令及资源文件内容示例如下：

   - 创建资源表table_resource1，并插入数据。

     ```
     create table if not exists table_resource1 (pageid string, adid_list array<int>);
     insert into table table_resource1 values("contact_page2",array(2,3,4)),("contact_page3",array(5,6,7));
     ```

     > **说明** 由于table_resource1中adid_list字段数据类型为ARRAY，读取表资源时需要在Session级别执行 `set odps.sql.python.version=cp37;` 命令开启Python 3来支持读取ARRAY类型数据。

   - 创建内部表tmp1，并插入数据。

     ```
     create table if not exists tmp1 (pageid string);
     insert into table tmp1 values ("front_page"),("contact_page1"),("contact_page3");
     ```

   - 资源文件test_json.txt的内容如下。

     ```
     {"front_page":[1, 2, 3], "contact_page1":[3, 4, 5]}
     ```

3. 通过MaxCompute客户端，将py_udtf_example.py文件、test_json.txt和表table_resource1添加为MaxCompute的资源。

   更多添加资源信息，请参见添加资源。命令示例如下。

   ```
   add py py_udtf_example.py;
   add file test_json.txt;
   add table table_resource1 as table_resource1;
   ```

4. 在MaxCompute客户端上创建UDTF函数my_udtf。

   更多创建函数信息，请参见注册函数。命令示例如下。

   ```
   create function my_udtf as 'py_udtf_example.UDTFExample' using 'py_udtf_example.py, test_json.txt, table_resource1';
   ```

5. 在MaxCompute客户端上执行SQL命令调用新创建的UDTF。

   命令示例如下：

   - 示例1：单纯使用UDTF函数运行SQL。

     ```
     select my_udtf(pageid) as (pageid, adid) from tmp1;
     ```

     返回结果如下。

```
+------------+------------+
| pageid     | adid       |
+------------+------------+
| front_page | 1          |
| front_page | 2          |
| front_page | 3          |
| contact_page1 | 3          |
| contact_page1 | 4          |
| contact_page1 | 5          |
| contact_page3 | 5          |
| contact_page3 | 6          |
| contact_page3 | 7          |
+------------+------------+
```

- 示例2：对示例1中的命令改写，结合Lateral View运行SQL。

```
select pageid, adid from tmp1 lateral view my_udtf(pageid) adTable as udtf_pageid, adid;
```

返回结果如下。

```
+--------+------------+
| pageid | adid       |
+--------+------------+
| front_page | 1          |
| front_page | 2          |
| front_page | 3          |
| contact_page1 | 3          |
| contact_page1 | 4          |
| contact_page1 | 5          |
| contact_page3 | 5          |
| contact_page3 | 6          |
| contact_page3 | 7          |
+--------+------------+
```

- 示例3：结合聚合函数和Lateral View运行SQL。

```
select adid, count(1) as cnt
    from tmp1 lateral view my_udtf(pageid) adTable as udtf_pageid, adid
group by adid;
```

返回结果如下。

```
+------------+------------+
| adid       | cnt        |
+------------+------------+
| 1          | 1          |
| 2          | 1          |
| 3          | 2          |
| 4          | 1          |
| 5          | 2          |
| 6          | 1          |
| 7          | 1          |
+------------+------------+
```

3.10.4. UDAF

3.10.4.1. UDAF概述

MaxCompute支持您通过Java、Python语言编写代码创建UDAF，扩展MaxCompute的函数能力，满足个性化业务需求。本文为您介绍UDAF类型、使用限制、注意事项、开发流程及使用说明等信息。

背景信息

UDAF为用户自定义聚合函数，适用于多进一出业务场景。即其输入与输出是多对一的关系，将多条输入记录聚合成一个输出值。MaxCompute支持您通过如下两种代码语言编写UDAF。

UDAF类型	说明
Java UDAF	使用Java语言编写UDAF逻辑实现代码。更多信息，请参见Java UDAF。
Python UDAF	使用Python语言编写UDAF逻辑实现代码，支持Python 2和Python 3： • Python 2 UDAF：Python语言版本为2.7。更多信息，请参见Python 2 UDAF。 • Python 3 UDAF：Python语言版本为CPython-3.7.3。更多信息，请参见Python 3 UDAF。

使用限制

不支持通过自定义函数访问外网。如果您需要通过自定义函数访问外网，请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

注意事项

使用自定义函数时，您需要注意：

- 在性能上，自定义函数的性能低于内建函数，建议您优先使用内建函数实现相同逻辑的业务需求。
- 在SQL语句中使用自定义函数时，如果计算的数据量过大并且存在倾斜，会导致作业占用的内存超出默认分配的内存。此时，您可以在Session级别设置 `set odps.sql.udf.joiner.jvm.memory=xxxx;` 属性来解决此问题。更多自定义函数常见问题，请参见MaxCompute UDF常见问题。
- 当自定义函数的名称与内建函数的名称相同时，自定义函数会覆盖同名的内建函数。例如，MaxCompute中存在一个名称为CONCAT的自定义函数，则系统默认会调用自定义的CONCAT，不会调用内建函数CONCAT。如果您希望调用内建函数，需要在内建函数前增加 `::` 符号，例如 `select :: concat('ab', 'c');` 。

开发流程

UDAF的开发流程如下。

- 当您使用Java代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档

①	可选	当您使用Maven编写代码时，需要先在Pom文件中添加代码相关SDK依赖，确保后续编写的代码可编译成功。例如开发自定义函数需要添加的SDK依赖为：
		`<dependency>`
		`<groupId>com.aliyun.odps</groupId>`
		`<artifactId>odps-sdk-udf</artifactId>`
		`<version>0.29.10-public</version>`
		`</dependency>`
		您可以从Maven库中搜索 `odps-sdk-udf` ，获取SDK依赖版本信息。
			IntelliJ IDEA（Maven）	无

②	必选	根据业务需求，编写自定义函数代码。	IntelliJ IDEA（Maven）、MaxCompute Studio	开发Java UDF
③	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		
④	必选	调试自定义函数代码，确保本地运行成功后打包为JAR包。		
⑤	必选	将JAR包作为资源上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	○ MaxCompute客户端 ■ 添加资源 ■ 注册函数 ○ MaxCompute Studio 打包、上传及注册 ○ DataWorks ■ 创建JAR资源 ■ 注册MaxCompute函数
⑥	必选	基于上传的JAR包资源创建自定义函数。		
⑦	可选	在查询数据代码中调用自定义函数。		无

- 当您使用Python代码编写MaxCompute UDF时，开发流程如下。



序号	可选/必选步骤	功能介绍	工具平台	相关文档
①	必选	根据业务需求，编写自定义函数代码。	MaxCompute Studio	开发Python UDF
②	必选	通过本地运行或单元测试方式调试自定义函数，查看运行结果是否符合预期。		

序号	可选/必选步骤	功能介绍	工具平台	相关文档
③	必选	将Python文件或依赖的资源（例如文件资源、表资源、第三方包）上传至MaxCompute项目。	使用客户端（odpscmd）连接、MaxCompute Studio、DataWorks	◦ MaxCompute客户端 ▪ 添加资源 ▪ 注册函数 ◦ MaxCompute Studio 上传及注册 ◦ DataWorks 创建Python资源并注册函数
④	必选	基于上传的Python文件或依赖的资源创建自定义函数。		
⑤	可选	在查询数据代码中调用自定义函数。		无

使用说明

自定义函数的使用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;`。更多跨项目分享信息，请参见基于Package跨项目访问资源。

3.10.4.2. Java UDAF

本文为您介绍如何通过Java语言编写UDAF。

UDAF代码结构

您可以通过IntelliJ IDEA（Maven）或MaxCompute Studio工具使用Java语言编写UDAF代码，代码中需要包含如下信息：

- Java包（Package）：可选。

  您可以将定义的Java类打包，为后续查找和使用类提供方便。

- 继承UDAF类：必选。

  必需携带的UDAF类为`import com.aliyun.odps.udf.Aggregator`和`com.aliyun.odps.udf.annotation.Resolve`（对应`@Resolve`注解）。`com.aliyun.odps.udf.UDFException`（可选，对应实现Java类初始化和结束的方法）。当您需要使用其他UDAF类或者需要用到复杂数据类型时，请根据MaxCompute SDK添加需要的类。

- `@Resolve`注解：必选。

  格式为`@Resolve(<signature>)`。`signature`为函数签名，用于定义函数的输入参数和返回值的数据类型。UDAF无法通过反射分析获取函数签名，只能通过`@Resolve`注解方式获取函数签名，例如`@Resolve("smallint->varchar(10)")`。更多`@Resolve`注解信息，请参见@Resolve注解。

- 自定义Java类：必选。

  UDAF代码的组织单位，定义了实现业务需求的变量及方法。

- 实现Java类的方法：必选。

  实现Java类需要继承`com.aliyun.odps.udf.Aggregator`类并实现如下方法。

```
import com.aliyun.odps.udf.ContextFunction;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
public abstract class Aggregator implements ContextFunction {
    //初始化方法。
    @Override
    public void setup(ExecutionContext ctx) throws UDFException {
    }
    //结束方法。
    @Override
    public void close() throws UDFException {
    }
    //创建聚合Buffer。
    abstract public Writable newBuffer();
    //iterate方法。
    //buffer为聚合buffer，是指一个阶段性的汇总数据，即在不同的Map任务中，group by后得出的数据（可理解为一个集合），每行执行一次。
    //Writable[]表示一行数据，在代码中指代传入的列。例如writable[0]表示第一列，writable[1]表示第二列。
    //args为SQL中调用UDAF时指定的参数，不能为NULL，但是args里面的元素可以为NULL，代表对应的输入数据是NULL。
    abstract public void iterate(Writable buffer, Writable[] args) throws UDFException;
    //terminate方法。
    abstract public Writable terminate(Writable buffer) throws UDFException;
    //merge方法。
    abstract public void merge(Writable buffer, Writable partial) throws UDFException;
}
```

其中：`iterate`、`merge`和`terminate`是最重要的三个方法，UDAF的主要逻辑依赖于这三个方法的实现。此外，还需要您实现自定义的Writable buffer。

Writable buffer将内存中的对象转换成字节序列（或其他数据传输协议）以便于储存到磁盘（持久化）和网络传输。因为MaxCompute使用分布式计算的方式来处理聚合函数，因此需要知道如何序列化和反序列化数据，以便于数据在不同的设备之间进行传输。

编写Java UDAF时可以使用Java Type或Java Writable Type，MaxCompute项目支持处理的数据类型与Java数据类型的详细映射关系，请参见数据类型。

UDAF代码示例如下。

```
//将定义的Java类组织在org.alidata.odps.udaf.examples包中。
package org.alidata.odps.udaf.examples;
//继承UDAF类。
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
//自定义Java类。
//@Resolve注解。
@Resolve("double->double")
public class AggrAvg extends Aggregator {
//实现Java类的方法。
  private static class AvgBuffer implements Writable {
    private double sum = 0;
    private long count = 0;
    @Override
    public void write(DataOutput out) throws IOException {
      out.writeDouble(sum);
      out.writeLong(count);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
      sum = in.readDouble();
      count = in.readLong();
    }
  }
  private DoubleWritable ret = new DoubleWritable();
  @Override
  public Writable newBuffer() {
    return new AvgBuffer();
  }
  @Override
  public void iterate(Writable buffer, Writable[] args) throws UDFException {
    DoubleWritable arg = (DoubleWritable) args[0];
    AvgBuffer buf = (AvgBuffer) buffer;
    if (arg != null) {
      buf.count += 1;
      buf.sum += arg.get();
    }
  }
  @Override
  public Writable terminate(Writable buffer) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    if (buf.count == 0) {
      ret.set(0);
    } else {
      ret.set(buf.sum / buf.count);
    }
    return ret;
  }
  @Override
  public void merge(Writable buffer, Writable partial) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    AvgBuffer p = (AvgBuffer) partial;
    buf.sum += p.sum;
    buf.count += p.count;
  }
}
```

使用限制

不支持通过自定义函数访问外网。如果您需要通过自定义函数访问外网，请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

注意事项

在编写Java UDAF时，您需要注意：

- 不同UDAF JAR包中不建议存在类名相同但实现逻辑不一样的类。例如UDAF1、UDAF2分别对应资源JAR包udaf1.jar、udaf2.jar，两个JAR包里都包含名称为 `com.aliyun.UserFunction.class` 的类但实现逻辑不一样，当同一条SQL语句中同时调用UDAF1和UDAF2时，MaxCompute会随机加载其中

一个类，此时会导致UDAF执行结果不符合预期甚至编译失败。

- Java UDAF中输入或返回值的数据类型是对象，数据类型首字母必须大写，例如String。
- SQL中的NULL值通过Java中的NULL表示。Java Primitive Type无法表示SQL中的NULL值，不允许使用。

@Resolve注解

`@Resolve` 注解格式如下。

```
@Resolve(<signature>)
```

`signature` 为字符串，用于标识输入参数和返回值的数据类型。执行UDAF时，UDAF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type'
```

其中：

- `arg_type_list` ：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

  `arg_type_list` 还支持星号（*）或为空（''）：

  - 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
  - 当 `arg_type_list` 为空（''）时，表示无输入参数。
- `type` ：表示返回值的数据类型。UDAF只返回一列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

> **说明** 在编写UDAF代码过程中，您可以根据MaxCompute项目的数据类型版本选取合适的数据类型，更多数据类型版本及各版本支持的数据类型信息，请参见数据类型版本说明。

合法 `@Resolve` 注解示例如下。

@Resolve注解示例	说明
`@Resolve('bigint,double->string')`	输入参数类型为BIGINT、DOUBLE，返回值类型为STRING。
`@Resolve('*->string')`	输入任意个参数，返回值类型为STRING。
`@Resolve('->double')`	无输入参数，返回值类型为DOUBLE。
`@Resolve('array<bigint>->struct<x:string, y:int>')`	输入参数类型为ARRAY<BIGINT>，返回值类型为STRUCT<x:STRING, y:INT>。

数据类型

在MaxCompute中不同数据类型版本支持的数据类型不同。从MaxCompute 2.0版本开始，扩展了更多的新数据类型，同时还支持ARRAY、MAP、STRUCT等复杂类型。更多MaxCompute数据类型版本信息，请参见数据类型版本说明。

为确保编写Java UDAF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute Type	Java Type	Java Writable Type
TINYINT	java.lang.Byte	ByteWritable
SMALLINT	java.lang.Short	ShortWritable
INT	java.lang.Integer	IntWritable
BIGINT	java.lang.Long	LongWritable
FLOAT	java.lang.Float	FloatWritable
DOUBLE	java.lang.Double	DoubleWritable
DECIMAL	java.math.BigDecimal	BigDecimalWritable
BOOLEAN	java.lang.Boolean	BooleanWritable
STRING	java.lang.String	Text
VARCHAR	com.aliyun.odps.data.Varchar	VarcharWritable
BINARY	com.aliyun.odps.data.Binary	BytesWritable

MaxCompute Type	Java Type	Java Writable Type
DATETIME	java.util.Date	DatetimeWritable
TIMESTAMP	java.sql.Timestamp	TimestampWritable
INTERVAL_YEAR_MONTH	不涉及	IntervalYearMonthWritable
INTERVAL_DAY_TIME	不涉及	IntervalDayTimeWritable
ARRAY	java.util.List	不涉及
MAP	java.util.Map	不涉及
STRUCT	com.aliyun.odps.data.Struct	不涉及

说明 当MaxCompute项目采用MaxCompute 2.0数据类型版本时，UDAF的输入或返回值才可以使用Java Writable Type。

使用说明

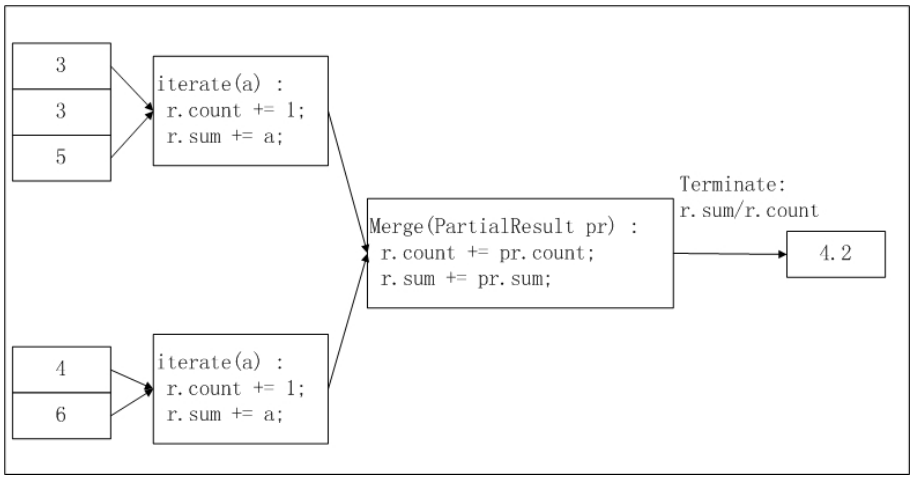
按照开发流程，完成Java UDAF开发后，您即可通过MaxCompute SQL调用Java UDAF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;`。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute Studio完整开发及调用Java UDAF的操作，请参见使用示例。

使用示例

以通过MaxCompute Studio开发计算平均值的UDAF函数 `AggrAvg` 为例，实现逻辑如下。



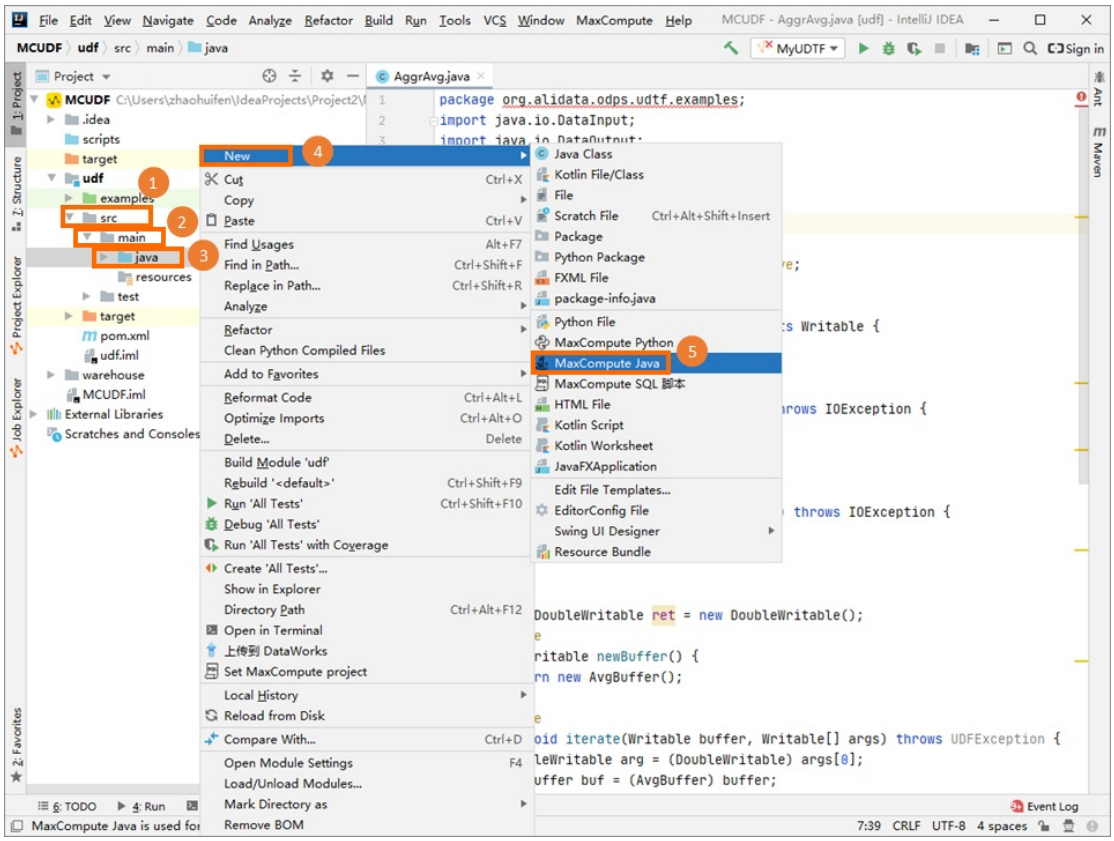
1. 输入数据分片：MaxCompute会按照MapReduce处理流程对输入数据按照一定的大小进行分片，每片的大小适合一个Worker在适当的时间内完成。

   分片大小需要您通过 `odps.stage.mapper.split.size` 参数进行配置。分片逻辑请参见MapReduce流程说明。
2. 计算平均值第一阶段：每个Worker统计分片内数据的个数及汇总值。您可以将每个分片内的数据个数及汇总值视为一个中间结果。
3. 计算平均值第二阶段：汇总第一阶段中每个分片内的信息。
4. 最终输出：`r.sum/r.count` 即是所有输入数据的平均值。

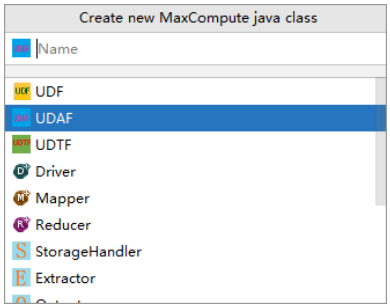
开发并调用Java UDAF的操作步骤如下：

1. 在IntelliJ IDEA上完成如下准备工作：
   i. 安装MaxCompute Studio
   ii. 创建MaxCompute项目连接
   iii. 创建MaxCompute Java Module
2. 编写UDAF代码。

i. 在**Project**区域，右键单击Module的源码目录（即**src > main > java**），选择**new > MaxCompute Java**。

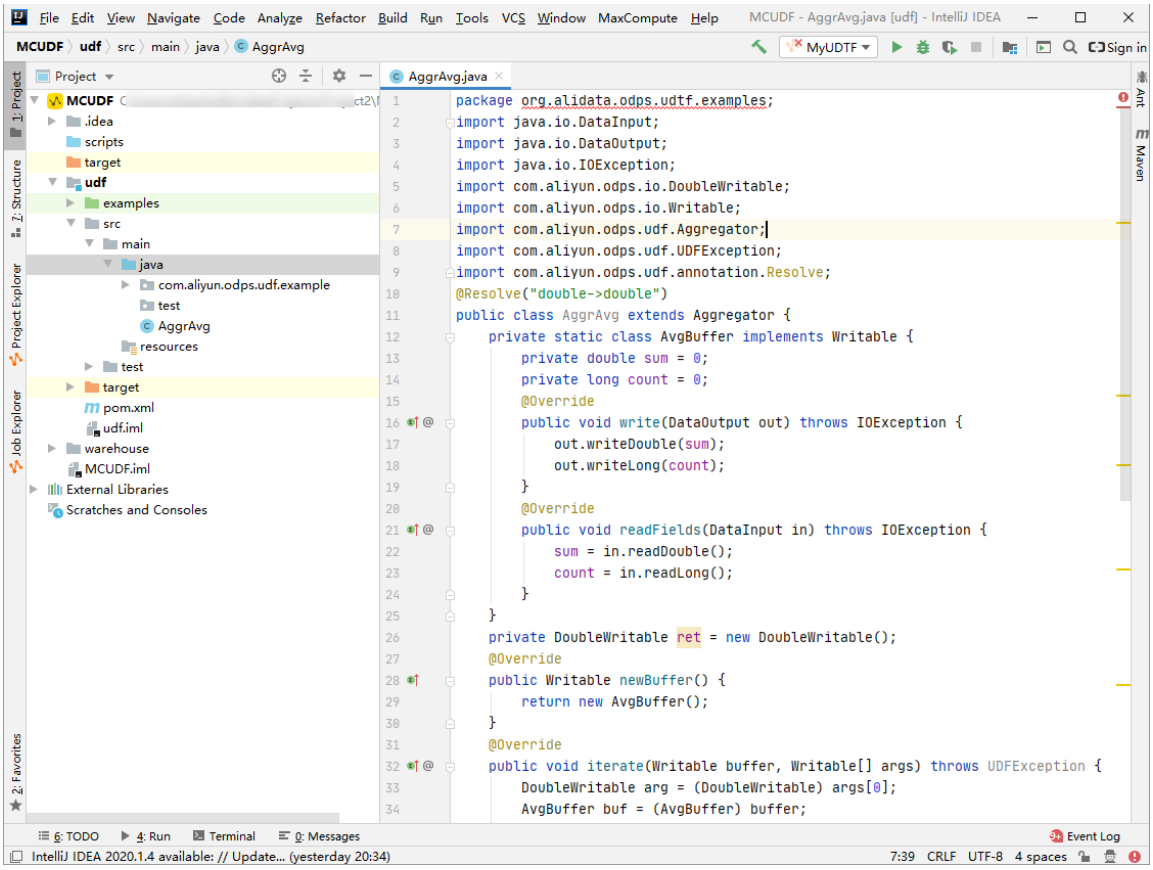


ii. 在**Create new MaxCompute java class**对话框，单击**UDAF**并填写**Name**后，按Enter键。例如Java Class名称为AggrAvg。



Name为创建的MaxCompute Java Class名称。如果还没有创建Package，在此处填写packagename.classname，会自动生成Package。

iii. 在代码编写区域写入如下代码。

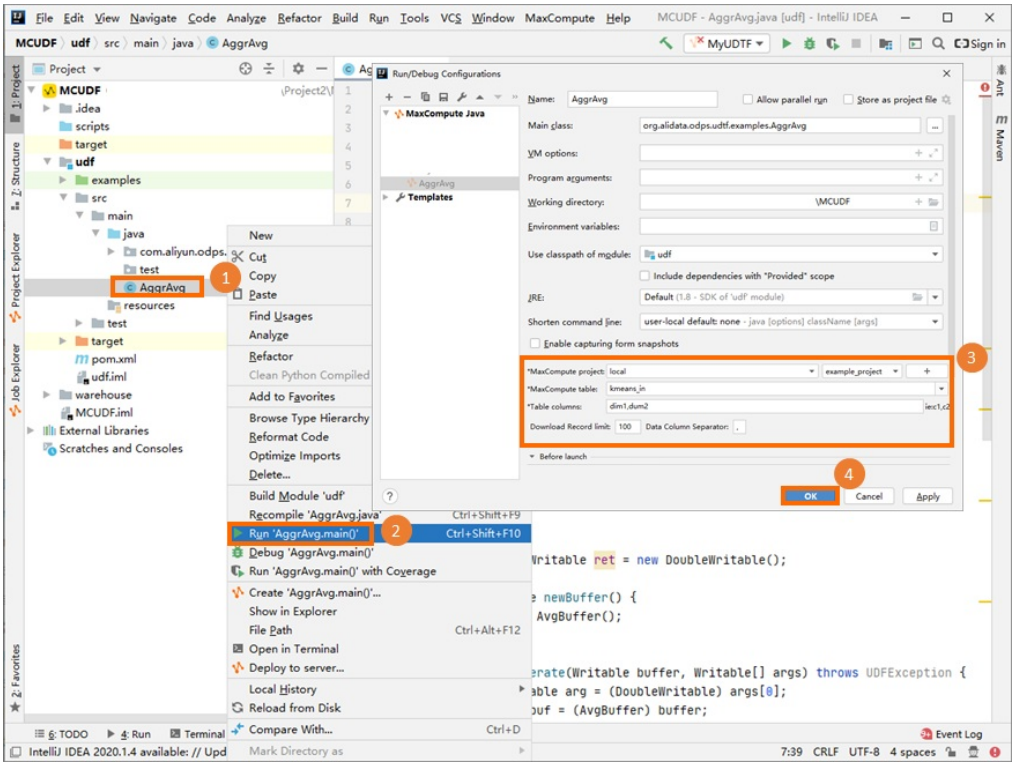

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window MaxCompute Help
MCUDF - AggrAvg.java [udf] - IntelliJ IDEA
MCUDF udf src main java AggrAvg
MyUDTF
Sign in
Project
AggrAvg.java
MCUDF
.idea
scripts
target
udf
examples
src
main
java
com.aliyun.odps.udf.example
test
AggrAvg
resources
test
target
pom.xml
udf.iml
warehouse
MCUDF.iml
External Libraries
Scratches and Consoles
package org.alidata.odps.udtf.examples;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("double->double")
public class AggrAvg extends Aggregator {
private static class AvgBuffer implements Writable {
private double sum = 0;
private long count = 0;
@Override
public void write(DataOutput out) throws IOException {
out.writeDouble(sum);
out.writeLong(count);
}
@Override
public void readFields(DataInput in) throws IOException {
sum = in.readDouble();
count = in.readLong();
}
}
private DoubleWritable ret = new DoubleWritable();
@Override
public Writable newBuffer() {
return new AvgBuffer();
}
@Override
public void iterate(Writable buffer, Writable[] args) throws UDFException {
DoubleWritable arg = (DoubleWritable) args[0];
AvgBuffer buf = (AvgBuffer) buffer;
6: TODO
4: Run
Terminal
0: Messages
Event Log
IntelliJ IDEA 2020.1.4 available: // Update... (yesterday 20:34)
7:39 CRLF UTF-8 4 spaces

UDAF代码示例如下。

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("double->double")
public class AggrAvg extends Aggregator {
  private static class AvgBuffer implements Writable {
    private double sum = 0;
    private long count = 0;
    @Override
    public void write(DataOutput out) throws IOException {
      out.writeDouble(sum);
      out.writeLong(count);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
      sum = in.readDouble();
      count = in.readLong();
    }
  }
  private DoubleWritable ret = new DoubleWritable();
  @Override
  public Writable newBuffer() {
    return new AvgBuffer();
  }
  @Override
  public void iterate(Writable buffer, Writable[] args) throws UDFException {
    DoubleWritable arg = (DoubleWritable) args[0];
    AvgBuffer buf = (AvgBuffer) buffer;
    if (arg != null) {
      buf.count += 1;
      buf.sum += arg.get();
    }
  }
  @Override
  public Writable terminate(Writable buffer) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    if (buf.count == 0) {
      ret.set(0);
    } else {
      ret.set(buf.sum / buf.count);
    }
    return ret;
  }
  @Override
  public void merge(Writable buffer, Writable partial) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    AvgBuffer p = (AvgBuffer) partial;
    buf.sum += p.sum;
    buf.count += p.count;
  }
}
```

3. 在本地运行调试UDAF，确保代码可以运行成功。

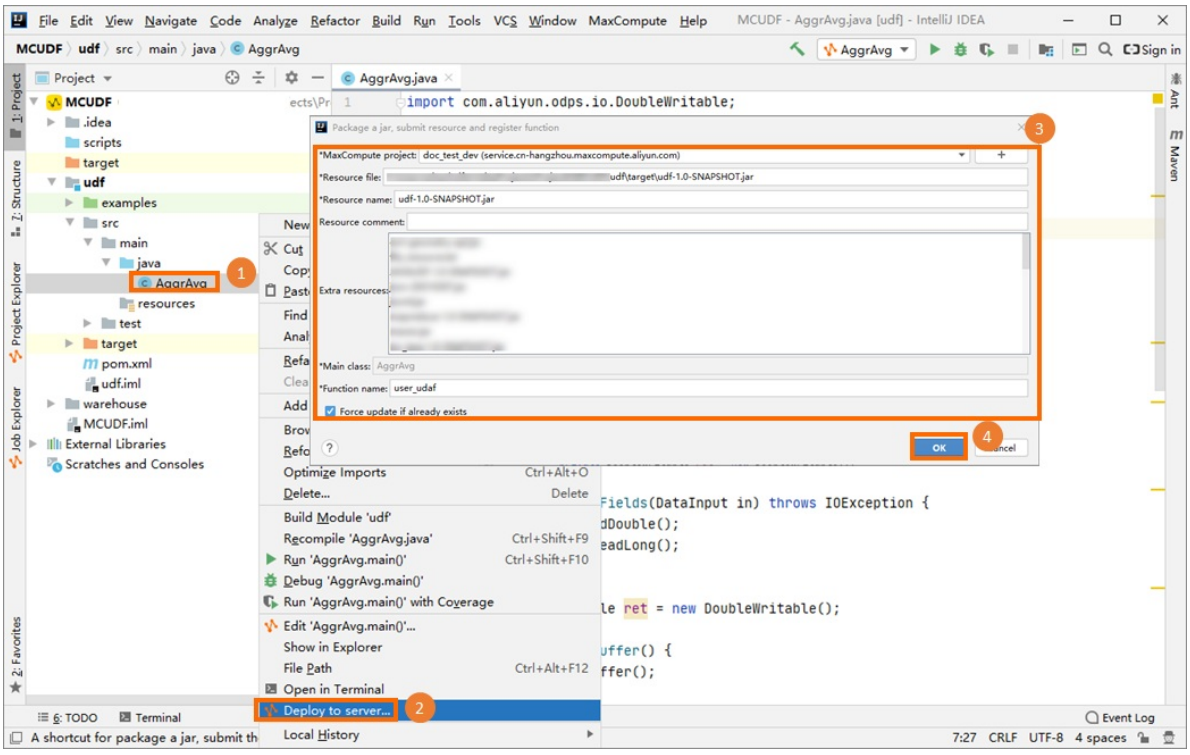
   更多调试操作，请参见通过本地运行调试UDF。



> **说明** 运行参数可参照图示数据填写。

4. 将创建的UDAF打包为JAR包，上传至MaxCompute项目并注册函数。例如函数名称为 `user_udaf` 。

   更多打包操作，请参见操作步骤。



5. 在MaxCompute Studio的左侧导航栏，单击**Project Explorer**，在目标MaxCompute项目上单击右键，启动MaxCompute客户端，并执行SQL命令调用新创建的UDAF。

   假设待查询目标表my_table的数据结构如下。

```
+------------+------------+
| col0       | col1       |
+------------+------------+
| 1.2        | 2.0        |
| 1.6        | 2.1        |
+------------+------------+
```

执行如下SQL命令调用UDAF。

```
select user_udaf(col0) as c0 from my_table;
```

返回结果如下。

```
+----+
| c0 |
+----+
| 1.4|
+----+
```

3.10.4.3. Python UDAF

3.10.4.3.1. Python 2 UDAF

MaxCompute使用的Python 2版本为2.7。本文为您介绍如何通过Python 2语言编写UDAF。

UDAF代码结构

您可以通过MaxCompute Studio工具使用Python 2语言编写UDAF代码，代码中需要包含如下信息：

- 编码声明：可选。

  固定声明格式为 `#coding:utf-8` 或 `# -*- coding: utf-8 -*-` ，二者等效。当Python 2代码中出现中文字符时，运行程序会报错，您需要在代码头部增加编码声明。

- 导入模块：必选。

  至少要包含 `from odps.udf import annotate` 和 `from odps.udf import BaseUDAF` 。 `from odps.udf import annotate` 用于导入函数签名模块，MaxCompute才可以识别后续代码中定义的函数签名。 `from odps.udf import BaseUDAF` 为Python UDAF的基类，您需要通过此类在派生类中实现 `iterate` 、 `merge` 、 `terminate` 等方法。

  当UDAF代码中需要引用文件资源或表资源时，需要包含 `from odps.distcache import get_cache_file` （文件资源）或 `from odps.distcache import get_cache_table` （表资源）。

- 函数签名：必选。

  格式为 `@annotate(<signature>)` ， `signature` 用于定义函数的输入参数和返回值的数据类型。更多函数签名信息，请参见函数签名及数据类型。

- 自定义Python类（派生类）：必选。

  UDAF代码的组织单位，定义了实现业务需求的变量及方法。您还可以在代码中引用MaxCompute内置的第三方库或引用文件、表资源。更多信息，请参见第三方库或引用资源。

- 实现Python类的方法：必选。

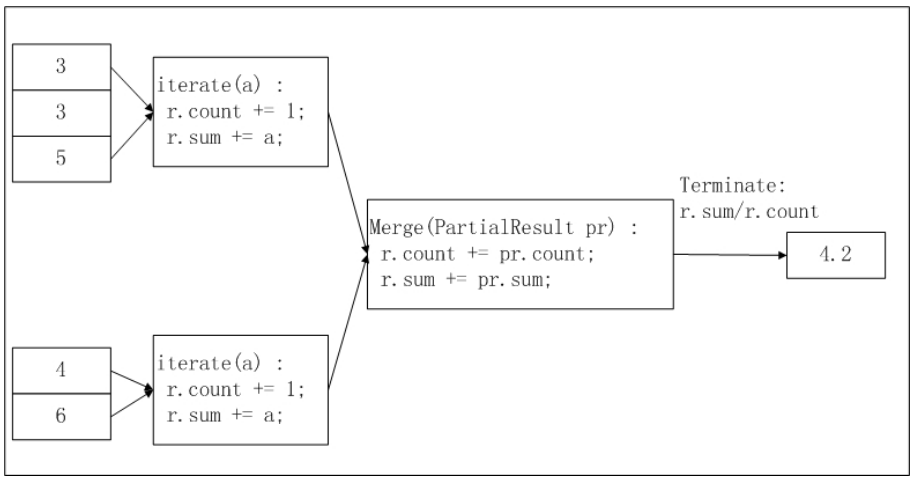
  Python类实现包含如下4个方法，您可以根据实际需要进行选择。

方法定义	描述
`BaseUDAF.new_buffer()`	返回聚合函数的中间值的buffer。 `buffer` 必须是Marshal对象（例如LIST、DICT），并且 `buffer` 的大小不应该随数据量递增。在极限情况下， `buffer` 在执行对象序列化后的大小不应该超过2 MB。
`BaseUDAF.iterate(buffer[, args, ...])`	将 `args` 聚合到中间值 `buffer` 中。
`BaseUDAF.merge(buffer, pbuffer)`	将中间值 `buffer` 和 `pbuffer` 合并的结果存放在 `buffer` 中。
`BaseUDAF.terminate(buffer)`	将 `buffer` 转换为MaxCompute SQL的基本类型。

UDAF代码示例如下。

```
#coding:utf-8
#导入函数签名模块及基类。
from odps.udf import annotate
from odps.udf import BaseUDAF
#函数签名。
@annotate('double->double')
#自定义Python类。
class Average(BaseUDAF):
#实现Python类的方法。
    def new_buffer(self):
        return [0, 0]
    def iterate(self, buffer, number):
        if number is not None:
            buffer[0] += number
            buffer[1] += 1
    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]
    def terminate(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

求平均值`avg`的MaxCompute UDAF的实现逻辑及计算流程如下。



`pbuffer`相当于上图中的`pr`，`buffer`相当于上图中的`r`。

使用限制

MaxCompute Python 2 UDAF使用的Python版本为2.7，并以沙箱模式执行用户代码，即代码是在一个受限的运行环境中执行的。在该环境中，以下行为会被禁止：

- 读写本地文件。
- 启动子进程。
- 启动线程。
- 使用Socket通信。
- 其他系统调用。

基于上述原因，您上传的代码都必须通过标准Python实现。Python标准库中涉及到上述功能的模块或C扩展模块都会被禁止使用。具体标准库的可用模块说明如下：

- 所有基于标准Python实现（不依赖扩展模块）的模块都可用。
- C扩展模块中下列模块可用：
  - array、audioop
  - binascii、bisect
  - cmath、_codecs_cn、_codecs_hk、_codecs_iso2022、_codecs_jp、_codecs_kr、_codecs_tw、_collections、cStringIO
  - datetime
  - _functools、future_builtins、
  - _heapq、_hashlib
  - itertools
  - _json

- _locale、_lsprof
- math、_md5、_multibytecodec
- operator
- _random
- _sha256、_sha512、_sha、_struct、strop
- time
- unicodedata
- _weakref
- cPickle

- 沙箱限制了您的代码最多可向标准输出和标准错误输出写入数据的大小为20 KB，即 `sys.stdout/sys.stderr` 最多能写入20 KB数据，多余的字符会被忽略。

第三方库

MaxCompute的Python 2运行环境中安装了除Python标准库外比较常用的第三方库，作为标准库的补充，例如Numpy。

> **说明** 使用第三方库存在限制，例如禁止本地访问、网络I/O受限，因此第三方库中涉及到相关功能的API也被禁止使用。

函数签名及数据类型

函数签名格式如下。

```
@annotate(<signature>)
```

`signature` 为字符串，用于标识输入参数和返回值的数据类型。执行UDAF时，UDAF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type'
```

其中：

- `arg_type_list` ：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

  `arg_type_list` 还支持星号（*）或为空（''）：
  - 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
  - 当 `arg_type_list` 为空（''）时，表示无输入参数。
- `type` ：表示返回值的数据类型。UDAF只返回一列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

> **说明** 在编写UDAF代码过程中，您可以根据MaxCompute项目的数据类型版本选取合适的数据类型，更多数据类型版本及各版本支持的数据类型信息，请参见数据类型版本说明。

合法函数签名示例如下。

函数签名示例	说明
`@annotate('bigint,double->string')`	输入参数类型为BIGINT、DOUBLE，返回值类型为STRING。
`@annotate('*->string')`	输入任意个参数，返回值类型为STRING。
`@annotate('->double')`	无输入参数，返回值类型为DOUBLE。
`@annotate('array<bigint>->struct<x:string, y:int>')`	输入参数类型为ARRAY<BIGINT>，返回值类型为STRUCT<x:STRING, y:INT>。

为确保编写Python UDAF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute SQL Type	Python 2 Type
BIGINT	INT
STRING	STR
DOUBLE	FLOAT

MaxCompute SQL Type	Python 2 Type
BOOLEAN	BOOL
DATETIME	INT
FLOAT	FLOAT
CHAR	STR
VARCHAR	STR
BINARY	BYTEARRAY
DATE	INT
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT
STRUCT	COLLECTIONS.NAMEDTUPLE

> 说明
> - DATETIME类型对应的Python类型是INT，值为Epoch UTC Time起至今的毫秒数。您可以通过Python标准库中的DATETIME模块处理日期时间类型。
> - `odps.udf.int(value,[silent=True])` 增加了参数 `silent` 。当 `silent` 为True时，如果 `value` 无法转为INT，则会返回None（不会返回异常）。
> - NULL值对应Python的None。

引用资源

Python UDAF可以通过 `odps.distcache` 模块引用资源，支持引用文件资源和表资源。

- `odps.distcache.get_cache_file(resource_name)` ：返回指定文件资源的内容。
  - `resource_name` 为STRING类型，对应当前MaxCompute项目中已存在的文件资源名。如果文件资源名非法或者没有相应的文件资源，会返回异常。

    > 说明 使用UDAF访问资源，在创建UDAF时需要声明引用的资源，否则会报错。

  - 返回值为File-like对象。在使用完此对象后，您需要调用 `close` 方法释放打开的资源文件。
- `odps.distcache.get_cache_table(resource_name)` ：返回指定表资源的内容。
  - `resource_name` 支持STRING类型，对应当前MaxCompute项目中已存在的表资源名。如果表资源名非法或者没有相应的表资源，会返回异常。
  - 返回值为GENERATOR类型，调用者以遍历方式获取表的内容，每次遍历可得到以数组形式存在的表中的一条记录。

具体使用方法请参见引用资源（Python UDF 2）和引用资源（Python UDTF 2）。

使用说明

按照开发流程，完成Python 2 UDAF开发后，您即可通过MaxCompute SQL调用Python 2 UDAF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例： `select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute Studio完整开发及调用Python 2 UDAF的操作，请参见开发Python UDF。

3.10.4.3.2. Python 3 UDAF

Python官方即将停止维护Python 2，MaxCompute已支持Python 3，对应版本为CPython-3.7.3。本文为您介绍如何通过Python 3语言编写UDAF。

UDAF代码结构

您可以通过MaxCompute Studio工具使用Python 3语言编写UDAF代码，代码中需要包含如下信息：

- 导入模块：必选。

  至少要包含 `from odps.udf import annotate` 和 `from odps.udf import BaseUDAF` 。 `from odps.udf import annotate` 用于导入函数签名模块，MaxCompute才可以识别后续代码中定义的函数签名。 `from odps.udf import BaseUDAF` 为Python UDAF的基类，您需要通过此类在派生类中实现 `iterate` 、 `merge` 、 `terminate` 等方法。

当UDAF代码中需要引用文件资源或表资源时，需要包含 `from odps.distcache import get_cache_file` （文件资源）或 `from odps.distcache import get_cache_table` （表资源）。

- 函数签名：必选。

  格式为 `@annotate(<signature>)` ， `signature` 用于定义函数的输入参数和返回值的数据类型。更多函数签名信息，请参见函数签名及数据类型。

- 自定义Python类（派生类）：必选。

  UDAF代码的组织单位，定义了实现业务需求的变量及方法。您还可以在代码中引用MaxCompute内置的第三方库或引用文件、表资源。更多信息，请参见第三方库或引用资源。

- 实现Python类的方法：必选。

  Python类实现包含如下4个方法，您可以根据实际需要进行选择。

方法定义	描述
`BaseUDAF.new_buffer()`	返回聚合函数的中间值的buffer。 `buffer` 必须是Marshal对象（例如LIST、DICT），并且 `buffer` 的大小不应该随数据量递增。在极限情况下， `buffer` 在执行对象序列化后的大小不应该超过2 MB。
`BaseUDAF.iterate(buffer[, args, ...])`	将 `args` 聚合到中间值 `buffer` 中。
`BaseUDAF.merge(buffer, pbuffer)`	将中间值 `buffer` 和 `pbuffer` 合并的结果存放在 `buffer` 中。
`BaseUDAF.terminate(buffer)`	将 `buffer` 转换为MaxCompute SQL的基本类型。

UDAF代码示例如下。

```
#导入函数签名模块及基类。
from odps.udf import annotate
from odps.udf import BaseUDAF
#函数签名。
@annotate('double->double')
#自定义Python类。
class Average(BaseUDAF):
#实现Python类的方法。
    def new_buffer(self):
        return [0, 0]
    def iterate(self, buffer, number):
        if number is not None:
            buffer[0] += number
            buffer[1] += 1
    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]
    def terminate(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

> **说明** Python 2 UDAF与Python 3 UDAF区别在于底层Python语言版本不一致，请您根据对应版本语言支持的能力编写UDAF。

使用限制

Python 3与Python 2不兼容。在您使用Python 3之前，需要考虑兼容性问题，在一个SQL中不允许同时使用Python 3和Python 2。

Python 2 UDAF迁移

Python 2官方即将停止维护，建议您根据项目类型执行迁移操作：

- 全新项目：新MaxCompute项目，或第一次使用Python语言编写UDAF的MaxCompute项目。建议所有的Python UDAF都直接使用Python 3语言编写。
- 存量项目：创建了大量Python 2 UDAF的MaxCompute项目。请您谨慎开启Python 3。如果您计划逐步将所有Python 2 UDAF迁移为Python 3 UDAF，推荐方法如下：
  - 新作业和新UDAF：使用Python 3语言编写，在Session级别开启Python 3。开启Python 3方法，请参见开启Python 3。
  - Python 2 UDAF：改写Python 2 UDAF，使其可以同时兼容Python 2和Python 3。改写方法请参见将Python 2代码移植到Python 3。

> **说明** 如果您需要编写公共UDAF，并为多个MaxCompute项目授权UDAF的操作权限，建议UDAF同时兼容Python 2和Python 3。

开启Python 3

MaxCompute默认使用Python 2，如果您要使用Python 3，可以在Session级别设置如下属性开启Python 3，并与SQL语句一起提交执行。

```
set odps.sql.python.version=cp37;
```

第三方库

MaxCompute内置的Python 3运行环境中未安装第三方库Numpy。如果您需要使用Numpy的UDAF，请手动上传Numpy的WHEEL包。从PyPI或镜像下载Numpy包时，包的文件名为*numpy-<版本号>-cp37-cp37m-manylinux1_x86_64.whl*。上传包的操作请参见资源操作或Python UDF使用第三方包。

函数签名及数据类型

函数签名格式如下。

```
@annotate(<signature>)
```

`signature` 为字符串，用于标识输入参数和返回值的数据类型。执行UDAF时，UDAF函数的输入参数和返回值类型要与函数签名指定的类型一致。查询语义解析阶段会检查不符合函数签名定义的用法，检查到类型不匹配时会报错。具体格式如下。

```
'arg_type_list -> type'
```

其中：

- `arg_type_list` ：表示输入参数的数据类型。输入参数可以为多个，用英文逗号（,）分隔。支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、CHAR、VARCHAR、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

  `arg_type_list` 还支持星号（*）或为空（''）：

  - 当 `arg_type_list` 为星号（*）时，表示输入参数为任意个数。
  - 当 `arg_type_list` 为空（''）时，表示无输入参数。
- `type` ：表示返回值的数据类型。UDAF只返回一列。支持的数据类型为：BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME、DECIMAL、FLOAT、BINARY、DATE、DECIMAL(precision,scale)、复杂数据类型（ARRAY、MAP、STRUCT）或复杂数据类型嵌套。

> **说明** 在编写UDAF代码过程中，您可以根据MaxCompute项目的数据类型版本选取合适的数据类型，更多数据类型版本及各版本支持的数据类型信息，请参见数据类型版本说明。

合法函数签名示例如下。

函数签名示例	说明
`@annotate('bigint,double->string')`	输入参数类型为BIGINT、DOUBLE，返回值类型为STRING。
`@annotate('*->string')`	输入任意个参数，返回值类型为STRING。
`@annotate('->double')`	无输入参数，返回值类型为DOUBLE。
`@annotate('array<bigint>->struct<x:string, y:int>')`	输入参数类型为ARRAY<BIGINT>，返回值类型为STRUCT<x:STRING, y:INT>。

为确保编写Python UDAF过程中使用的数据类型与MaxCompute支持的数据类型保持一致，您需要关注二者间的数据类型映射关系。具体映射关系如下。

MaxCompute SQL Type	Python 3 Type
BIGINT	INT
STRING	UNICODE
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	DATETIME.DATETIME
FLOAT	FLOAT
CHAR	UNICODE
VARCHAR	UNICODE
BINARY	BYTES
DATE	DATETIME.DATE

MaxCompute SQL Type	Python 3 Type
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT
STRUCT	COLLECTIONS.NAMEDTUPLE

引用资源

Python UDAF可以通过 `odps.distcache` 模块引用资源，支持引用文件资源和表资源。

- `odps.distcache.get_cache_file(resource_name)` ：返回指定文件资源的内容。
  - `resource_name` 为STRING类型，对应当前MaxCompute项目中已存在的文件资源名。如果文件资源名非法或者没有相应的文件资源，会返回异常。

    > **说明** 使用UDAF访问资源，在创建UDAF时需要声明引用的资源，否则会报错。

  - 返回值为File-like对象。在使用完此对象后，您需要调用 `close` 方法释放打开的资源文件。
- `odps.distcache.get_cache_table(resource_name)` ：返回指定表资源的内容。
  - `resource_name` 支持STRING类型，对应当前MaxCompute项目中已存在的表资源名。如果表资源名非法或者没有相应的表资源，会返回异常。
  - 返回值为GENERATOR类型，调用者以遍历方式获取表的内容，每次遍历可得到以数组形式存在的表中的一条记录。

具体使用方法请参见引用资源（Python UDF 3）和引用资源（Python UDTF 3）。

使用说明

按照开发流程，完成Python 3 UDAF开发后，您即可通过MaxCompute SQL调用Python 3 UDAF。调用方法如下：

- 在归属MaxCompute项目中使用自定义函数：使用方法与内建函数类似，您可以参照内建函数的使用方法使用自定义函数。
- 跨项目使用自定义函数：即在项目A中使用项目B的自定义函数，跨项目分享语句示例：`select B:udf_in_other_project(arg0, arg1) as res from table_t;` 。更多跨项目分享信息，请参见基于Package跨项目访问资源。

使用MaxCompute Studio完整开发及调用Python 3 UDAF的操作，请参见开发Python UDF。

3.10.5. 代码嵌入式UDF

本文为您介绍如何通过代码嵌入式UDF（Embedded UDF）将Java或Python代码嵌入SQL脚本。

背景信息

您可以通过MaxCompute的代码嵌入式UDF解决以下代码实现过程繁琐，且不方便阅读和维护的问题：

- 创建UDF并完成代码开发后，您还需要完成代码编译（Java）、创建资源和创建函数操作，过程比较繁琐。
- SQL脚本中如果包含UDF，您无法直接查看实现逻辑，或无法获取到JAR包的源码，维护不方便。
- 通过UDT使用Java库函数时，您需要编写长代码将Java代码写为表达式，阅读和维护不方便。另外还会存在Java代码无法写为表达式的问题，命令示例如下。

```
Foo f = new Foo();
f.execute();
f.getResult();
```

功能介绍

代码嵌入式UDF支持将Java或Python代码嵌入SQL脚本。Janino-compiler编译器会识别并提取嵌入的代码，完成代码编译（Java）、动态生成资源和创建临时函数操作。

代码嵌入式UDF允许您将SQL脚本和第三方代码放入同一个源码文件，减少使用UDT或UDF的操作步骤，方便日常开发。

使用限制

嵌入式Java代码使用Janino-compiler编译器进行编译，且支持的Java语法只是标准Java JDK的一个子集。嵌入式Java代码使用限制包含但不限于以下内容：

- 不支持Lambda表达式。
- 不支持Catch多种Exception类型。例如 `catch(Exception1 | Exception2 e)` 。
- 不支持自动推导泛型。例如 `Map map = new HashMap<>();` 。
- 类型参数的推导会被忽略，必须显示Cast。例如 `(String) myMap.get(key)` 。
- Assert会强制开启，不受JVM的-ea参数控制。
- 不支持Java 8以上（不包含Java 8）版本的语言功能。

UDT引用嵌入式代码

示例代码如下。您需要通过脚本模式提交执行，详情请参见脚本模式SQL。

```
SELECT
  s,
  com.mypackage.Foo.extractNumber(s)
FROM VALUES ('abc123def'),('apple') AS t(s);
#CODE ('lang'='JAVA')
package com.mypackage;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Foo {
  final static Pattern compile = Pattern.compile(".*?([0-9]+).*");
  public static String extractNumber(String input) {
    final Matcher m = compile.matcher(input);
    if (m.find()) {
      return m.group(1);
    }
    return null;
  }
}
#END CODE;
```

- `#CODE` 和 `#END CODE` ：表示嵌入式代码的开始和结束位置。位于脚本末尾的嵌入式代码块作用域为整个脚本。
- `'lang'='JAVA'` ：表示嵌入式代码为Java代码。还支持 `PYTHON` 。
- 在SQL脚本里可以使用UDT语法直接调用 `Foo.extractNumber` 。

Java代码嵌入式UDF

示例代码如下。

```
CREATE TEMPORARY FUNCTION foo AS 'com.mypackage.Reverse' USING
#CODE ('lang'='JAVA')
package com.mypackage;
import com.aliyun.odps.udf.UDF;
public class Reverse extends UDF {
  public String evaluate(String input) {
    if (input == null) return null;
    StringBuilder ret = new StringBuilder();
    for (int i = input.toCharArray().length - 1; i >= 0; i--) {
      ret.append(input.toCharArray()[i]);
    }
    return ret.toString();
  }
}
#END CODE;
SELECT foo('abdc');
```

- 嵌入式代码块可以置于 `USING` 后或脚本末尾，置于 `USING` 后的代码块作用域仅为 `CREATE TEMPORARY FUNCTION` 语句。
- `CREATE TEMPORARY FUNCTION` 创建的函数为临时函数，仅在本次执行生效，不会存入MaxCompute的Meta系统。

Java代码嵌入式UDTF

示例代码如下。

```
CREATE TEMPORARY FUNCTION foo AS 'com.mypackage.Reverse' USING
#CODE ('lang'='JAVA', 'filename'='embedded.jar')
package com.mypackage;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve({"string->string,string"})
public class Reverse extends UDTF {
  @Override
  public void process(Object[] objects) throws UDFException {
    String str = (String) objects[0];
    String[] split = str.split(",");
    forward(split[0], split[1]);
  }
}
#END CODE;
SELECT foo('ab,dc') AS (a,b);
```

由于 `@Resolve` 返回值要求为 `string[]` ，但Janino-compiler编译器无法将 `"string->string,string"` 识别为 `string[]` ， `@Resolve` 注解的参数需要加大括号（ `{}` ），为嵌入式代码特有内容。用普通方式创建Java UDTF时可省略大括号（ `{}` ）。

Python代码嵌入式UDF

示例代码如下。

```
CREATE TEMPORARY FUNCTION foo AS 'embedded.UDFTest' USING
#CODE ('lang'='PYTHON', 'filename'='embedded')
from odps.udf import annotate
@annotate("bigint->bigint")
class UDFTest(object):
  def evaluate(self, a):
    return a * a
#END CODE;
SELECT foo(4);
```

- Python代码的缩进需要符合Python语言规范。
- 由于注册Python UDF时 `AS` 后的类名需要包含Python源码的文件名，您可以通过 `'filename'='embedded'` 指定一个虚拟文件名。

3.10.6. SQL语言定义函数

本文为您介绍如何通过SQL语言定义函数（SQL Function）在SQL脚本中使用SQL定义的UDF。

背景信息

SQL语言定义函数作为一种用户自定义函数，弥补了MaxCompute只能用Java或Python创建UDF的不足，还扩展了函数类型的参数和匿名函数特性，提升表达业务逻辑的灵活性。您可以通过该函数实现简单功能，提高代码复用率。具体功能如下：

- 支持在SQL脚本中使用SQL定义的UDF，并调用UDF。
- 支持函数类型的参数，调用时可以传入内置函数、UDF或SQL语言定义函数。
- 函数类型的参数可以为匿名函数，调用时可以传入匿名函数。

您可以通过MaxCompute的SQL语言定义函数解决如下问题：

- 代码中通常会存在很多相似部分，维护不方便，且容易出错。如果引入UDF并编写代码后，您还需要进行代码编译（Java）、创建资源和创建函数操作，过程比较繁琐，且性能不如内建函数。示例如下。

  ```
  select
      nvl(str_to_map(get_json_object(col, '$.key1')), 'default') as key1,
      nvl(str_to_map(get_json_object(col, '$.key2')), 'default') as key2,
      ...
      nvl(str_to_map(get_json_object(col, '$.keyN')), 'default') as keyN
  from t;
  ```

- 使用类似Java中Lambda表达式的功能，把函数作为参数传给另一个函数。

注意事项

在调用SQL语言定义函数时，为避免出现函数解析失败的问题，请确保读取的数据类型满足SQL语言定义函数的数据类型要求。

在创建、查询、调用、删除SQL自定义函数时，需要具备Function级别的权限。更多Function权限及授权操作，请参见权限列表。

创建永久SQL语言定义函数

创建永久SQL语言定义函数并存入Meta系统后，所有的查询操作都可以调用该函数，调用函数操作，请参见调用SQL语言定义函数。命令格式如下。

```
create sql function <function_name>(@<parameter_in1> <datatype>[, @<parameter_in2> <datatype>...])
[returns @<parameter_out> <datatype>]
as [begin]
<function_expression>
[end];
```

- function_name：必填。新建SQL语言定义函数的名称。函数名称需要唯一，同名函数只能注册一次，不能与系统内建函数同名。您可以通过 `list functions;` 命令查看项目下的全部函数。
- parameter_in：必填。函数的输入参数。支持函数类型参数或匿名函数参数。函数类型参数，请参见函数类型参数。匿名函数参数，请参见匿名函数参数。
- datatype：必填。参数的数据类型。
- returns：必填。函数返回值变量。如果不指定，默认返回function_name的同名变量。
- parameter_out：必填。函数返回参数。
- function_expression：必填。函数表达式。

命令示例如下。

```
create sql function my_add(@a BIGINT) as @a + 1;
```

`@a + 1` 为函数实现逻辑，可直接写为表达式，支持内置操作符、内建函数和UDF。

如果逻辑复杂，可以在定义中使用begin和end，内部可以编写多条语句。returns指定返回值变量，如果不指定，默认返回function_name的同名变量。命令示例如下。

```
create sql function my_sum(@a BIGINT, @b BIGINT, @c BIGINT) returns @my_sum BIGINT
as begin
    @temp := @a + @b;
    @my_sum := @temp + @c;
end;
```

> **说明** 写入多条语句时，需要使用脚本模式，详情请参见脚本模式SQL。

创建临时SQL语言定义函数

如果您不需要把SQL语言定义函数存入MaxCompute的Meta系统，可以使用临时SQL语言定义函数。临时SQL语言定义函数只在当前脚本有效。命令格式如下。

```
function <function_name>(@<parameter_in1> <datatype>[, @<parameter_in2> <datatype>...])
[returns @<parameter_out> <datatype>]
as [begin]
<function_expression>
[end];
```

详细参数说明，请参见创建永久SQL语言定义函数。

> **说明** 写入多条语句时，需要使用脚本模式，详情请参见脚本模式SQL。

命令示例如下。

```
function my_add(@a BIGINT) as @a + 1;
```

查询SQL语言定义函数的基本信息

查询SQL语言定义函数的方式与Java UDF或Python UDF保持一致。命令格式如下。

```
desc function <function_name>;
```

function_name：已创建的SQL语言定义函数的名称。

命令示例如下。

```
desc function my_add;
```

返回结果如下。

```
Name                                my_add
Owner                               ALIYUN$santie_doctest@test.aliyunid.com
Created Time                        2021-05-08 11:26:02
SQL Definition Text                 CREATE SQL FUNCTION MY_ADD(@a BIGINT) AS @a + 1
```

> **说明** 客户端版本需要升级至0.34.0以上。查看客户端版本及获取客户端操作，请参见使用客户端（odpscmd）连接。

删除SQL语言定义函数

删除SQL语言定义函数的方式与Java UDF或Python UDF保持一致。语法格式如下。

```
drop function <function_name>;
```

function_name：已创建的SQL语言定义函数的名称。

命令示例如下。

```
drop function my_add;
```

调用SQL语言定义函数

调用SQL语言定义函数的方式和现有内建函数的调用方式一致。命令示例如下。

```
select <function_name>(<column_name>[,...]) from <table_name>;
```

- function_name：已创建的SQL语言定义函数的名称。
- column_name：待查询的目标表的列名称。列的数据类型必须与SQL语言定义函数定义的数据类型保持一致。
- table_name：待查询目标表的名称。

命令示例如下。

```
--创建目标表src。
create table src (c bigint, d string);
insert into table src values (1,100.1),(2,100.2),(3,100.3);
--调用my_add函数。
select my_add(c) from src;
--返回结果如下。
+------------+
| _c0        |
+------------+
| 2          |
| 3          |
| 4          |
+------------+
```

函数类型参数

调用SQL语言定义函数时，可以传入内建函数、UDF或SQL语言定义函数。命令示例如下。

```
function add(@a BIGINT) as @a + 1;
function op(@a BIGINT, @fun function (BIGINT) returns BIGINT) as @fun(@a);
select op(key, add), op(key, abs) from values (1),(2) as t (key);
--返回结果如下。
+------------+------------+
| _c0        | _c1        |
+------------+------------+
| 2          | 1          |
| 3          | 2          |
+------------+------------+
```

示例中，函数 `op` 定义了2个输入参数。 `@a` 定义一个BIGINT类型数值； `@fun` 定义一个函数，输入和输出均为BIGINT类型。函数 `op` 将 `@a` 传入 `@fun` 函数，并传入ADD和ABS函数，对 `@a` 进行操作。ABS函数详情请参见数学函数。

匿名函数参数

函数类型的参数可以为匿名函数，SQL语言定义函数调用时可以传入匿名函数。命令示例如下。

```
function op(@a BIGINT, @fun function (BIGINT) returns BIGINT) as @fun(@a);
select op(key, function (@a) as @a + 1) from values (1),(2) as t (key);
```

示例中， `function (@a) as @a + 1` 为匿名函数。匿名参数的输入参数 `@a` 不需要指定类型，编译器会根据OP函数的参数定义推导 `@a` 的类型。

典型示例

场景：将 `yyyy-mm-dd` 格式的日期转换为 `yyyymmdd` 格式。

假设待转换的日期为：2020-11-21、2020-1-01、2019-5-1和19-12-1。

处理方案如下：

- 方案1：通过创建SQL语言定义函数实现。**推荐**使用。命令示例如下：

  ```
  create sql function y_m_d2yyyymmdd(@y_m_d string) returns @yyyymmdd string
  as begin
      @yyyymmdd := concat(lpad(split_part(@y_m_d, '-', 1), 4, '0'), lpad(split_part(@y_m_d, '-', 2), 2, '0'), lpad(split_part(@y_m_d, '-', 3), 2, '0')) ;
  end;
  select y_m_d2yyyymmdd(d) from values('2020-11-21'),('2020-1-01'), ('2019-5-1'), ('19-12-1') t (d);
  ```

  返回结果如下：

```
+------------+
| _c0        |
+------------+
| 20201121   |
| 20200101   |
| 20190501   |
| 00191201   |
+------------+
```

- 方案2：该方案存在重复调用函数的问题，代码复用率较低，**不推荐**使用。命令示例如下：

```
select concat(lpad(split_part(d, '-', 1), 4, '0'), lpad(split_part(d, '-', 2), 2, '0'), lpad(split_part(d, '-', 3), 2, '0')) f
rom values('2020-11-21'),('2020-1-01'), ('2019-5-1'), ('19-12-1') t (d);
```

> 说明 写入多条语句时，需要使用脚本模式，详情请参见脚本模式SQL。

3.10.7. 开源地理空间UDF

本文为您介绍如何使用开源地理空间UDF分析空间数据。

前提条件

请确认您已完成以下操作：

- 已安装Git。
- 已安装Maven并设置环境变量。
- 已安装MaxCompute客户端。

  更多安装MaxCompute客户端信息，请参见安装并配置MaxCompute客户端。

背景信息

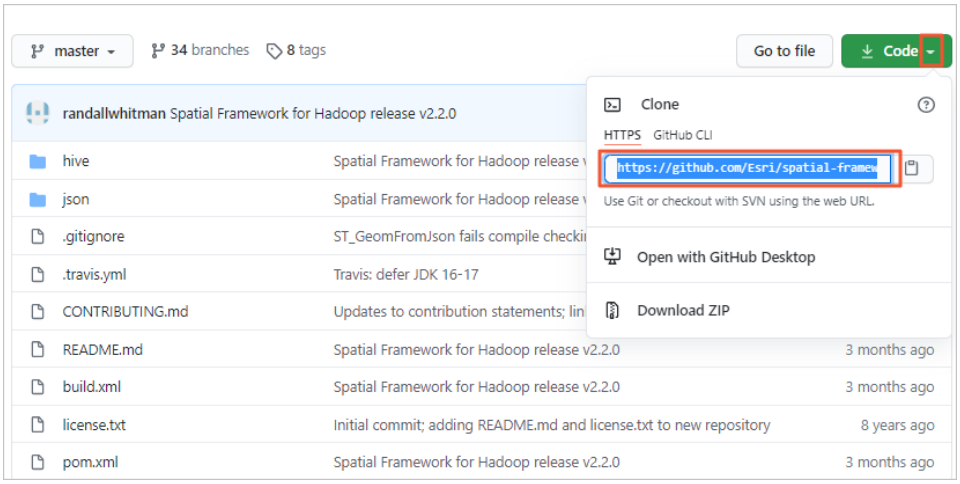
Apache Hive有一套开源的地理空间UDF，详情请参见Github。MaxCompute原生支持直接使用Hive UDF，因此也支持在MaxCompute中使用Hive地理空间函数。

MaxCompute使用Hive UDF的示例请参见兼容Hive UDF。

> 说明 在使用过程中，如果您有任何问题，请直接在GitHub上提交issues获取帮助。

步骤一：准备本地函数

1. 获取地理空间UDF代码的URL下载链接。



2. 打开Git命令行工具，下载2.1.0版本Hive（对应Hadoop版本为2.7.2）下的地理空间UDF代码至本地。

   命令示例如下。

```
git clone https://github.com/Esri/spatial-framework-for-hadoop.git
```

   或

```
git clone -b "v2.1.0" --single-branch git@github.com:Esri/spatial-framework-for-hadoop.git
```

3. 使用Maven构建项目。

   命令示例如下。

```
cd spatial-framework-for-hadoop
mvn clean package -DskipTests -P java-8,hadoop-2.7,hive-2.1
```

4. 复制构建好的JAR包。此JAR包包含开源地理空间UDF的所有方法。

   命令示例如下。

```
cp hive/target/spatial-sdk-hive-2.1.1-SNAPSHOT.jar .../spatial-sdk-hive.jar
```

5. 下载项目所依赖的JAR包。

   命令示例如下。

```
wget https://repo1.maven.org/maven2/com/esri/geometry/esri-geometry-api/2.2.0/esri-geometry-api-2.2.0.jar .../esri-geometry-api.jar
```

   或

```
cp ~/.m2/repository/com/esri/geometry/esri-geometry-api/2.2.0/esri-geometry-api-2.2.0.jar .../esri-geometry-api.jar
```

步骤二：在MaxCompute上注册函数

1. 在使用客户端（odpscmd）连接上执行如下命令将本地准备好的两个依赖JAR包上传至目标Project。

   更多添加资源信息，请参见添加资源。

```
add jar esri-geometry-api.jar;
add jar spatial-sdk-hive.jar;
```

2. 执行如下命令注册函数。

```
CREATE FUNCTION ST_Aggr_ConvexHull  AS  'com.esri.hadoop.hive.ST_Aggr_ConvexHull' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Aggr_Intersection    AS  'com.esri.hadoop.hive.ST_Aggr_Intersection'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Aggr_Union   AS  'com.esri.hadoop.hive.ST_Aggr_Union'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Area AS  'com.esri.hadoop.hive.ST_Area'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsBinary AS  'com.esri.hadoop.hive.ST_AsBinary'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsGeoJson    AS  'com.esri.hadoop.hive.ST_AsGeoJson'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsJson   AS  'com.esri.hadoop.hive.ST_AsJson'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsShape  AS  'com.esri.hadoop.hive.ST_AsShape' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsText   AS  'com.esri.hadoop.hive.ST_AsText'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Bin  AS  'com.esri.hadoop.hive.ST_Bin' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_BinEnvelope  AS  'com.esri.hadoop.hive.ST_BinEnvelope' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Boundary AS  'com.esri.hadoop.hive.ST_Boundary'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Buffer   AS  'com.esri.hadoop.hive.ST_Buffer'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Centroid AS  'com.esri.hadoop.hive.ST_Centroid'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Contains AS  'com.esri.hadoop.hive.ST_Contains'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_ConvexHull   AS  'com.esri.hadoop.hive.ST_ConvexHull'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_CoordDim AS  'com.esri.hadoop.hive.ST_CoordDim'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Crosses  AS  'com.esri.hadoop.hive.ST_Crosses' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Difference   AS  'com.esri.hadoop.hive.ST_Difference'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Dimension    AS  'com.esri.hadoop.hive.ST_Dimension'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Disjoint AS  'com.esri.hadoop.hive.ST_Disjoint'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Distance AS  'com.esri.hadoop.hive.ST_Distance'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_EndPoint AS  'com.esri.hadoop.hive.ST_EndPoint'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Envelope AS  'com.esri.hadoop.hive.ST_Envelope'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_EnvIntersects    AS  'com.esri.hadoop.hive.ST_EnvIntersects'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Equals   AS  'com.esri.hadoop.hive.ST_Equals'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_ExteriorRing AS  'com.esri.hadoop.hive.ST_ExteriorRing'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeodesicLengthWGS84  AS  'com.esri.hadoop.hive.ST_GeodesicLengthWGS84' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomCollection   AS  'com.esri.hadoop.hive.ST_GeomCollection'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Geometry AS  'com.esri.hadoop.hive.ST_Geometry'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeometryN    AS  'com.esri.hadoop.hive.ST_GeometryN'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeometryType AS  'com.esri.hadoop.hive.ST_GeometryType'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromGeoJson  AS  'com.esri.hadoop.hive.ST_GeomFromGeoJson' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromJson AS  'com.esri.hadoop.hive.ST_GeomFromJson'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromShape    AS  'com.esri.hadoop.hive.ST_GeomFromShape'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromText AS  'com.esri.hadoop.hive.ST_GeomFromText'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
```

```
CREATE FUNCTION ST_GeomFromWKB  AS  'com.esri.hadoop.hive.ST_GeomFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_InteriorRingN    AS  'com.esri.hadoop.hive.ST_InteriorRingN'   USING 'spatial-sdk-hive.jar,esri-geometry-a
pi.jar';
CREATE FUNCTION ST_Intersection AS  'com.esri.hadoop.hive.ST_Intersection'    USING 'spatial-sdk-hive.jar,esri-geometry-api.j
ar';
CREATE FUNCTION ST_Intersects   AS  'com.esri.hadoop.hive.ST_Intersects'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Is3D AS  'com.esri.hadoop.hive.ST_Is3D'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_IsClosed AS  'com.esri.hadoop.hive.ST_IsClosed'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_IsEmpty  AS  'com.esri.hadoop.hive.ST_IsEmpty' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_IsMeasured   AS  'com.esri.hadoop.hive.ST_IsMeasured'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_IsRing   AS  'com.esri.hadoop.hive.ST_IsRing'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_IsSimple AS  'com.esri.hadoop.hive.ST_IsSimple'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Length   AS  'com.esri.hadoop.hive.ST_Length'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_LineFromWKB  AS  'com.esri.hadoop.hive.ST_LineFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_LineString   AS  'com.esri.hadoop.hive.ST_LineString'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_M    AS  'com.esri.hadoop.hive.ST_M'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MaxM AS  'com.esri.hadoop.hive.ST_MaxM'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MaxX AS  'com.esri.hadoop.hive.ST_MaxX'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MaxY AS  'com.esri.hadoop.hive.ST_MaxY'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MaxZ AS  'com.esri.hadoop.hive.ST_MaxZ'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MinM AS  'com.esri.hadoop.hive.ST_MinM'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MinX AS  'com.esri.hadoop.hive.ST_MinX'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MinY AS  'com.esri.hadoop.hive.ST_MinY'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MinZ AS  'com.esri.hadoop.hive.ST_MinZ'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MLineFromWKB AS  'com.esri.hadoop.hive.ST_MLineFromWKB'    USING 'spatial-sdk-hive.jar,esri-geometry-api.j
ar';
CREATE FUNCTION ST_MPointFromWKB    AS  'com.esri.hadoop.hive.ST_MPointFromWKB'   USING 'spatial-sdk-hive.jar,esri-geometry-a
pi.jar';
CREATE FUNCTION ST_MPolyFromWKB AS  'com.esri.hadoop.hive.ST_MPolyFromWKB'    USING 'spatial-sdk-hive.jar,esri-geometry-api.j
ar';
CREATE FUNCTION ST_MultiLineString  AS  'com.esri.hadoop.hive.ST_MultiLineString' USING 'spatial-sdk-hive.jar,esri-geometry-a
pi.jar';
CREATE FUNCTION ST_MultiPoint   AS  'com.esri.hadoop.hive.ST_MultiPoint'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MultiPolygon AS  'com.esri.hadoop.hive.ST_MultiPolygon'    USING 'spatial-sdk-hive.jar,esri-geometry-api.j
ar';
CREATE FUNCTION ST_NumGeometries    AS  'com.esri.hadoop.hive.ST_NumGeometries'   USING 'spatial-sdk-hive.jar,esri-geometry-a
pi.jar';
CREATE FUNCTION ST_NumInteriorRing  AS  'com.esri.hadoop.hive.ST_NumInteriorRing' USING 'spatial-sdk-hive.jar,esri-geometry-a
pi.jar';
CREATE FUNCTION ST_NumPoints    AS  'com.esri.hadoop.hive.ST_NumPoints'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Overlaps AS  'com.esri.hadoop.hive.ST_Overlaps'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Point    AS  'com.esri.hadoop.hive.ST_Point'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_PointFromWKB AS  'com.esri.hadoop.hive.ST_PointFromWKB'    USING 'spatial-sdk-hive.jar,esri-geometry-api.j
ar';
CREATE FUNCTION ST_PointN   AS  'com.esri.hadoop.hive.ST_PointN'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_PointZ   AS  'com.esri.hadoop.hive.ST_PointZ'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_PolyFromWKB  AS  'com.esri.hadoop.hive.ST_PolyFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Polygon  AS  'com.esri.hadoop.hive.ST_Polygon' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Relate   AS  'com.esri.hadoop.hive.ST_Relate'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_SetSRID  AS  'com.esri.hadoop.hive.ST_SetSRID' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_SRID AS  'com.esri.hadoop.hive.ST_SRID'    USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_StartPoint   AS  'com.esri.hadoop.hive.ST_StartPoint'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_SymmetricDiff    AS  'com.esri.hadoop.hive.ST_SymmetricDiff'   USING 'spatial-sdk-hive.jar,esri-geometry-a
pi.jar';
CREATE FUNCTION ST_Touches  AS  'com.esri.hadoop.hive.ST_Touches' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Union    AS  'com.esri.hadoop.hive.ST_Union'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Within   AS  'com.esri.hadoop.hive.ST_Within'  USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_X    AS  'com.esri.hadoop.hive.ST_X'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Y    AS  'com.esri.hadoop.hive.ST_Y'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Z    AS  'com.esri.hadoop.hive.ST_Z'   USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
```

步骤三：测试UDF

通过MaxCompute客户端提交SQL语句，测试UDF是否可以正常使用。命令示例如下。

```
--打开Hive兼容性并提交测试函数。
set odps.sql.hive.compatible=true;
select ST_AsText(ST_Point(1, 2));
```

返回结果如下，说明安装成功。

```
+-----+
| _c0 |
+-----+
| POINT (1 2) |
+-----+
```

3.10.8. MaxCompute UDF常见问题

3.10.8.1. MaxCompute UDF（Java）常见问题

本文为您介绍使用Java语言编写的MaxCompute UDF的常见问题。

类或依赖问题

调用MaxCompute UDF运行代码时的常见类或依赖问题如下：

- 问题现象一：**运行报错描述为** `ClassNotFoundException` **或** `Some dependencies are missing` 。
  - 产生原因：
    - 原因一：创建MaxCompute UDF时指定的资源JAR包不正确。
    - 原因二：MaxCompute UDF依赖的资源JAR包未上传至MaxCompute，例如依赖的第三方包没有上传。
    - 原因三：调用MaxCompute UDF运行代码时，所处的项目不正确。即MaxCompute UDF不在MaxCompute项目中。例如MaxCompute UDF注册到了开发项目，但却在生产项目执行调用操作。
    - 原因四：文件资源不存在或资源类型不正确。例如PY文件，资源类型是PY，但MaxCompute UDF代码中 `get_cache_file` 需要的类型是FILE。
  - 解决措施：
    - 原因一的解决措施：检查JAR包的正确性，并确认JAR包中已经包含需要的类，重新打包并上传至MaxCompute项目。更多打包上传操作，请参见打包、上传及注册。
    - 原因二的解决措施：将MaxCompute UDF依赖的第三方包作为资源上传至MaxCompute项目，然后在注册函数时，依赖资源列表中添加此包。更多上传资源及注册函数操作，请参见添加资源和注册函数。
    - 原因三的解决措施：在报错的项目下通过MaxCompute客户端执行 `list functions;` 命令，确保MaxCompute UDF是真实存在的，且MaxCompute UDF的类和依赖的资源正确。
    - 原因四的解决措施：通过MaxCompute客户端执行 `desc function <function_name>;` 命令，确保Resources列表已覆盖所有需要的文件资源。如果资源类型不匹配，可执行 `add <file_type> <file_name>;` 重新添加资源。
- 问题现象二：**运行报错描述为** `NoClassDefFoundError` **或** `NoSuchMethodError` 。
  - 产生原因：
    - 原因一：用户上传的JAR包中包含的第三方库的版本与MaxCompute内置的第三方库的版本不一致。
    - 原因二：Java沙箱限制。作业Instance的Stderr中出现 `java.security.AccessControlException: access denied ("java.lang.RuntimePermission" "createClassLoader")` 详细信息，表示是沙箱限制。MaxCompute UDF在分布式环境中运行时收到Java沙箱的限制。更多Java沙箱限制信息，请参见Java沙箱。
  - 解决措施：
    - 原因一的解决措施：使用 `maven-shade-plugin` 解决版本不一致问题并修改Import路径，重新打包为JAR包并上传至MaxCompute项目。更多打包上传操作，请参见打包、上传及注册。
    - 原因二的解决措施：请参见Java沙箱限制问题。

Java沙箱限制问题

- 问题现象：**调用MaxCompute UDF访问本地文件、外网或分布式文件系统，创建Java线程等时，代码运行会报错。**
- 产生原因：
  - 原因一：MaxCompute UDF在分布式环境中运行时受到Java沙箱的限制。更多Java沙箱限制信息，请参见Java沙箱。
  - 原因二：MaxCompute UDF不支持访问外网。
- 解决措施：
  - 原因一的解决措施：在Session级别通过 `set odps.isolation.session.enable=true;` 开启沙箱，如果开启沙箱后运行还报错，请填写钉钉群申请表单加入钉钉群，联系MaxCompute技术支持团队进行反馈。
  - 原因二的解决措施：请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

性能问题

调用MaxCompute UDF运行代码时的常见性能问题如下：

- 问题现象一：**运行报错描述为** `kInstanceMonitorTimeout` 。
  - 产生原因：MaxCompute UDF处理时间过长导致超时。默认情况下UDF处理数据的时间是有限制，在处理一批（通常情况下为1024条）记录时，必须在1800秒内处理完。这个时间限制并不是针对Worker的总运行时间，而是处理一小批记录的时间。通常情况下SQL处理数据的速率超过了万条/秒，该限制只是为了防止MaxCompute UDF中出现死循环，导致长时间占用CPU资源的情况。

- 解决措施：
  - 如果实际计算量很大，可以在MaxCompute UDF的实现Java类的方法中调用`ExecutionContext.claimAlive`来重置计时器。
  - 重点优化MaxCompute UDF代码逻辑。后续调用MaxCompute UDF时，可同时在Session级别配置如下参数辅助调节MaxCompute UDF运行过程，提升处理速度。

参数	说明
`set odps.function.timeout=xxx;`	调整UDF运行超时时长。默认值为1800s。可根据实际情况酌情调大。取值范围为1s~3600s。
`set odps.stage.mapper.split.size=xxx;`	调整Map Worker的输入数据量。默认值为256 MB。可根据实际情况酌情调小。
`set odps.sql.executionengine.batch.rowcount=xxx;`	调整MaxCompute一次处理的数据行数。默认值为1024行。可根据实际情况酌情调小。

- 问题现象二：**运行报错描述为`errMsg:SigKill(OOM)`或`OutOfMemoryError`。**
  - 产生原因：MaxCompute运行作业主要分为三个阶段：Map、Reduce和Join。如果处理的数据量比较大，会导致各个阶段的每个Instance处理的时间比较长。
  - 解决措施：
    - 如果是`fuxi`或`runtime`相关代码报错，您可以通过设置如下资源参数提升处理速度。

参数	说明
`set odps.stage.mapper.mem=xxx;`	调整Map Worker的内存大小。默认值为1024 MB。可根据实际情况酌情调大。
`set odps.stage.reducer.mem=xxx;`	调整Reduce Worker的内存大小。默认值为1024 MB。可根据实际情况酌情调大。
`set odps.stage.joiner.mem=xxx;`	调整Join Worker的内存大小。默认值为1024 MB。可根据实际情况酌情调大。
`set odps.stage.mapper.split.size=xxx;`	调整Map Worker的输入数据量。默认值为256 MB。可根据实际情况酌情调大。
`set odps.stage.reducer.num=xxx;`	调整Reduce阶段的Worker数量。可根据实际情况酌情调大。
`set odps.stage.joiner.num=xxx;`	调整Join阶段的Worker数量。可根据实际情况酌情调大。

    - 如果是Java代码本身报错，可以在调整上述参数的同时，通过`set odps.sql.udf.jvm.memory=xxx;`参数调大Jvm内存。

更多参数详细信息，请参见SET操作。

UDTF相关问题

调用Java UDTF运行代码时的常见问题如下：

- 问题现象一：**运行报错描述为`Semantic analysis exception - only a single expression in the SELECT clause is supported with UDTF's`。**
  - 产生原因：在SELECT语句中调用Java UDTF时，存在Java UDTF与其他列或表达式混用的问题，Java UDTF暂不支持该用法。错误示例如下。

```
select b.*, 'x', udtffunction_name(v) from table lateral view udtffunction_name(v) b as f1, f2;
```

  - 解决措施：您可以将Java UDTF与Lateral View配合使用。命令示例如下。

```
select b.*, 'x' from table lateral view udtffunction_name(v) b as f1, f2;
```

- 问题现象二：**运行报错描述为`Semantic analysis exception - expect 2 aliases but have 0`。**
  - 产生原因：Java UDTF代码中没有指定输出列名。
  - 解决措施：您可以在调用Java UDTF的SELECT语句中通过`as`子句给出列名。命令示例如下。

```
select udtffunction_name(paramname) as (col1, col2);
```

3.10.8.2. MaxCompute UDF（Python）常见问题

本文为您介绍使用Python语言编写的MaxCompute UDF的常见问题。

类或资源问题

调用MaxCompute UDF运行代码时的常见类或资源问题如下：

- 问题现象一：**运行报错描述为`function 'xxx' cannot be resolved`。**

- 产生原因：
  - 原因一：调用MaxCompute UDF运行代码时，所处的项目不正确。即MaxCompute UDF不在MaxCompute项目中。例如MaxCompute UDF注册到了开发项目，但却在生产项目执行调用操作。
  - 原因二：MaxCompute UDF的类不正确或资源不正确。
  - 原因三：MaxCompute UDF依赖的资源类型不正确。例如PY文件，资源类型是PY，但MaxCompute UDF代码中 `get_cache_file` 需要的类型是FILE。
  - 原因四：MaxCompute UDF依赖的资源不是最新的。当您通过DataWorks上传MaxCompute资源时，从DataWorks同步至MaxCompute会存在延时情况，非最新资源。
  - 原因五：Python环境版本不正确。MaxCompute默认采用Python 2运行作业，当Python代码中存在非ASCII编码字符时，运行会报错。
- 解决措施：
  - 原因一的解决措施：在报错的项目下通过MaxCompute客户端执行 `list functions;` 命令，确保MaxCompute UDF是真实存在的。
  - 原因二的解决措施：通过MaxCompute客户端执行 `desc function <function_name>;` 命令，检查输出结果中的Class及Resources的正确性。

    如果不正确，需要执行 `create function <function_name> as <'package_to_class'> using <'resource_list'>;` 命令重新注册函数，package_to_class为Python脚本名.类名，resource_list为MaxCompute中需要引用的所有文件资源、表资源、压缩包资源或第三方包。

    更多注册函数操作，请参见注册函数。
  - 原因三的解决措施：通过MaxCompute客户端执行 `desc resource <resource_name>;` 命令，检查输出结果中的Type的正确性。如果资源类型不正确，可执行 `add <file_type> <file_name>;` 重新添加资源。
    - 如果MaxCompute UDF代码中的引用资源方式为 `get_cache_file` ，表明引用的是文件资源，资源类型必须为FILE。
    - 如果MaxCompute UDF代码中的引用资源方式为 `get_cache_table` ，表明引用的是表资源，资源类型必须为TABLE。
    - 如果MaxCompute UDF代码中的引用资源方式为 `get_cache_archive` ，表明引用的是压缩包资源，资源类型必须为ARCHIVE。

    更多上传资源操作，请参见添加资源。
  - 原因四的解决措施：通过MaxCompute客户端执行 `desc resource <resource_name>;` 命令，检查输出结果中的LastModifiedTime，确保为最近一次变更的时间。
  - 原因五的解决措施：在Python代码头部增加 `#coding:utf-8` 或 `# -*- coding: utf-8 -*-` 编码声明，或在调用MaxCompute UDF的SQL语句前增加 `set odps.sql.python.version=cp37;` 与SQL语句一起提交，在Python 3环境下运行作业。

- 问题现象二：MaxCompute UDF中使用 `get_cache_archive('xxx.zip')` 时，**运行报错描述为** `IOError: Download resource: xxx.zip failed` 、 `odps.distcache.DistributedCacheError` **或** `fuxi job failed: Download resource failed: xxx.zip` 。
  - 产生原因：
    - 原因一：压缩包资源不存在。注册MaxCompute UDF时未同步指定压缩包资源。
    - 原因二：压缩包资源类型不正确，非ARCHIVE。
    - 原因三：压缩包资源名及后缀格式与实际资源包名或后缀格式不一致。例如压缩包资源名及后缀为xxx.zip，但实际上传的文件是xxx.tar.gz，此时会按ZIP格式解压，导致解压失败报错。
    - 原因四：同一个作业中有两个UDF依赖了不同项目下的同名资源。
  - 解决措施：
    - 原因一的解决措施：通过MaxCompute客户端执行 `desc function <function_name>;` 命令，检查输出结果中的Resources是否包含报错信息中的压缩资源包。

      如果不存在，可执行 `create function <function_name> as <'package_to_class'> using <'resource_list'>;` 命令重新注册函数，并在resource_list中加上缺失的压缩包资源。

      更多注册函数操作，请参见注册函数。
    - 原因二的解决措施：通过MaxCompute客户端执行 `desc resource <resource_name>;` 命令，检查输出结果中的Type是否为ARCHIVE。

      如果不是ARCHIVE类型，可执行 `add archive <file_name>;` 命令重新上传资源。

      更多上传资源操作，请参见添加资源。
    - 原因三的解决措施：通过MaxCompute客户端执行 `desc function <function_name>;` 命令，检查输出结果中的Resources中的压缩包资源名及后缀是否与实际文件名、后缀一致。

      如果不一致，可执行 `add archive <file_name>;` 命令重新上传资源，file_name必须与实际压缩包资源名及后缀保持一致。
    - 原因四的解决措施：排查作业依赖的所有UDF（包括视图中依赖的UDF），检查UDF所属项目和对应资源的名称。如果不同项目下存在同名资源，建议修改依赖的UDF或资源名称。

- 问题现象三：MaxCompute UDF中使用 `get_cache_table(table_name)` 时，**运行报错描述为** `odps.distcache.DistributedCacheError: Table resource "xxx_table_name" not found` 。
  - 产生原因：
    - 原因一：表资源不存在。注册MaxCompute UDF时未同步指定表资源。
    - 原因二：表资源类型不正确，非TABLE。

- 解决措施：
  - 原因一的解决措施：通过MaxCompute客户端执行 `desc function <function_name>;` 命令，检查输出结果中的Resources是否包含报错信息中的表资源。

    如果不存在，可执行 `create function <function_name> as <'package_to_class'> using <'resource_list'>;` 命令重新注册函数，并在resource_list中加上缺失的表资源。

    更多注册函数操作，请参见注册函数。
  - 原因二的解决措施：通过MaxCompute客户端执行 `desc resource <resource_name>;` 命令，检查输出结果中的Type是否为TABLE。

    如果不是TABLE类型，可执行 `add table <table_name>;` 命令重新上传表资源。

    更多上传资源操作，请参见添加资源。

- 问题现象四：MaxCompute UDF引用第三方包时，**运行报错描述为** `ImportError: No module named 'xxx'` 。
  - 产生原因：
    - 原因一：第三方包的资源类型不正确，非ARCHIVE。
    - 原因二：注册MaxCompute UDF时，未同步指定第三方包。
    - 原因三：MaxCompute UDF代码中未添加第三方包路径。
    - 原因四：第三方包为WHEEL包，但后缀不正确。您需要根据Python环境版本下载对应的WHEEL文件。
    - 原因五：第三方包不是WHEEL包，且非纯Python包，但包中存在setup.py文件。
    - 原因六：MaxCompute UDF对应的PY文件名称与需要引用的第三方模块的名称冲突。例如MaxCompute UDF对应的Python文件是A.py，import A时默认会导入A.py而不是三方包里的模块。
  - 解决措施：
    - 原因一的解决措施：通过MaxCompute客户端执行 `desc resource <resource_name>;` 命令，检查输出结果中的Type是否为ARCHIVE。

      如果不是ARCHIVE类型，可执行 `add archive <file_name>;` 命令重新上传资源。

      更多上传资源操作，请参见添加资源。
    - 原因二的解决措施：通过MaxCompute客户端执行 `desc function <function_name>;` 命令，检查输出结果中的Resources是否包含第三方包。

      如果不包含，可执行 `create function <function_name> as <'package_to_class'> using <'resource_list'>;` 命令重新注册函数，并在resource_list中加上第三方包。

      更多注册函数操作，请参见注册函数。
    - 原因三的解决措施：检查MaxCompute UDF代码中是否添加了第三方包路径，即是否配置了 `sys.path.insert(0, 'work/第三方包路径')` 。假设模块名称为A，对应Python文件为A.py，确定其资源包路径及在代码中添加路径的方法如下：
      - 假设PY文件位于文件夹resource_dir中，如果将文件夹resource_dir直接压缩为resource-of-A.zip， `sys.path.insert` 中填写的路径为 `work/resource-of-A.zip/resource_dir/` 。
      - 假设PY文件位于文件夹resource_dir中，如果将文件夹resource_dir内的所有文件压缩为resource-of-A.zip， `sys.path.insert` 中填写的路径为 `work/resource-of-A.zip/` 。
      - 假设PY文件位于文件夹resource_dir/path1/path2中，如果将文件夹resource_dir内的所有文件压缩为resource-of-A.zip， `sys.path.insert` 中填写的路径为 `work/resource-of-A.zip/path1/path2/` 。

      > **说明** ARCHIVE资源默认放在MaxCompute UDF执行路径的相对路径 `./work/` 中。
    - 原因四的解决措施：Python 2和Python 3环境对应的WHEEL文件不相同，Python 2要求WHEEL文件名称中包含 `cp27-cp27m-manylinux1_x86_64` ，Python 3要求WHEEL文件名称中包含 `cp37-cp37m-manylinux1_x86_64` ，请下载合适的WHEEL文件。下载的WHEEL文件可以直接修改后缀为.zip，不需要对WHEEL文件再次打包生成ZIP文件。
    - 原因五的解决措施：需要先在与MaxCompute兼容的环境下将setup.py编译生成WHEEL包，然后再执行上传资源及注册函数操作。更多编译第三方包信息，请参见使用需要编译的第三方包。
    - 原因六的解决措施：修改MaxCompute UDF对应的Python文件名称。

- 问题现象五：MaxCompute UDF引用Python 3的标准库时，**运行报错描述为** `ImportError: No module named enum` 。
  - 产生原因：MaxCompute项目未开启Python 3，默认使用Python 2环境运行MaxCompute UDF，无法识别Python 3的标准库。
  - 解决措施：在调用MaxCompute UDF的SQL语句前增加 `set odps.sql.python.version=cp37;` 与SQL语句一起提交执行。

- 问题现象六：**运行报错描述为** `failed to get Udf info from xxx.py` 。
  - 产生原因：编写的UDTF或UDAF代码中，基类的导入写法不正确。例如 `import odps.udf.BaseUDTF` 或 `import odps.udf.BaseUDAF` 。
  - 解决措施：修改为 `from odps.udf import BaseUDTF` 或 `from odps.udf import BaseUDAF` 。

性能问题

- 问题现象：**运行报错描述为** `kInstanceMonitorTimeout` 。
- 产生原因：MaxCompute UDF处理时间过长导致超时。默认情况下UDF处理数据的时间是有限制，在处理一批（通常情况下为1024条）记录时，必须在1800秒内处理完。这个时间限制并不是针对Worker的总运行时间，而是处理一小批记录的时间。通常情况下SQL处理数据的速率超过了万条/

秒，该限制只是为了防止MaxCompute UDF中出现死循环，导致长时间占用CPU资源的情况。

- 解决措施：
  - 在MaxCompute UDF代码中增加日志，用于检查代码中是否有死循环问题，或者可以在日志里打印时间信息来检查MaxCompute UDF处理单条数据的时长是否符合预期。代码中需要增加如下打印日志相关信息，作业运行成功后，您可以在Logview的StdOut中获取到日志信息。
    - Python 2环境

      ```
      sys.stdout.write('your log')
      sys.stdout.flush()
      ```

    - Python 3环境

      ```
      print('your log', flush=True)
      ```

  - 如果实际计算量很大，MaxCompute UDF预计的运行时间很长，您可以通过调整如下参数避免超时报错。

参数	说明
`set odps.function.timeout=xxx;`	调整UDF运行超时时长。默认值为1800s。可根据实际情况酌情调大。取值范围为1s~3600s。
`set odps.sql.executionengine.batch.rowcount=xxx;`	调整MaxCompute一次处理的数据行数。默认值为1024行。可根据实际情况酌情调小。

网络问题

- 问题现象：**调用MaxCompute UDF访问外网时，代码运行会报错。**
- 产生原因：MaxCompute UDF不支持访问外网。
- 解决措施：请根据业务情况填写并提交网络连接申请表单，MaxCompute技术支持团队会及时联系您完成网络开通操作。表单填写指导，请参见网络开通流程。

沙箱问题

- 问题现象：**运行报错描述为** `RuntimeError: xxx has been blocked by sandbox` 。
- 产生原因：Python UDF中的某些函数调用被沙箱阻断了。
- 解决措施：
  - 在调用Python UDF的SQL语句前，增加 `set odps.isolation.session.enable=true;` 设置，与SQL语句一起提交执行。
  - 使用Python 3 UDF，默认会设置 `set odps.isolation.session.enable=true;` 。

编码问题

调用MaxCompute UDF运行代码时的常见编码问题如下：

- 问题现象一：**运行报错描述为** `SyntaxError: Non-ASCII charactor '\xe8' in file xxx. on line yyy` 。
  - 产生原因：MaxCompute UDF对应的Python文件中存在非ASCII编码字符，且运行在Python 2环境中。
  - 解决措施：
    - 在调用MaxCompute UDF的SQL语句前增加 `set odps.sql.python.version=cp37;` 与SQL语句一起提交，在Python 3环境下运行作业。
    - 将Python 2的默认解析器编码方式修改为UTF-8，即在Python文件开头添加如下语句。

      ```
      import sys
      reload(sys)
      sys.setdefaultencoding('utf-8')
      ```

- 问题现象二：调用Python 2 UDF时，**运行报错描述为** `UnicodeEncodeError: 'ascii' code can't encode characters in position x-y: ordinal not in range(128)` 。
  - 产生原因：函数签名中返回值类型是STRING，但MaxCompute UDF返回UNICODE类型的Python对象，假设对象名为ret。MaxCompute默认会将返回值ret按照ASCII编码格式转换为STR类型，返回 `str(ret)` 。当ret本身在ASCII编码范围内时，可以成功转换为STR类型，但如果ret不在ASCII编码范围内时，转换会失败并返回报错。
  - 解决措施：在Python代码的 `evaluate` 方法中增加如下语句。

    ```
    return ret.encode('utf-8')
    ```

- 问题现象三：调用Python 3 UDF时，**运行报错描述为** `UnicodeDecodeError: 'utf-8' codec can't decode byte xxx in position xxx: invalid continuation byte` 。
  - 产生原因：函数签名中输入参数类型是STRING，但是调用Python 3 UDF时输入的字符串不能按照UTF-8解码为STR类型的Python对象。

- 解决措施：
  - 避免向MaxCompute表中写入非UTF-8编码的字符串。

    例如，Python 2 UDF返回的Python对象是按GBK编码的STR，可以正常写入MaxCompute表中，但无法被Python 3 UDF读取，Python 2 UDF返回数据时建议转为UTF-8编码后再返回，例如返回 `ret.decode('gbk').encode('utf-8')` 。
  - 在SQL语句中使用内建函数 `is_encoding` 提前过滤掉非UTF-8编码的数据。代码示例如下。

    ```
    select py_udf(input_col) from example_table where is_encoding(input_col, 'utf-8', 'utf-8') = true;
    ```
  - 将Python代码中的函数签名输入参数类型修改为BINARY，并在SQL语句中将STRING类型列转换为BINARY类型作为Python 3 UDF入参。代码示例如下。

    ```
    select py_udf(cast(input_col as binary)) from example_table;
    ```

函数签名问题

调用MaxCompute UDF运行代码时的常见函数签名问题如下：

- 问题现象一：**运行报错描述为** `resolve annotation of class xxx for UDTF/UDF/UDAF yyy contains invalid content '<EOF>'` 。
  - 产生原因：MaxCompute UDF的输入或输出参数为复杂数据类型，但函数签名不合法。
  - 解决方案：修改函数签名中的复杂数据类型写法，确保为合法的函数签名。更多函数签名信息，请参见函数签名及数据类型。
- 问题现象二：**运行报错描述为** `TypeError: expected <class 'xxx'> but <class 'yyy'> found, value:zzz` 。
  - 产生原因：函数签名指定的返回值类型与MaxCompute UDF代码实际返回的数据类型不一致。
  - 解决措施：确认期望返回结果，修改函数签名或MaxCompute UDF代码，确保二者数据类型一致。
- 问题现象三：**运行报错描述为** `Semantic analysis exception - evaluate function in class xxx.yyy for user defined function zz does not match annotation ***->***` 。
  - 产生原因：函数签名中指定的入参个数与MaxCompute UDF代码中对应方法的入参个数不一致。
  - 解决措施：确认实际入参个数，修改函数签名或MaxCompute UDF代码，确保二者入参个数一致。

第三方包问题

- 问题现象：**运行报错描述为** `GLIBCXX_x.x.x not found` 。
- 产生原因：报错的so链接库文件依赖的GLIBCXX版本高于MaxCompute本身支持的版本。GLIBC、CXXABI同理。
- 解决措施：使用兼容的WHEEL包或在兼容的环境中重新编译so链接库文件。MaxCompute支持的二进制可执行文件或so链接库文件依赖的最大版本如下。

  ```
  GLIBC <= 2.17
  CXXABI <= 1.3.8
  GLIBCXX <= 3.4.19
  GCC <= 4.2.0
  ```

UDTF相关问题

- 问题现象：**运行报错描述为** `Semantic analysis exception - expect 2 aliases but have 0` 。
- 产生原因：Python UDTF代码中没有指定输出列名。
- 解决措施：您可以在调用Python UDTF的SELECT语句中通过 `as` 子句给出列名。命令示例如下。

  ```
  select my_udtf(col0, col1) as (ret_col0, ret_col1, ret_col2) from tmp1;
  ```

UDAF相关问题

- 问题现象一：**运行报错描述为** `Script exception - ValueError: unmarshallable object` 。
  - 产生原因：Python UDAF代码中的 `buffer` 不是Marshal对象。
  - 解决措施：为 `buffer` 赋值时，需要确保值为Marshal对象。假设Python UDAF中要使用两个 `buffer` ，类型分别为LIST和DICT，则 `new_buffer` 方法中应该写为 `return [list(), dict()]` 。在 `iterate/merge/terminate` 方法中使用 `buffer/pbuffer` 时，LIST类型的 `buffer` 对应 `buffer[0]/pbuffer[0]` ，DICT类型的 `buffer` 对应 `buffer[1]/pbuffer[1]` 。如果 `buffer` 的元素为LIST或DICT，这些元素也必须是Marshal对象。
- 问题现象二：**运行报错描述为** `Python UDAF buffer size overflowed: 2821486749` 。
  - 产生原因：Python UDAF中的 `buffer` 经过 `Marshal` 处理后的大小超过2 GB，用户使用 `buffer` 的方式有误， `buffer` 的大小不应该随数据量递增。
  - 解决措施：重新设计Python UDAF的逻辑， `buffer` 的大小不应该随数据量递增。例如声明了一个 `buffer` 是 `list` ， `iterate` 和 `merge` 阶段不能一直往 `buffer` 里增加数据。更多Python UDAF信息，请参见UDAF概述。

3.11. MaxCompute UDT

3.11.1. UDT概述

MaxCompute基于新一代的SQL引擎推出新功能UDT（User Defined Type）。MaxCompute的UDT功能允许您在SQL中直接调用第三方语言的类使用其方法、或直接使用第三方对象获取其数据内容。

UDT介绍

很多SQL引擎中UDT与MaxCompute的复杂类型STRUCT类似，相比之下，MaxCompute中的UDT与Create Type的概念更类似，Type中包含数据域和方法。MaxCompute不需要用特殊的DDL语法来定义新类型，通过UDT可以在SQL中直接使用新类型。通过如下示例，为您直观地介绍UDT的功能。

例如，在SQL语句中调用Java的*java.lang*包。您可以使用以下两种方法：

- 通过UDT功能在SQL语句中直接调用*java.lang*。

  ```
  --打开新类型，因为下面的操作会用到INTEGER，即INT类型。
  set odps.sql.type.system.odps2=true;
  SELECT java.lang.Integer.MAX_VALUE;
  ```

  和Java语言一样，*java.lang*包可以省略，所以上述示例可以简写为如下语句。

  ```
  set odps.sql.type.system.odps2=true;
  SELECT Integer.MAX_VALUE;
  ```

  输出结果如下。

  ```
  +------------+
  | max_value |
  +------------+
  | 2147483647 |
  +------------+
  ```

- 使用UDF在SQL语句中调用*java.lang*。

  i. 定义一个UDF的类。

  ```
  package com.aliyun.odps.test;
  public class IntegerMaxValue extends com.aliyun.odps.udf.UDF {
    public Integer evaluate() {
       return Integer.MAX_VALUE;
    }
  }
  ```

  ii. 将上面的UDF编译，并打成JAR包，然后上传JAR包，并创建Function。

  ```
  add jar odps-test.jar;
  create function integer_max_value as 'com.aliyun.odps.test.IntegerMaxValue' using 'odps-test.jar';
  ```

  iii. 在SQL中使用。

  ```
  select integer_max_value();
  ```

  由上例可以看出，UDT简化了上述一系列的过程，方便您使用其它语言扩展SQL的功能。

应用场景

UDT的常用场景如下：

- MaxCompute没有提供，但可以使用其它语言简单实现的功能。

  例如，只需调用一次Java内置类的方法即可实现，但MaxCompute却没有提供简单的方法实现的功能。如果使用UDF实现，整个过程会过于繁杂。

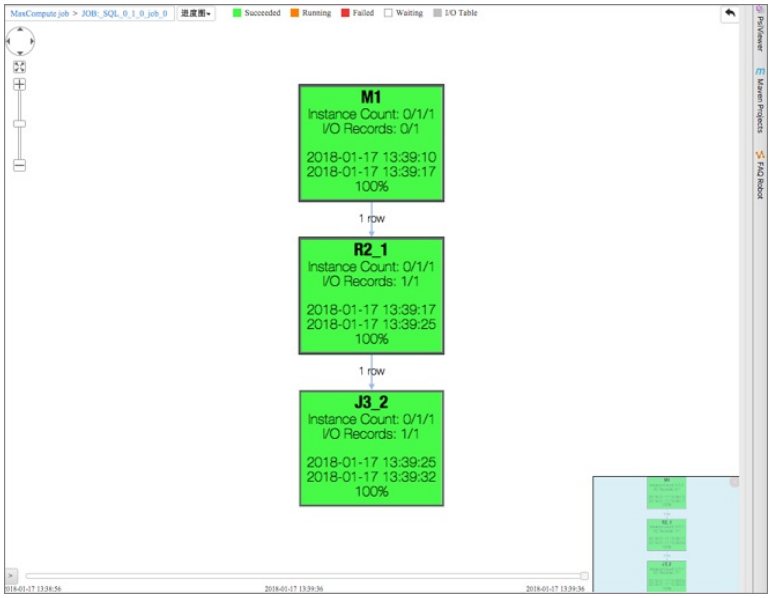
- SQL中需要调用第三方库实现相关功能。希望能够在SQL中直接调用，而不需要再Wrap一层UDF。
- SQL中需要直接调用第三方语言源代码。Select Transform支持把脚本写到SQL语句中，提升可读性和代码易维护性。但是某些语言无法这样使用，例如Java源代码必须经过编译才能执行，通过UDT功能将这些语言也可以直接写入SQL中。

实现原理

通过以下示例为您介绍UDT的执行过程。

```
--示例数据。
@table1 := select * from values ('10000000000000000000') as t(x);
@table2 := select * from values (100L) as t(y);
--代码逻辑。
--new创建对象。
@a := select new java.math.BigInteger(x) x from @table1;
--静态方法调用。
@b := select java.math.BigInteger.valueOf(y) y from @table2;
--实例方法调用。
select /*+mapjoin(b)*/ x.add(y).toString() from @a a join @b b;
--输出结果如下所示。
100000000000000000100
```

整个示例的运行过程如下图所示。



该UDT共有三个Stage：M1、R2和J3。如果您熟悉MapReduce原理即可知道，由于 `Join` 的存在需要做数据Reshuffle，所以会出现多个Stage。通常，不同的Stage是在不同的进程、不同的物理机器上运行的。

M1只执行 `new java.math.BigInteger(x)` 操作。

J3在不同阶段执行了 `java.math.BigInteger.valueOf(y)` 和 `x.add(y).toString()` 操作。这几个操作不仅分阶段执行，而且在不同的进程、不同的物理机器上执行。UDT把这个过程封装起来，将这个过程变得看起来和在同一个JVM中执行的效果几乎一样。

从上述示例中，您可以看到子查询的结果允许UDT类型的列。例如上面变量a的x列是 `java.math.BigInteger` 类型，而不是内置类型。UDT类型的数据可以被带到下一个Operator中，再调用其它方法，甚至可以参与数据Shuffle。

功能说明

- UDT仅支持Java语言，Java SDK的类都是默认可用的。

  > **说明** Runtime使用的JDK版本是JDK1.8，可能不支持更新版本的JDK。

- UDT支持您上传自己的JAR包，并且直接引用。当前提供了一些Flag方便您的使用。
  - `set odps.sql.session.resources` 指定引用的资源，可以指定多个，用逗号隔开，例如 `set odps.sql.session.resources=foo.sh,bar.txt;`。

    > **说明** 这个Flag和 `select transform` 中指定资源的Flag相同，所以这个Flag会同时影响两个功能。例如UDT概述中UDF的JAR包，用UDT使用。

    ```
    set odps.sql.type.system.odps2=true;
    set odps.sql.session.resources=odps-test.jar;
    --指定要引用的JAR。这些JAR需要提前上传至Project，并且需要是JAR类型的资源。
    select new com.aliyun.odps.test.IntegerMaxValue().evaluate();
    ```

- `odps.sql.session.java.imports` 指定默认的Java Package，可以指定多个，用逗号隔开。和Java的 `import` 语句类似，可以提供完整类路径，例如 `java.math.BigInteger` ，也可以使用 `*` 。暂不支持 `static import` 。

  UDT概述中UDF的JAR包，使用UDT功能还有如下写法。

  ```
  set odps.sql.type.system.odps2=true;
  set odps.sql.session.resources=odps-test.jar;
  set odps.sql.session.java.imports=com.aliyun.odps.test.*;
  -- 指定默认的Package。
  select new IntegerMaxValue().evaluate();
  ```

- UDT支持资源（Resource）的访问，您可以在SQL中通过 `com.aliyun.odps.udf.impl.UDTExecutionContext.get()` 静态方法获取 `ExecutionContext` 对象，从而访问当前的 `ExecutionContext` ，进而访问资源（例如文件资源和表格资源）。
- UDT支持的操作：
  - 实例化对象的 `new` 操作。
  - 实例化数组的 `new` 操作，包括使用初始化列表创建数组，例如 `new Integer[] { 1, 2, 3 }` 。
  - 方法调用，包括静态方法调用。
  - 域访问，包括静态域。

  > **说明**
  > - 仅支持公有方法和公有域的访问。
  > - UDT中的标识符是大小写敏感的，包括Package、类名、方法名和域名。
  > - 暂不支持匿名类和Lambda表达式。
  > - 暂不支持无返回值的函数调用（因为UDT均出现在Expression中，没有返回值的函数调用无法嵌入到Expression中，这个问题在后续的版本中会有解决方案）。

- UDT支持的数据类型：
  - UDT支持类型转换，支持SQL的风格，例如 `cast(1 as java.lang.Object)` 。但不支持Java风格的类型转换，例如 `(Object)1` 。
  - UDT内置类型与特定Java类型有一一映射关系，详情请参见Java UDF中的数据类型映射表，这个映射关系对UDT也有效。
    - 内置类型的数据能够直接调用其映射到的Java类型的方法，例如 `'123'.length()` ，`1L.hashCode()` 。
    - UDT类型能够直接参与内置函数或者UDF的运算，例如 `chr(Long.valueOf('100'))` ，其中 `Long.valueOf` 返回的是 `java.lang.Long` 类型的数据，而内置函数 `Chr` 接受的数据类型是内置类型BIGINT。
    - Java的PRIMITIVE类型可以自动转化为其BOXING类型，并应用前两条规则。

    > **说明** 部分内置的新数据类型需要先设置 `set odps.sql.type.system.odps2=true;` 才可使用，否则会报错。

  - UDT扩展了类型转换规则：
    - UDT对象可以被隐式类型转换为其基类对象。
    - UDT对象可以被强制类型转换为其基类或子类对象。
    - 没有继承关系的两个对象之间遵守原来的类型转换规则，注意这时可能会导致内容的变化。例如 `java.lang.Long` 类型的数据是可以强制转换为 `java.lang.Integer` 的，应用的是内置类型的BIGINT强制转换为INT的过程，而这个过程会导致数据内容的变化，甚至可能导致精度的损失。

    > **说明** 目前除隐式类型转换变成内置类型外，UDT对象不能存储到硬盘，即不能将UDT对象 `INSERT` 到表中（实际上DDL不支持UDT，不能创建这样的表）。内置类型支持BINARY，即支持自己实现序列化的过程，将byte[]的数据存储到硬盘，下次读出时再还原回来。因此需要您自己调用序列化反序列化方法，变成BINARY数据类型再存储到硬盘。

    屏显的最终结果不可以是UDT类型。对于屏显的场景，由于所有的Java类都有 `toString()` 方法，而 `java.lang.String` 类型是合法的。所以Debug时，可以用这种方法观察UDT的内容。

    您也可以设置 `set odps.sql.udt.display.tostring=true;` ，MaxCompute会自动帮您把所有以UDT为最终输出的列Wrap上 `java.util.Objects.toString(...)` ，以方便调试。这个Flag只对屏显语句生效，对 `INSERT` 语句不生效，所以它只在调试的过程中使用。
  - UDT支持比较完整的泛型。例如 `java.util.Arrays.asList(new java.math.BigInteger('1'))` ，编译器可以根据参数类型判断出该方法的返回值是 `java.util.List<java.math.BigInteger>` 类型。

    > **说明** 构造函数需要指定类型参数，否则需要使用 `java.lang.Object` ，这点和Java保持一致。
    >
    > `new java.util.ArrayList(java.util.Arrays.asList('1', '2'))` 的结果是 `java.util.ArrayList<Object>` 类型，而 `new java.util.ArrayList<String>(java.util.Arrays.asList('1', '2'))` 的结果是 `java.util.ArrayList<String>` 类型。

- 所有的运算符都是MaxCompute SQL的语义，不是UDT的语义。例如：
  - STRING的相加操作： `String.valueOf(1) + String.valueOf(2)` 的结果是3（STRING隐式转换为DOUBLE，并且DOUBLE相加），而不是12（Java中STRING相加是Concatenate的语义）。

- `=` 操作：SQL中的 `=` 不是赋值而是判断相等。而对于Java对象来说，判断相等应该用Equals方法，而非 `=` 操作。

- UDT对同一对象的概念是模糊的，这是由数据的Reshuffle导致的。对象有可能会在不同进程、不同物理机器之间传输。在传输过程中，同一个对象可能分别引用了不同的对象（例如对象先被Shuffle到两台机器，然后下次又Shuffle回一起）。所以在使用UDT时，应该使用 `equals` 方法判断相等，避免使用 `=` 判断相等。

  某行某列的对象，其内部包含的各个数据对象的相关性是可以保证的。不同行或者不同列的对象的数据相关性是不保证的。

- UDT不能用作Shuffle Key，包括 `Join` 、 `Group By` 、 `Distribute By` 、 `Sort By` 、 `Order By` 、 `Cluster By` 等结构的Key。

  UDT可以在Expression中间的任意阶段使用，但不能作为最终输出。例如，不可以使用语句 `group by new java.math.BigInteger('123')` ，但可以使用语句 `group by new java.math.BigInteger('123').hashCode()` 。因为 `hashCode` 方法的返回值是 `int.class` 类型，可以当做内置类型INT来使用（应用上述内置类型与特定Java类型规则）。

- UDT不仅可以实现Scalar函数的功能，配合内置函数COLLECT_SET和其他函数，UDT还可以实现Aggregator和Table Function功能。

功能优势

UDT在功能优势：

- 使用简单，无需定义任何Function。
- 支持JDK的所有功能，扩展了SQL的能力。
- 代码可与SQL放于同一文件，便于管理。
- 可直接使用其它类库，代码重用率高。
- 可以使用面向对象的思想设计某些功能。

后续待完善功能：

- 支持无返回值的函数调用，或支持（有返回值但忽略返回值）直接取操作数本身的函数调用。例如，调用List的 `add` 方法会返回执行完 `add` 操作的List。
- 支持匿名类和Lambda表达式。
- 支持用作Shuffle Key。
- 支持Java外的其他语言，例如Python。

性能

因为UDT和UDF的执行过程非常接近，所以UDT与UDF的性能几乎一致。优化后的计算引擎使得UDT在特定场景下的性能更高。

- UDT对象只有在跨进程时才需要做序列化和反序列化，因此在执行不需要数据Reshuffle的操作（如 `JOIN` 或 `AGGREGATE` ）时，UDT可节省序列化和反序列化的开销。
- 因为UDT的Runtime基于Codegen而非反射实现的，所以不存在反射带来的性能损失。在您使用过程中，连续多个UDT操作会合并在一个FunctionCall里一起执行。例如在之前的示例中， `values[x].add(values[y]).divide(java.math.BigInteger.valueOf(2))` 实际上只会调用一次UDT。所以，UDT操作的单元虽然较小，却并不会因多次函数调用而造成接口额外开销。

安全性

在安全控制方面，UDT和UDF完全一样，都会受到Java沙箱Policy的限制。因此如果要使用受限的操作，需要打开沙箱隔离，或者申请沙箱白名单。

3.11.2. UDT示例

本文为您提供常用的UDT示例。例如Java数组、JSON、复杂类型、聚合操作、表值函数、函数重载和引用嵌入式代码。

> 说明 请您在脚本模式下运行如下示例代码，脚本模式详情请参见脚本模式SQL。

Java数组

```
set odps.sql.type.system.odps2=true;
set odps.sql.udt.display.tostring=true;
SELECT
    new Integer[10],     --创建一个包含10个元素的数组。
    new Integer[] {c1, c2, c3},  --通过初始化列表创建一个长度为3的数组。
    new Integer[][] { new Integer[] {c1, c2}, new Integer[] {c3, c4} },  --创建多维数组。
    new Integer[] {c1, c2, c3} [2], --通过下标操作访问数组元素。
    java.util.Arrays.asList(c1, c2, c3);     --创建一个List<Integer>，也能当做array<int>来用，是另一种创建内置ARRAY数据的方法。
FROM VALUES (1,2,3,4) AS t(c1, c2, c3, c4);
```

JSON

UDT运行时自带一个JSON的依赖（2.2.4），可以直接使用JSON。

> 说明 除JSON外，MaxCompute运行时自带的依赖还包括 `commons-logging (1.1.1)` 、 `commons-lang (2.5)` 、 `commons-io (2.4)` 和 `protobuf-java (2.4.1)` 。

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*,java,com.google.gson.*; --同时导入多个Package时用逗号隔开。
@a := select new Gson() gson;   --构建Gson对象。
select
gson.toJson(new ArrayList<Integer>(Arrays.asList(1, 2, 3))), --将任意对象转成JSON字符串。
cast(gson.fromJson('["a","b","c"]', List.class) as List<String>) --反序列化JSON字符串，注意Gson的接口，直接反序列化后是List<Object>类型，这里强制转换成List<String>，方便后续使用。
from @a;
```

相比于内建函数字符串函数，该方法不仅使用方便，还会在提取JSON字符串内容时，将JSON字符串反序列化为格式化数据，提升工作效率。

复杂类型

内置类型ARRAY与 `java.util.List` 、MAP和 `java.util.Map` 存在映射关系。

- Java中实现 `java.util.List` 或 `java.util.Map` 接口类的对象，都可参与MaxCompute SQL的复杂类型操作。
- MaxCompute中ARRAY或MAP的数据，能够直接调用List或者MAP的接口。

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*;
select
    size(new ArrayList<Integer>()),        --对ArrayList数据调用内置函数Size。
    array(1,2,3).size(),                   --对内置类型ARRAY调用List的Size方法。
    sort_array(new ArrayList<Integer>()),  --对ArrayList的数据进行排序。
    al[1],                                 --虽然Java的List不支持下标操作，但ARRAY支持。
    Objects.toString(a),        --之前不支持将ARRAY类型Cast成STRING，现在有绕过方法了。
    array(1,2,3).subList(1, 2)             --计算subList。
from (select new ArrayList<Integer>(array(1,2,3)) as al, array(1,2,3) as a) t;
```

聚合操作

UDT实现聚合的原理是，先用内建函数COLLECT_SET或COLLECT_LIST将数据转变成List，之后对该List应用UDT的标量方法计算数据的聚合值。

示例如下，计算BigInteger的中位数（由于数据是 `java.math.BigInteger` 类型的，所以不能直接用内建函数MEDIAN）。

```
set odps.sql.session.java.imports=java.math.*;
@test_data := select * from values (1),(2),(3),(5) as t(value);
@a := select collect_list(new BigInteger(value)) values from @test_data;  --先把数据聚合成List。
@b := select sort_array(values) as values, values.size() cnt from @a;  --计算中位数的逻辑，先将数据排序。
@c := select if(cnt % 2 == 1, new BigDecimal(values[cnt div 2]), new BigDecimal(values[cnt div 2 - 1].add(values[cnt div 2])).div
ide(new BigDecimal(2))) med from @b;
--最终结果。
select med.toString() from @c;
```

`collect_list` 会先将所有数据都收集到一起，无法实现Partial Aggregate，处理效率比内置聚合函数或者UDAF低，请尽量使用内置聚合函数。同时，把一个Group的所有数据都收集到一起，会增加数据倾斜的风险。

如果UDAF的逻辑是要将所有数据收集到一起（例如类似内置聚合函数 `WM_CONCAT` 的功能），使用上述方法，处理效率比UDAF高。

表值函数

表值函数允许输入多行多列数据，输出多行多列数据。可以按照如下操作实现：

- 输入多行多列数据，详情请参见聚合操作。
- 输出多行多列数据，可以用UDT方法输出一个Collection类型的数据（List或者MAP），然后调用Explode函数，将Collections展开成多行。
- UDT可以包含多个数据域，通过调用不同的Getter方法获取各个域的内容即可展开成多列。

展开一个JSON字符串的内容，示例如下。

```
@a := select '[{"a":"1","b":"2"},{"a":"1","b":"2"}]' str; --示例数据。
@b := select new com.google.gson.Gson().fromJson(str, java.util.List.class) l from @a; --反序列化JSON。
@c := select cast(e as java.util.Map<Object,Object>) m from @b lateral view explode(l) t as e;  --用explode展开成多行。
@d := select m.get('a') as a, m.get('b') as b from @c; --展开成多列。
select a.toString() a, b.toString() b from @d; --最终结果输出（注意变量d的输出中a、b两列是Object类型）。
```

函数重载

MaxCompute UDF使用 `evaluate` 方法重载函数。该方式不支持泛型，当您需要定义一个支持任何数据类型的函数时，必须为每种类型都写一个 `evaluate` 函数。该方法无法实现个别输入类型（例如ARRAY）的重载函数。在没有提供Resolve注解的情况下，Python UDF或UDTF会根据参数个数决定输入参数，同时支持变长参数，但会导致编译器无法静态找到某些错误。

通过UDT实现函数重载，可以解决上述问题。UDT支持泛型、类继承和变长参数，为您提供灵活的函数定义方式，示例如下。

```
public class UDTClass {
    // 这个函数支持一个数值类型（可以是TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE以及任何以Number为基类的UDT），返回DOUBLE。
    public static Double doubleValue(Number input) {
        return input.doubleValue();
    }
    // 这个方法支持一个数值类型参数和一个任意类型的参数，返回值类型与第二个参数的类型相同。
    public static <T extends Number, R> R nullOrValue(T a, R b) {
        return a.doubleValue() > 0 ? b : null;
    }
    // 这个方法支持一个任意元素类型的ARRAY或List，返回BIGINT。
    public static Long length(java.util.List<? extends Object> input) {
        return input.size();
    }
    // 注意在不做强制转换的情况下参数只支持UDT的java.util.Map<Object, Object>对象。如果需要传入任何MAP对象，例如map<bigint,bigint>可以考虑
:
    // 1. 定义函数时使用java.util.Map<? extends Object, ? extends Object>。
    // 2. 调用时做强制转换，例如UDTClass.mapSize(cast(mapObj as java.util.Map<Object, Object>))。
    public static Long mapSize(java.util.Map<Object, Object> input) {
        return input.size();
    }
}
```

特定场景下，UDF可以通过 `com.aliyun.odps.udf.ExecutionContext` （在 `setup` 方法中传入）获取上下文；UDT可以通过 `com.aliyun.odps.udt.UDTExecutionContext.get()` 方法获取 `ExecutionContext` 对象。

UDT引用嵌入式代码

代码嵌入式UDF允许您将SQL脚本和第三方代码放入同一个源码文件，减少使用UDT的操作步骤，方便日常开发。详情请参见代码嵌入式UDF。

3.12. MaxCompute UDJ

基于MaxCompute 2.0计算引擎，MaxCompute在UDF框架中引入了新的扩展机制UDJ（User Defined Join），来实现灵活的跨表、多表自定义操作，同时减少通过MapReduce等方式对分布式系统底层细节的操作。

背景信息

MaxCompute内置了多种Join操作，包括Inner/Right Join、Outer/Left Join、Outer/Full Join、Outer/semi/Anti-semi Join等。这些内置的Join操作功能强大，但由于其标准的Join实现，无法满足很多跨表操作的需求场景。

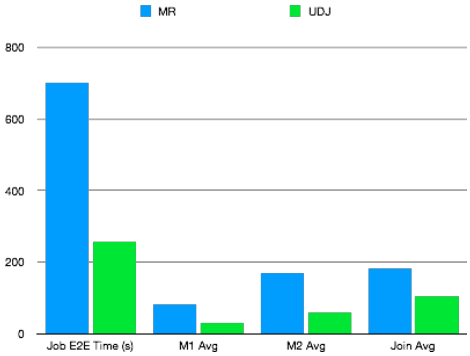
通常，您可以通过UDF（User Defined Function）描述代码框架，但现有的UDF/UDTF/UDAF接口主要是针对在单个数据表上的操作而设计。一旦涉及多表的自定义操作，经常还需要依赖于内置**Join +各种UDF/UDTF**，并配合比较复杂的SQL语句来完成。在多表操作的场景上，您不得不放弃SQL而使用自定义MapReduce，才能完成所需的计算。

无论是**Join +各种UDF/UDTF+复杂SQL**还是自定义MapReduce门槛都比较高，同时还会带来一些问题：

- 使用**Join +各种UDF/UDTF+复杂SQL**：多个复杂的Join和散布在SQL语言各处的代码揉合在一起，将带来多处的逻辑黑盒，不利于生成最优的执行计划。
- 使用MapReduce：不仅更大程度上限制了系统进行执行优化的可能性，而且在深度优化本地运行代码时，由于MapReduce绝大部分代码由Java完成，在执行效率上会远低于MaxCompute基于LLVM的代码生成器。

UDJ的性能

通过一个真实的线上MapReduce作业进行测试验证UDJ的性能，该MapReduce作业实现一套比较复杂的算法。将两个表合并在一起，用UDJ对该MapReduce进行改写，并且验证UDJ实现结果的正确性。在并发度相同的情况下，两者性能对比如下。



由上图可见，UDJ接口的引入，一方面让您能更方便地描述对多表数据进行操作的复杂逻辑，一方面大幅提升了性能。代码只在UDJ内被调用，其上下游的逻辑（例如这个例子中的整个Mapper逻辑）则完全通过MaxCompute高效的Native运行完成。在Java代码中，由于MaxCompute UDJ运行引擎和Java接口之间的数据交互逻辑有深度的优化，通过UDJ实现的Join逻辑也比其对等的Reducer更高效。

UDJ跨表Join功能

通过如下样例为您详细介绍MaxCompute UDJ跨表Join的使用方法。

假设存在两个日志表，分别是payment和user_client_log。

- payment：表中保存了用户的支付记录，一笔支付记录包含用户ID、支付时间和支付内容。样例数据如下。

user_id	time	pay_info
2656199	2018-02-13 22:30:00	gZhvdySOQb
8881237	2018-02-13 08:30:00	pYvotuLDIT
8881237	2018-02-13 10:32:00	KBuMzRpsko

- user_client_log：保存了用户的客户端日志，每一条日志包含了用户ID、日志时间和日志内容。样例数据如下。

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzlDyL
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzlDyL

对于每一条客户端日志，找出该用户在payment表里时间最接近的一条支付记录，将其中的支付内容和日志内容合并输出。达到如下效果。

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn, pay pYvotuLDIT
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzlDyL, pay pYvotuLDIT
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzlDyL, pay KBuMzRpsko

面对此类需求通常有如下2种解决方案：

- 使用内置Join。SQL伪代码如下。

```
SELECT
  p.user_id,
  p.time,
  merge(p.pay_info, u.content)
FROM
  payment p RIGHT OUTER JOIN user_client_log u
ON p.user_id = u.user_id and abs(p.time - u.time) = min(abs(p.time - u.time))
```

关联时需要知道相同user_id下的p.time与u.time差异最小的值，且聚合函数不能出现在关联条件上。因此，这个看似简单的需求，无法通过标准的关联操作实现。

- 使用UDJ方法实现。
  i. 注册UDJ函数。
    a. 配置新版本的SDK。

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.29.10-public</version>
  <scope>provided</scope>
</dependency>
```

    b. 编写UDJ代码，并将代码打包为odps-udj-example.jar。

```
package com.aliyun.odps.udf.example.udj;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.Yieldable;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDJ;
import com.aliyun.odps.udf.annotation.Resolve;
import java.util.ArrayList;
import java.util.Iterator;
/** 对于右表的每个记录，找到最近的左表记录
 * 合并两条记录。
 */
```

```
@Resolve("->string,bigint,string")
public class PayUserLogMergeJoin extends UDJ {
  private Record outputRecord;
  /** 将在数据处理阶段之前调用。 用户可以实施这个方法做初始化工作。
   */
  @Override
  public void setup(ExecutionContext executionContext, DataAttributes dataAttributes) {
    //
    outputRecord = new ArrayRecord(new Column[]{
      new Column("user_id", OdpsType.STRING),
      new Column("time", OdpsType.BIGINT),
      new Column("content", OdpsType.STRING)
    });
  }
  /** 重写此方法以实现连接逻辑。
   * @param key 当前连接键。
   * @param 左表对应当前键的记录组。
   * @param right对应当前键的右表记录组。
   * @param output用于输出UDJ的结果。
   */
  @Override
  public void join(Record key, Iterator<Record> left, Iterator<Record> right, Yieldable<Record> output) {
    outputRecord.setString(0, key.getString(0));
    if (!right.hasNext()) {
      // 空右组，什么都不做。
      return;
    } else if (!left.hasNext()) {
      // 空左组。 输出右侧组的所有记录而不合并。
      while (right.hasNext()) {
        Record logRecord = right.next();
        outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
        outputRecord.setString(2, logRecord.getString(1));
        output.yield(outputRecord);
      }
      return;
    }
    ArrayList<Record> pays = new ArrayList<>();
    // 左侧记录组将从开始到结束迭代。
    // 对于右组的每个记录，迭代器无法重置。
    // 所以我们将左边的每个记录保存到ArrayList。
    left.forEachRemaining(pay -> pays.add(pay.clone()));
    while (right.hasNext()) {
      Record log = right.next();
      long logTime = log.getDatetime(0).getTime();
      long minDelta = Long.MAX_VALUE;
      Record nearestPay = null;
      // 迭代左边的所有记录，找到有的记录时间上的微小差异。
      for (Record pay: pays) {
        long delta = Math.abs(logTime - pay.getDatetime(0).getTime());
        if (delta < minDelta) {
          minDelta = delta;
          nearestPay = pay;
        }
      }
      // 将日志记录与最近的支付记录合并，并输出到结果。
      outputRecord.setBigint(1, log.getDatetime(0).getTime());
      outputRecord.setString(2, mergeLog(nearestPay.getString(1), log.getString(1)));
      output.yield(outputRecord);
    }
  }
  String mergeLog(String payInfo, String logContent) {
    return logContent + ", pay " + payInfo;
  }
  @Override
  public void close() {
  }
}
```

c. 在MaxCompute中添加Jar包资源。

```
add jar odps-udj-example.jar;
```

d. 在MaxCompute中注册UDJ函数*pay_user_log_merge_join*。

```
create function pay_user_log_merge_join
  as 'com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin'
  using 'odps-udj-example.jar';
```

ii. 准备示例数据。

a. 创建示例表payment和user_client_log。

```
create table payment(user_id string,time datetime,pay_info string);
create table user_client_log(user_id string,time datetime,content string);
```

b. 为示例表中插入数据。

```
--向payment表中插入数据。
INSERT OVERWRITE TABLE payment VALUES
('1335656', datetime '2018-02-13 19:54:00', 'PEqMSHyktn'),
('2656199', datetime '2018-02-13 12:21:00', 'pYvotuLDIT'),
('2656199', datetime '2018-02-13 20:50:00', 'PEqMSHyktn'),
('2656199', datetime '2018-02-13 22:30:00', 'gZhvdySOQb'),
('8881237', datetime '2018-02-13 08:30:00', 'pYvotuLDIT'),
('8881237', datetime '2018-02-13 10:32:00', 'KBuMzRpsko'),
('9890100', datetime '2018-02-13 16:01:00', 'gZhvdySOQb'),
('9890100', datetime '2018-02-13 16:26:00', 'MxONdLckwa')
;
--向user_client_log表中插入数据。
INSERT OVERWRITE TABLE user_client_log VALUES
('1000235', datetime '2018-02-13 00:25:36', 'click FNOXAibRjkIaQPB'),
('1000235', datetime '2018-02-13 22:30:00', 'click GczrYaxvkiPultZ'),
('1335656', datetime '2018-02-13 18:30:00', 'click MxONdLckpAFUHRS'),
('1335656', datetime '2018-02-13 19:54:00', 'click mKRPGOciFDyzTgM'),
('2656199', datetime '2018-02-13 08:30:00', 'click CZwafHsbJOPNitL'),
('2656199', datetime '2018-02-13 09:14:00', 'click nYHJqIpjevkKToy'),
('2656199', datetime '2018-02-13 21:05:00', 'click gbAfPCwrGXvEjpI'),
('2656199', datetime '2018-02-13 21:08:00', 'click dhpZyWMuGjBOTJP'),
('2656199', datetime '2018-02-13 22:29:00', 'click bAsxnUdDhvfqaBr'),
('2656199', datetime '2018-02-13 22:30:00', 'click XIhZdLaOocQRmrY'),
('4356142', datetime '2018-02-13 18:30:00', 'click DYqShmGbIoWKier'),
('4356142', datetime '2018-02-13 19:54:00', 'click DYqShmGbIoWKier'),
('8881237', datetime '2018-02-13 00:30:00', 'click MpkvilgWSmhUuPn'),
('8881237', datetime '2018-02-13 06:14:00', 'click OkTYNUHMqZzlDyL'),
('8881237', datetime '2018-02-13 10:30:00', 'click OkTYNUHMqZzlDyL'),
('9890100', datetime '2018-02-13 16:01:00', 'click vOTQfBFjcgXisYU'),
('9890100', datetime '2018-02-13 16:20:00', 'click WxaLgOCcVEvhiFJ')
;
```

iii. 在SQL中使用UDJ函数。

```
SELECT r.user_id, from_unixtime(time/1000) as time, content FROM (
SELECT user_id, time as time, pay_info FROM payment
) p JOIN (
SELECT user_id, time as time, content FROM user_client_log
) u
ON p.user_id = u.user_id
USING pay_user_log_merge_join(p.time, p.pay_info, u.time, u.content)
r
AS (user_id, time, content);
```

USING子句中的参数说明：

- `pay_user_log_merge_join` 是注册的UDJ在SQL中的函数名。
- `(p.time, p.pay_info, u.time, u.content)` 是UDJ中用到的左右表的列。
- `r` 是UDJ结果的别名，方便其他地方引用UDJ的结果。
- `(user_id, time, content)` 是UDJ产生的结果的列名。

运行结果如下。

```
+---------+------------+---------+
| user_id | time       | content |
+---------+------------+---------+
| 1000235 | 2018-02-13 00:25:36 | click FNOXAibRjkIaQPB |
| 1000235 | 2018-02-13 22:30:00 | click GczrYaxvkiPultZ |
| 1335656 | 2018-02-13 18:30:00 | click MxONdLckpAFUHRS, pay PEqMSHyktn |
| 1335656 | 2018-02-13 19:54:00 | click mKRPGOciFDyzTgM, pay PEqMSHyktn |
| 2656199 | 2018-02-13 08:30:00 | click CZwafHsbJOPNitL, pay pYvotuLDIT |
| 2656199 | 2018-02-13 09:14:00 | click nYHJqIpjevkKToy, pay pYvotuLDIT |
| 2656199 | 2018-02-13 21:05:00 | click gbAfPCwrGXvEjpI, pay PEqMSHyktn |
| 2656199 | 2018-02-13 21:08:00 | click dhpZyWMuGjBOTJP, pay PEqMSHyktn |
| 2656199 | 2018-02-13 22:29:00 | click bAsxnUdDhvfqaBr, pay gZhvdySOQb |
| 2656199 | 2018-02-13 22:30:00 | click XIhZdLaOocQRmrY, pay gZhvdySOQb |
| 4356142 | 2018-02-13 18:30:00 | click DYqShmGbIoWKier |
| 4356142 | 2018-02-13 19:54:00 | click DYqShmGbIoWKier |
| 8881237 | 2018-02-13 00:30:00 | click MpkvilgWSmhUuPn, pay pYvotuLDIT |
| 8881237 | 2018-02-13 06:14:00 | click OkTYNUHMqZzlDyL, pay pYvotuLDIT |
| 8881237 | 2018-02-13 10:30:00 | click OkTYNUHMqZzlDyL, pay KBuMzRpsko |
| 9890100 | 2018-02-13 16:01:00 | click vOTQfBFjcgXisYU, pay gZhvdySOQb |
| 9890100 | 2018-02-13 16:20:00 | click WxaLgOCcVEvhiFJ, pay MxONdLckwa |
+---------+------------+---------+
```

UDJ预排序功能

为了找到payment值相差最小的一条记录，需要反复对payment表的数据进行iterator遍历。所以事先将相同user_id的payment记录全部加载到了ArrayList，当同一个用户一天之内的支付行为比较少时，这个方法可用。在其它场景中，有时同组内的数据量可能非常大，大到无法在内存中存放，此时就需要通过其他方式解决。此时，您可以通过SORT BY预排序解决。

当某个用户的支付数据量非常大，导致无法将payment放在内存中时，如果组内所有数据如果已经按照时间排序，则只需要比较两边iterator最顶部的数据，就可以实现这个功能。

该方式主要是使用SORT BY子句对UDJ的数据进行预排序。在这个过程中，最多只需要同时缓存3条记录，就可以实现和之前算法的相同的功能。Java UDJ代码如下。

```
@Override
public void join(Record key, Iterator<Record> left, Iterator<Record> right, Yieldable<Record> output) {
  outputRecord.setString(0, key.getString(0));
  if (!right.hasNext()) {
    return;
  } else if (!left.hasNext()) {
    while (right.hasNext()) {
      Record logRecord = right.next();
      outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
      outputRecord.setString(2, logRecord.getString(1));
      output.yield(outputRecord);
    }
    return;
  }
  long prevDelta = Long.MAX_VALUE;
  Record logRecord = right.next();
  Record payRecord = left.next();
  Record lastPayRecord = payRecord.clone();
  while (true) {
    long delta = logRecord.getDatetime(0).getTime() - payRecord.getDatetime(0).getTime();
    if (left.hasNext() && delta > 0) {
      //两个记录之间的时间差正在减少，我们仍然可以操作。
      //探索左侧组以尝试获得更小的增量。
      lastPayRecord = payRecord.clone();
      prevDelta = delta;
      payRecord = left.next();
    } else {
     //到达最小delta点。 检查最后的记录，
     //输出合并结果并准备处理下一条记录。
     //右组。
      Record nearestPay = Math.abs(delta) < prevDelta ? payRecord : lastPayRecord;
      outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
      String mergedString = mergeLog(nearestPay.getString(1), logRecord.getString(1));
      outputRecord.setString(2, mergedString);
      output.yield(outputRecord);
      if (right.hasNext()) {
        logRecord = right.next();
        prevDelta = Math.abs(
          logRecord.getDatetime(0).getTime() - lastPayRecord.getDatetime(0).getTime()
        );
      } else {
        break;
      }
    }
  }
}
```

3.13. 脚本模式SQL

MaxCompute当前的SQL引擎支持脚本模式SQL（Script Mode SQL）。在脚本模式下编译脚本时，一个多语句的SQL脚本文件将被作为一个整体进行编译，无需对单个语句进行编译。在提交运行时，SQL脚本文件会被整体提交，并生成一个执行计划，保证只需排队一次、执行一次，让您能充分利用MaxCompute的资源。

> **说明** 脚本模式SQL无法使用计量预估完成费用预估，具体费用请以实际费用账单为准，详情请参见查看账单详情。

Script Mode的SQL语句书写便利，您只需要按照业务逻辑，用类似于普通编程语言的方式书写，无需考虑如何组织语句。

语法结构

```
--set
set odps.sql.type.system.odps2=true;
[set odps.stage.reducer.num=xxx;]
[...]
--ddl
create table table1 xxx;
[create table table2 xxx;]
[...]
--dml
@var1 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM table3
        [WHERE where_condition];
@var2 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM table4
        [WHERE where_condition];
@var3 := SELECT [ALL | DISTINCT] var1.select_expr, var2.select_expr, ...
        FROM @var1 join @var2 on ...;
INSERT OVERWRITE|INTO TABLE [PARTITION (partcol1=val1, partcol2=val2 ...)]
        SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM @var3;
[@var4 := SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr, ... FROM @var1
        UNION ALL | UNION
        SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr, ... FROM @var2;
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
        AS
        SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM @var4;]
[...]
```

语法说明

- 脚本模式支持SET语句、部分DDL语句（不支持结果为屏显类型的语句如DESC、SHOW）、DML语句。
- 一个脚本的完整形式是SET、DDL、DML语句按先后顺序排列。每种语句都可以包含0到多个具体的SQL语句，且不同类型的语句不能交错。
- 使用 `@` 声明变量。
- 一个脚本最多支持一个屏显结果的语句（如单独的SELECT语句），否则会发生报错。不建议在脚本中执行屏显的SELECT语句。
- 一个脚本最多支持一个CREATE TABLE AS语句并且必须是最后一句。推荐您将建表语句与INSERT语句分开写。
- 脚本模式下，一旦有一个语句失败，整个脚本的语句都无法执行成功。
- 脚本模式下，只有当所有输入的数据都准备好并插入成功，才会生成一个作业进行数据处理。
- 脚本模式下，如果一个表先被写再被读，则会发生报错，如下。

  ```
  INSERT OVERWRITE table src2 SELECT * FROM src WHERE key > 0;
  @a := SELECT * FROM src2;
  SELECT * FROM @a;
  ```

  所以，为避免因表的先写后读产生的报错，应修改SQL脚本如下。

  ```
  @a := SELECT * FROM src WHERE key > 0;
  INSERT OVERWRITE table src2 SELECT * FROM @a;
  SELECT * FROM @a;
  ```

- 脚本模式的适用场景如下：
  - 脚本模式适合用来改写本来要用层层嵌套子查询的单个语句，或者因为脚本复杂性而不得不拆成多个语句的脚本。
  - 如果多个输入的数据源数据准备完成的时间间隔很长（例如一个01:00可以准备好，一个07:00可以准备好），则不适合通过table variable衔接拼装为一个大的脚本模式SQL。
  - 脚本模式下，您可以对一个变量赋常量值，然后执行 `SELECT * FROM 变量` 语句转化为标量与其它列进行计算。常量值也可以存放在一个单行的表中，命令示例如下。转化语法请参见子查询（SUBQUERY）。

    ```
    @a := SELECT 10; --对@a赋值常量10，或者赋值存在一个单行表t1中，SELECT col1 FROM t1。
    @b := SELECT key,value+(SELECT * FROM @a) FROM t2 WHERE key >10000; --t2表中value值与@a中的值进行计算。
    SELECT * FROM @b;
    ```

示例

脚本模式SQL示例如下。

```
CREATE TABLE IF NOT EXISTS dest(key STRING, value BIGINT) PARTITIONED BY (d STRING);
CREATE TABLE IF NOT EXISTS dest2(key STRING, value BIGINT) PARTITIONED  BY (d STRING);
@a := SELECT * FROM src WHERE value >0;
@b := SELECT * FROM src2 WHERE key is not null;
@c := SELECT * FROM src3 WHERE value is not null;
@d := SELECT a.key,b.value FROM @a LEFT OUTER JOIN @b ON a.key=b.key AND b.value>0;
@e := SELECT a.key,c.value FROM @a INNER JOIN @c ON a.key=c.key;
@f := SELECT * FROM @d UNION SELECT * FROM @e UNION SELECT * FROM @a;
INSERT OVERWRITE table dest PARTITION (d='20171111') SELECT * FROM @f;
@g := SELECT e.key,c.value FROM @e JOIN @c ON e.key=c.key;
INSERT OVERWRITE TABLE dest2 PARTITION (d='20171111') SELECT * FROM @g;
```

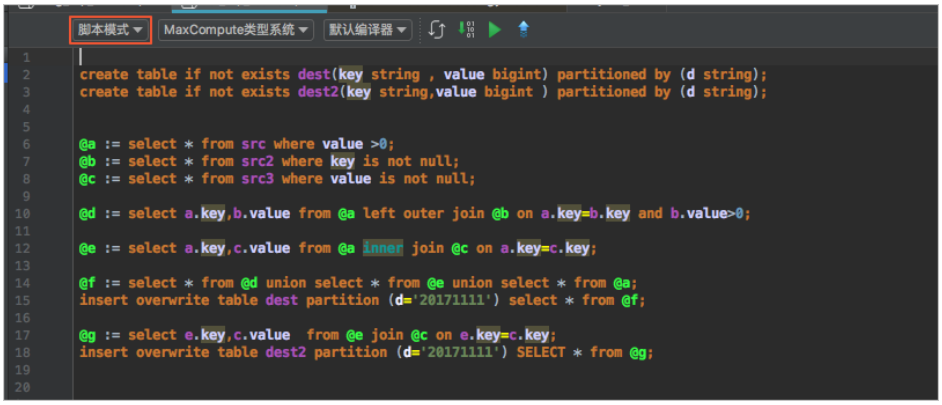
工具支持

- MaxCompute Studio，详情请参见认识MaxCompute Studio。
- MaxCompute客户端（odpscmd），详情请参见安装并配置MaxCompute客户端。
- DataWorks，详情请参见创建ODPS Script节点。

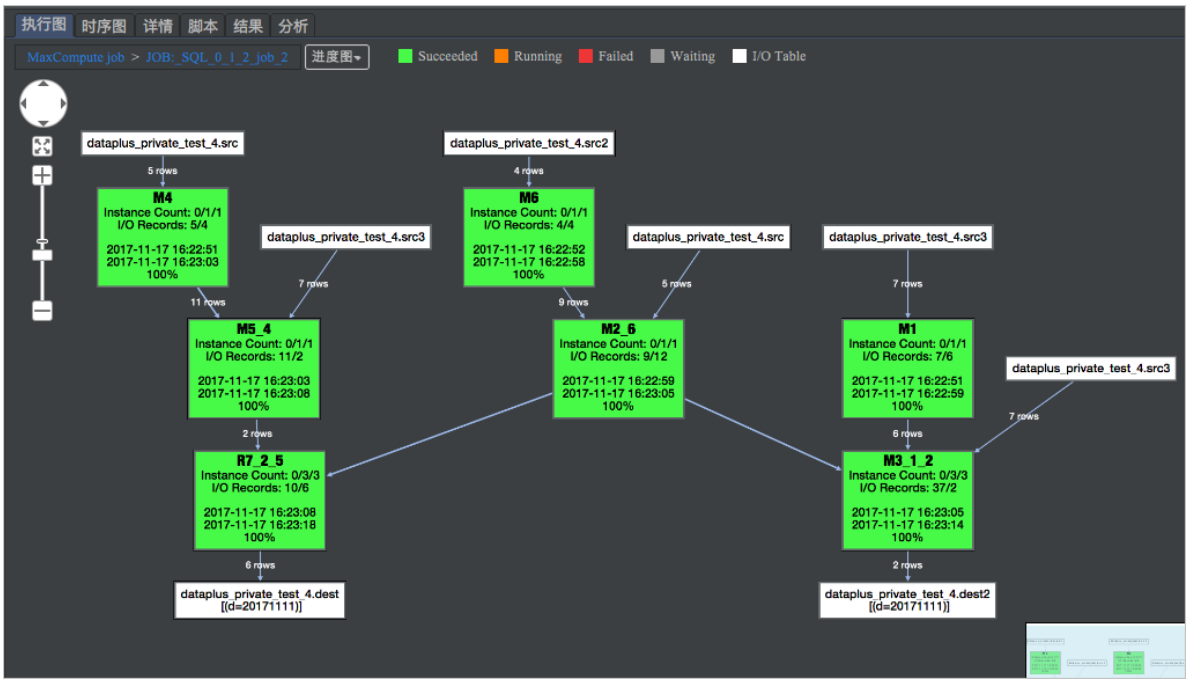
使用脚本模式

- **通过MaxCompute Studio使用脚本模式。**

  使用MaxCompute Studio脚本模式，首先请保证MaxCompute Studio完成安装、添加项目链接、建立MaxCompute SQL脚本文件，详情请参见安装Intellij IDEA、管理项目连接、创建MaxCompute Script Module。编辑脚本页面如下。



脚本编译后提交运行，查看执行计划图。虽然脚本上是多个语句，但执行计划图是同一个DAG图。



- **通过MaxCompute客户端（odpscmd）使用脚本模式。**

  您需要使用0.27以上版本的odpscmd提交脚本。建议您安装最新版本MaxCompute客户端安装包。安装后，请使用`-s`参数提交脚本。

编辑脚本模式的源码*myscript.sql*文件，在系统命令行窗口调用odpscmd执行如下命令。更多通过系统命令行窗口运行MaxCompute客户端的操作，请参见运行MaxCompute客户端。

```
..\bin>odpscmd -s myscript.sql
```

> 说明 `-s` 为odpscmd的命令行选项，类似于 `-f` 、 `-e` ，而非交互环境中的命令。odpscmd的交互环境中暂不支持脚本模式与表变量。

- **通过DataWorks使用脚本模式。**

  在DataWorks中可以建立脚本模式的节点ODPS Script，示例如下。



  在此节点中进行脚本模式编辑，编辑完成后单击工具栏的**运行**图标，提交脚本到MaxCompute执行。从输出信息的Logview URL中可以查看执行计划图和结果。

3.14. 渐进式计算

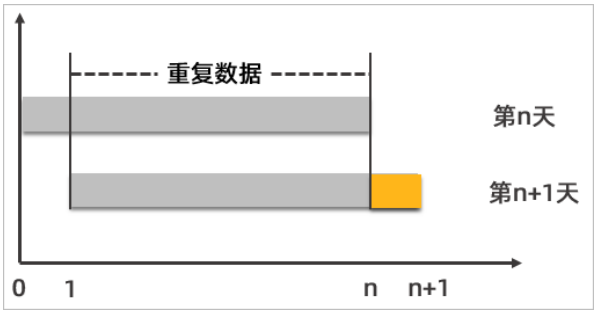
渐进式计算（Progressive Computing）是处于传统的流计算和批处理之间的一种计算方式，具有高资源利用率和低延迟的优势。本文为您介绍渐进式计算的基础配置、作业调优配置以及使用示例。

背景信息

渐进式计算能够处理增量数据并维护中间结果数据，提高了在计算资源消耗、执行时间调度、查询延迟、数据处理粒度等方面的灵活性。例如在范围查询（Range Query）场景下（即查询一段时间范围内的数据），使用传统批处理方式和渐进式计算方式每次统计过去一周的产品总销量，渐进式计算方式除首次执行外，其余每次执行的计算量减少了70%。如下所示：

- 传统批处理方式：

  第n（n>=7）天执行需要处理[n-6,n]区间的数据；第n+1天执行需要处理[n-5,n+1]区间的数据。可以看出，第n+1天执行过程中有6天的数据是被重复计算的。如下图所示：



- 渐进式计算方式：

  第n（n>=7）天执行需要处理[n-6,n]区间的数据；第n+1天执行需要处理[n,n+1]区间的数据。即第n+1天执行只需要处理当天的增量数据，再和之前6天计算好的数据整合后得到过去一周的结果。第n+1天执行时的数据不会被重复计算，因此计算量减少了70%。

> 说明
> - 由于范围查询场景在第一次使用渐进式计算时，需要计算出每一天的中间结果，因此部分作业的运行时间会比普通模式还要长，建议用户先手动在渐进式计算模式下运行一次，以免在基线时段第一次运行影响结果产出时间。
> - 由于渐进式计算会存储每天的中间结果，因此会占用额外的存储空间，中间结果会在不再会被使用的时候自动回收。额外空间开销成本和运行开销成本相比，总体上渐进式计算仍然会节省费用，运行速度更快。

渐进式计算基础配置

第一次开启渐进式计算时，建议先在开发环境进行测试，确保性能达到预期效果后再在生产环境启用。请您按如下步骤配置渐进式计算：

1. 在提交作业时配置渐进式计算。
   - 配置格式：
     ```
     set odps.progressive.enable=[true|false];
     ```
   - 参数说明：
     - true：表示使用渐进式计算方式处理提交的作业。
     - false：默认值为false。关闭渐进式计算方式。
2. 配置渐进式计算模式：
   - 配置格式：
     ```
     set odps.progressive.range.query.input.partition.pattern=<pattern_value>;
     ```
   - 参数说明：

     pattern_value的值以及含义如下：
     - PASS_BY_HOUR：输入数据为多个小时或分钟且非递增。适用于查询最近5分钟或最近3小时的数据。
     - ASS_BY_HOUR_AND_DAY：输入数据为多天，最近时间范围内的分区按小时分区存储。适用于查询最近3小时或最近3天的数据。
     - PASS_BY_DAY：输入数据为多天且非递增。适用于查询最近3天的数据。
     - INCREASING：输入数据为多天且递增，最近时间范围内的一个分区按天分区存储。适用于查询最近50天或最近3年的数据。
     - INCREASING_IN_A_MONTH：输入数据为本月所有数据。适用于计算从当前月一日开始到当前的作业。
     - INCREASING_IN_A_YEAR：输入数据为本年所有数据。适用于计算从当前年一月一日开始到当前的作业。
3. （可选）如果渐进式计算模式为PASS_BY_HOUR时，您需根据实际作业周期长短，配置更细粒度时间列名。如果渐进式计算为其他模式，则不需要配置。
   - 配置格式
     - 格式一：
       ```
       set odps.progressive.range.query.time.partition.col.names=default:<col_name_day>:<col_name_hour>:<col_name_minute>|<col_name_day>:<col_name_hour>:<col_name_minute>|...;
       ```
     - 格式二：
       ```
       set odps.progressive.range.query.time.partition.col.names=<table_name>:<col_name_day>:<col_name_hour>:<col_name_minute>|<col_name_day>:<col_name_hour>:<col_name_minute>|...;
       ```
     - 格式三：
       ```
       set odps.progressive.range.query.time.partition.col.names=default:<col_name_day>:<col_name_hour>:<col_name_minute>,<table_name>:<col_name_day>:<col_name_hour>:<col_name_minute>,...;
       ```
   - 参数说明：
     - default：关键字。默认搜索符合条件的全部表。
     - table_name：指定查询对应时间列的表名。可以同时指定多张表，例如**set odps.progressive.range.query.time.partition.col.names=table_1:day:hour,table_2:hour:minute;**
     - <col_name_day>:<col_name_hour>:<col_name_minute>：指定查询对应表的时间列。可以同时指定多个列，例如**set odps.progressive.range.query.time.partition.col.names=table_1:day:hour|day:hour2,table_2:day:hour:minute|day:hour2;**
   - 配置示例：

     运行频率为1小时的作业，渐进式计算配置示例如下：

     a. 配置渐进式计算模式的基础单位为1小时。
     ```
     set odps.progressive.range.query.input.partition.pattern=PASS_BY_HOUR:1;
     ```
     b. 配置更细粒度的时间列。
     ```
     set odps.progressive.range.query.time.partition.col.names=default:ds:hh|dt:hour;
     ```

渐进式计算调优配置

- 强制设置中间表存储方式为cluster table。

  渐近式计算过程中如果生成中间表，在读取中间表数据后会将数据重新组合（shuffle）。如果将中间表存储方式设置为cluster table，计算性能会显著提高。

- 配置格式：

```
set range.query.force.cluster.table=[true|false];
```

- 参数说明：
  - true：强制设置中间表存储方式为cluster table。
  - false：默认为false。系统会根据是否有shuffle操作，自动选择不同的存储方式。

> **说明** 建议您强制设置为cluster table存储方式。

- 设置作业并发运行实例数。

  渐进式计算初次执行时需要刷新涉及渐进式计算周期范围内的历史数据，此时会同时运行很多子任务。为了避免资源被全部占用，导致其它作业不能正确运行，建议您设置作业并发运行的最大实例数。

  - 配置格式：

```
set odps.progressive.combine.exec.time.limit.num=<number>;
```

  - 参数说明：

    number：默认值为15。作业并发运行的最大实例数>=1；

> **说明** 请您根据实际作业运行情况设置作业并发运行的最大实例数，实例数设置过大或者过小均会影响实际运行效率。

渐进式计算使用示例

1. 第一天新建作业

   使用渐进式计算方式查询时间列范围为[20200801,20200807]的数据，并设置渐进式计算模式为天。作业脚本如下所示：

```
set odps.progressive.enable=true;
set odps.progressive.range.query.input.partition.pattern=PASS_BY_DAY;
CREATE TABLE adl_aegis_webshell_test_neoke AS
SELECT
request_datetime,host,uri,src_ip,src_port,dst_ip,dst_port,method,post_data,user_agent,ret_code,cookie,
referer,x_forward_for,rsp_content_type,rqs_content_type,content_length,jump_location,set_cookie,ttl,
get_bigwebshell_uri(uri) AS nopar_uri,internet_ip
FROM secbase.adl_aegis_webshell_newadd_beaverlog
WHERE ds >= to_char(dateadd(to_date('20200807','yyyymmdd'),-6,'dd') ,'yyyymmdd')
AND ds <= '20200807';
```

2. 第二天（第二次）查询

   第一天执行作业脚本后，系统会根据每一个分区的数据进行计算，并将计算结果以天为单位保存到7个中间表中。在第二天查询时间列范围为[20200802,20200808]的数据时，首先会单独计算20200808列的数据，再和第一次运行生成7个中间表的数据整合起来得到最终结果。

3.15. 附录

3.15.1. 运算符

运算符用于执行程序代码运算。本文为您介绍MaxCompute支持的运算符。

MaxCompute支持的运算符如下。

运算符类型	说明
关系运算符	用于比较运算。
算术运算符	用于各类数值运算。
位运算符	用于按照二进制位进行运算。
逻辑运算符	用于多条件连接，一般用于连接BOOLEAN类型的表达式或者值。

关系运算符

运算符	说明
A=B	• A或B为NULL，返回NULL。 • A等于B，返回TRUE，否则返回FALSE。

运算符	说明
A<>B	• A或B为NULL，返回NULL。 • A不等于B，返回TRUE，否则返回FALSE。
A<B	• A或B为NULL，返回NULL。 • A小于B，返回TRUE，否则返回FALSE。
A<=B	• A或B为NULL，返回NULL。 • A小于等于B，返回TRUE，否则返回FALSE。
A>B	• A或B为NULL，返回NULL。 • A大于B，返回TRUE，否则返回FALSE。
A>=B	• A或B为NULL，返回NULL。 • A大于等于B，返回TRUE，否则返FALSE。
A IS NULL	A为NULL，返回TRUE，否则返回FALSE。
A IS NOT NULL	A不为NULL，返回TRUE，否则返回FALSE。

A LIKE B	如果A或B为NULL，返回NULL。A为字符串，B为要匹配的模式，如果匹配，返回TRUE，否则返回FALSE。
	• 百分号（%）匹配任意多个字符。
	• 下划线（_）匹配单个字符。
	• 要匹配百分号（%）或下划线（_）要用转义符表示为 `'%'` 或 `'_'` 。
	`'aaa' like 'a__'= TRUE`
	`'aaa' like 'a%' = TRUE`
	`'aaa' like 'aab'= FALSE`
	`'a%b' like 'a\\%b'= TRUE`
	`'axb' like 'a\\%b'= FALSE`

A RLIKE B	A是字符串，B是字符串常量或者正则表达式。如果匹配成功，返回TRUE，否则返回FALSE。如果B为空串会报错。如果A或B为NULL，返回NULL。
A IN B	• B为一个集合，如果A在B中，返回TRUE，否则返回FALSE。 • 如果A为NULL，返回NULL。 • 如果B包含一个NULL元素，则返回NULL。 • B必须是常数集合且至少包含一个项，且所有项的数据类型需要一致。
BETWEEN AND	表达式为 `A [NOT] BETWEEN B AND C` 。 • A、B或C为NULL，则为空。 • A大于等于B且小于等于C，返回TRUE，否则返回FALSE。
IS [NOT] DISTINCT FROM	表达式为 `A IS [NOT] DISTINCT FROM B` 。详情请参见IS DISTINCT FROM和IS NOT DISTINCT FROM。

常见用法如下。

```
SELECT * FROM user WHERE user_id = '0001';
SELECT * FROM user WHERE user_name <> 'maggie';
SELECT * FROM user WHERE age > '50';
SELECT * FROM user WHERE birth_day >= '1980-01-01 00:00:00';
SELECT * FROM user WHERE is_female is null;
SELECT * FROM user WHERE is_female is not null;
SELECT * FROM user WHERE user_id in (0001,0010);
SELECT * FROM user WHERE user_name like 'M%';
```

在执行部分关系运算之前，您首先需要进行类型转换，否则可能返回NULL，详情请参见数据类型转换。

由于DOUBLE类型存在一定的精度差，因此，不建议您直接使用等于号（=）对两个DOUBLE类型的数据进行比较。您可以将两个DOUBLE类型数据相减，然后取绝对值进行判断。当绝对值足够小时，认为两个DOUBLE数值相等，示例如下。

```
ABS(0.9999999999 - 1.0000000000) < 0.000000001
 -- 0.9999999999和1.0000000000为10位精度，而0.000000001为9位精度。
 -- 此时可以认为0.9999999999和1.0000000000相等。
```

说明
- ABS是MaxCompute提供的内建函数，用于取绝对值，详情请参见ABS。
- 通常，MaxCompute的DOUBLE类型能够保障14位有效数字。
- 比较STRING和BIGINT类型数值时，两个类型都将自动转换为DOUBLE类型，在比较过程中可能出现精度丢失现象。您可以通过CAST STRING AS BIGINT方式将STRING类型转换为BIGINT类型后再比较。

算术运算符

运算符	说明
A+B	A或B为NULL，返回NULL，否则返回A+B的结果。
A-B	A或B为NULL，返回NULL，否则返回A-B的结果。
A*B	A或B为NULL，返回NULL，否则返回A×B的结果。
A/B	A或B为NULL，返回NULL，否则返回A÷B的结果。 说明 如果A和B为BIGINT类型，返回结果为DOUBLE类型。
A%B	A或B为NULL，返回NULL，否则返回A÷B并取余数的结果。
+A	仍然返回A。
-A	如果A为NULL，返回NULL，否则返回-A。
A DIV B	A或B为NULL，返回NULL，否则返回A DIV B的结果。

常见用法如下。

```
SELECT age+10, age-10, age%10, -age, age*age, age/10, age div 10 FROM user;
```

说明
- 只有STRING、BIGINT或DOUBLE类型的参数才能参与算术运算，日期和布尔类型不允许参与运算。
- STRING类型在参与运算前会隐式转换为DOUBLE类型。
- BIGINT和DOUBLE类型共同参与计算时，系统会将BIGINT类型隐式转换为DOUBLE类型再进行计算，返回结果为DOUBLE类型。
- A和B都是BIGINT类型，执行A/B运算，返回结果为DOUBLE类型。执行上述其他运算，返回BIGINT类型。

位运算符

运算符	说明
A&B	返回A和B进行按位与运算的结果。例如1&2返回0，1&3返回1，NULL和任何值按位与运算都为NULL。A和B必须为BIGINT类型。
A|B	返回A和B进行按位或运算的结果。例如1|2返回3，1|3返回3，NULL和任何值按位或运算都为NULL。A和B必须为BIGINT类型。
A||B	字符串连接操作符。例如 `a||b||c` 相当于 `CONCAT(a, b, c)` 。

说明 位运算符不支持隐式转换，只允许BIGINT类型参与运算。

逻辑运算符

运算符	说明
A and B	TRUE and TRUE=TRUE
	TRUE and FALSE=FALSE
	FALSE and TRUE=FALSE
	FALSE and FALSE=FALSE
	FALSE and NULL=FALSE

运算符	说明
	NULL and FALSE=FALSE
	TRUE and NULL=NULL
	NULL and TRUE=NULL
	NULL and NULL=NULL
A or B	TRUE or TRUE=TRUE
	TRUE or FALSE=TRUE
	FALSE or TRUE=TRUE
	FALSE or FALSE=FALSE
	FALSE or NULL=NULL
	NULL or FALSE=NULL
	TRUE or NULL=TRUE
	NULL or TRUE=TRUE
	NULL or NULL=NULL
NOT A	A是NULL，返回NULL。
	A是TRUE，返回FALSE。
	A是FALSE，返回TRUE。

> **说明** 逻辑运算符只允许BOOLEAN类型参与运算，不支持隐式类型转换。

运算符优先级

不同运算符的优先级如下表所示，从上到下优先级递减。相同优先级默认是从左到右进行计算。

运算符	优先级
IS (NOT) NULL	1
^	2
*、/、%、DIV	3
+、-	4
||	5
&	6
|	7
(NOT) LIKE、(NOT) RLIKE、=、==、IS (NOT) DISTINCT FROM、<>、!=、<=、<、>=、>	8
(NOT) IN、(NOT) BETWEEN AND	9
NOT	10
AND	11
OR	12

如果某些运算要优先处理，需要添加英文括号。运算优先级示例如下。

```
a=1 and b=1 or c=1          --先计算a和b，再计算c。
a=1 and (b=1 or c=1)        --先计算b和c，再计算a。
```

3.15.2. 转义字符

在MaxCompute SQL中，反斜线（ `\` ）是转义字符，用来表达字符串中的特殊字符，或将其后跟的字符解释为其本身。

当读入字符串常量时，如果反斜线后跟三位有效的8进制数字，范围在001~177之间。系统会根据ASCII值转为相应的字符。

特殊字符

以下字符会被解释为特殊字符。

转义	字符
\b	backspace
\t	tab
\n	newline
\r	carriage-return
\'	单引号
\"	双引号
\ \	反斜线
\;	分号
\Z	control-Z
\0或\00	结束符

示例

- 转义字符后字符解释为其本身。

  MaxCompute SQL中的字符串常量可以用单引号或双引号表示。您可以在单引号括起的字符串中包含双引号，或在双引号括起的字符串中包含单引号，否则要用转义符来表达。表达方式如下。

  ```
  "I'm a happy manong."
  'I\'m a happy manong.'
  ```

- 特殊字符。

  ```
  select length('a\tb');
  ```

  运行结果为3。表示字符串里有三个字符， `\t` 被视为一个字符。

  ```
  select 'a\ab',length('a\ab');
  ```

  运行结果为aab，3。 `\a` 被解释成了普通的a。

3.15.3. LIKE字符匹配

本文为您介绍LIKE字符匹配的使用方法。

在LIKE匹配时， `%` 表示匹配任意多个字符， `_` 表示匹配单个字符。

如果要匹配 `%` 或 `_` 本身，则要对其进行转义， `\\%` 匹配字符 `%` ， `\\_` 匹配字符 `_` 。

```
'abcd' like 'ab%' -- true
'abcd' like 'ab_' -- false
'ab_cde' like 'ab\\_c%'; -- true
```

> **注意** 目前MaxCompute SQL支持UTF-8的字符集。如果数据以其它格式编码，计算结果可能不正确。

3.15.4. 正则表达式规范

MaxCompute SQL中的正则表达式采用的是PCRE的规范，匹配时按字符进行。

RLIKE

您可以使用**RLIKE**语句匹配正则表达式，支持的元字符如下表所示。

元字符	说明
^	行首。
$	行尾。
.	任意字符。
*	匹配0次或多次。
+	匹配1次或多次。
?	匹配0次或1次。
?	匹配修饰符，当该字符跟在任何一个其他限制符 (* 、+、?、 {n}、 {n,}、 {n,m}）后面时，匹配模式为非贪婪模式。非贪婪模式尽可能少的匹配所搜索的字符串， 而默认的贪婪模式则尽可能多的匹配所搜索的字符串。
A|B	A或B。
(abc)*	匹配abc序列0次或多次。
{n}或{m,n}	匹配的次数。
[ab]	匹配括号中的任一字符。模式匹配a或b。
[a-d]	匹配a、b、c、d任一字符。
[^ab]	^表示非，匹配任一非a非b的字符。
[::]	见下表POSIX字符组。
\	转义符。
\n	n为数字1-9，后向引用。
\d	数字。
\D	非数字。

字符组

字符组类型	字符组	说明	范围
POSIX字符组	[[:alnum:]]	字母字符和数字字符	[a-zA-Z0-9]
	[[:alpha:]]	字母	[a-zA-Z]
	[[:ascii:]]	ASCII字符	[\x00-\x7F]
	[[:blank:]]	空格字符和制表符	[\t]
	[[:cntrl:]]	控制字符	[\x00-\x1F\x7F]
	[[:digit:]]	数字字符	[0-9]
	[[:graph:]]	空白字符之外的字符	[\x21-\x7E]
	[[:lower:]]	小写字母字符	[a-z]
	[[:print:]]	[:graph:]和空白字符	[\x20-\x7E]
	[[:punct:]]	标点符号	[][!" #$%&'()*+,./:;<=>? @\^_`{|}~-]
	[[:space:]]	空白字符	[\t\r\n\v\f]
	[[:upper:]]	大写字母字符	[A-Z]
	[[:xdigit:]]	十六进制字符	[A-Fa-f0-9]
	不涉及	包括汉字在内的双字节字符	[^\\x{00}-\\x{ff}]
	不涉及	中文字符	[\\x{4e00}-\\x{9fa5}]

字符组类型 中文字符组	字符组	说明	范围
	不涉及	中文标点	中文标点无统一的编码范围，您可以在搜索引擎上搜索**中文标点符号Unicode码**，然后结合运算符来逐个排除。 例如句号（。）对应正则表达式为 `[\\x{3002}]` 。执行 `select * from values ('你好。'),('nihao！') t(d) where d rlike '[\\x{3002}]';` 返回结果为 `你好。` 。

转义符

由于系统采用反斜线 `\` 作为转义符，因此正则表达式的模式中出现的 `\` 都要进行二次转义。例如正则表达式要匹配字符串 `a+b` 。其中 `+` 是正则中的一个特殊字符，因此要用转义的方式表达，在正则引擎中的表达方式是 `a\\+b` 。由于系统还要解释一层转义，因此能够匹配该字符串的表达式是 `a\\\+b` 。

示例如下。

```
select 'a+b' rlike 'a\\\+b';
+------+
| _c1  |
+------+
| true |
+------+
```

极端的情况，如果要匹配字符 `\` ，由于在正则引擎中 `\` 是一个特殊字符，因此要表示为 `\\` ，而系统还要对表达式进行一次转义，因此写成 `\\\\` 。

```
select 'a\\b', 'a\\b' rlike 'a\\\b';
+-----+------+
| _c0 | _c1  |
+-----+------+
| a\b | false |
+-----+------+
select 'a\\b', 'a\\b' rlike 'a\\\\b';
+-----+------+
| _c0 | _c1  |
+-----+------+
| a\b | true |
+-----+------+
```

> **说明** 在MaxCompute SQL中写 `a\\b` ，而在输出结果中显示 `a\b` ，同样是因为MaxCompute会对表达式进行转义。

如果字符串中有制表符TAB，系统在读入 `\t` 这两个字符时，已经将其存为一个字符，因此在正则的模式中它也是一个普通的字符。

```
select 'a\tb', 'a\tb' rlike 'a\tb';
+---------+------+
| _c0     | _c1  |
+---------+------+
| a     b | true |
+---------+------+
```

3.15.5. 保留字与关键字

本文为您介绍MaxCompute SQL中的所有保留字与关键字。

> **注意**
> - 命名表、列或分区时，不要使用保留字与关键字，否则可能会报错。
> - 保留字不区分大小写。
> - 在对表、列或是分区命名时如若使用关键字，需给关键字加 ``` 符号进行转义，否则会报错。

```
%    &    &&    (    )    *    +
-    .    /    ;    <    <=    <>
=    >    >=    ?    ADD    AFTER    ALL
ALTER    ANALYZE    AND    ARCHIVE    ARRAY    AS    ASC
BEFORE    BETWEEN    BIGINT    BINARY    BLOB    BOOLEAN    BOTH    DECIMAL
BUCKET    BUCKETS    BY    CASCADE    CASE    CAST    CFILE
CHANGE    CLUSTER    CLUSTERED    CLUSTERSTATUS    COLLECTION    COLUMN    COLUMNS
COMMENT    COMPUTE    CONCATENATE    CONTINUE    CREATE    CROSS    CURRENT
CURSOR    DATA    DATABASE    DATABASES    DATE    DATETIME    DBPROPERTIES
DEFERRED    DELETE    DELIMITED    DESC    DESCRIBE    DIRECTORY    DISABLE
DISTINCT    DISTRIBUTE    DOUBLE    DROP    ELSE    ENABLE    END    EXCEPT
ESCAPED    EXCLUSIVE    EXISTS    EXPLAIN    EXPORT    EXTENDED    EXTERNAL
FALSE    FETCH    FIELDS    FILEFORMAT    FIRST    FLOAT    FOLLOWING
FORMAT    FORMATTED    FROM    FULL    FUNCTION    FUNCTIONS    GRANT
GROUP    HAVING    HOLD_DDLTIME    IDXPROPERTIES    IF    IMPORT    IN
INDEX    INDEXES    INPATH    INPUTDRIVER    INPUTFORMAT    INSERT    INT
INTERSECT    INTO    IS    ITEMS    JOIN    KEYS    LATERAL
LEFT    LIFECYCLE    LIKE    LIMIT    LINES    LOAD    LOCAL
LOCATION    LOCK    LOCKS    LONG    MAP    MAPJOIN    MATERIALIZED
MINUS    MSCK    NOT    NO_DROP    NULL    OF    OFFLINE    OFFSET
ON    OPTION    OR    ORDER    OUT    OUTER    OUTPUTDRIVER
OUTPUTFORMAT    OVER    OVERWRITE    PARTITION    PARTITIONED    PARTITIONPROPERTIES    PARTITIONS
PERCENT    PLUS    PRECEDING    PRESERVE    PROCEDURE    PURGE    RANGE
RCFILE    READ    READONLY    READS    REBUILD    RECORDREADER    RECORDWRITER
REDUCE    REGEXP    RENAME    REPAIR    REPLACE    RESTRICT    REVOKE
RIGHT    RLIKE    ROW    ROWS    SCHEMA    SCHEMAS    SELECT
SEMI    SEQUENCEFILE    SERDE    SERDEPROPERTIES    SET    SHARED    SHOW
SHOW_DATABASE    SMALLINT    SORT    SORTED    SSL    STATISTICS    STATUS    STORED
STREAMTABLE    STRING    STRUCT    TABLE    TABLES    TABLESAMPLE    TBLPROPERTIES
TEMPORARY    TERMINATED    TEXTFILE    THEN    TIMESTAMP    TINYINT    TO
TOUCH    TRANSFORM    TRIGGER    TRUE    TYPE    UNARCHIVE    UNBOUNDED    UNDO
UNION    UNIONTYPE    UNIQUEJOIN    UNLOCK    UNSIGNED    UPDATE    USE
USING    UTC    UTC_TMESTAMP    VIEW    WHEN    WHERE    WHILE    DIV
```

3.15.6. 数据类型映射表

本文为您介绍MaxCompute与Hive、Oracle、MySQL之间数据类型的映射关系。

MaxCompute与Hive、Oracle、MySQL的数据类型映射表，如下所示。

MaxCompute数据类型	Hive数据类型	Oracle数据类型	MySQL数据类型
BOOLEAN	BOOLEAN	无 **说明** 使用过程中用 `CHAR(1)` 、 `INTEGER` 或 `NUMBER(1)` 等进行替代，1为真，0为假。	无 **说明** 使用过程用 `TINYINT(1)` 替代。
TINYINT	TINYINT	NUMBER(3,0)	TINYINT
SMALLINT	SMALLINT	NUMBER(5,0)	SMALLINT
INT	INT	NUMBER(7,0)	MEDIUMINT
INT	INT	NUMBER(10,0)	INT
BIGINT	BIGINT	NUMBER(20,0)	BIGINT
FLOAT	FLOAT	BINARY_FLOAT **说明** Oracle自10g开始有此类型。	FLOAT
DOUBLE	DOUBLE	BINARY_DOUBLE **说明** Oracle自10g开始有此类型。	DOUBLE

MaxCompute数据类型	Hive数据类型	Oracle数据类型	MySQL数据类型
DECIMAL	DECIMAL	NUMBER(P,S)	• DECIMAL • NUMERIC
STRING	STRING	• VARCHAR • VARCHAR2 • CHAR • NCHAR • NVARCHAR3	• VARCHAR • CHAR
VARCHAR	VARCHAR	• VARCHAR • VARCHAR2 • CHAR • NCHAR • NVARCHAR3	VARCHAR
STRING	CHAR	CHAR	CHAR
BINARY	BINARY	RAW	• BINARY • VARBINARY
TIMESTAMP	TIMESTAMP	TIMESTAMP(N)	TIMESTAMP
DATETIME	DATE	DATE	DATETIME
ARRAY	ARRAY	不支持	不支持
MAP	`MAP<key,value>`	不支持	不支持
STRUCT	STRUCT	不支持	不支持
不支持	UNION	不支持	不支持

3.15.7. 数据类型转换

MaxCompute SQL允许数据类型之间的转换，类型转换方式包括显式类型转换和隐式类型转换。

显式类型转换

显式类型转换是通过CAST函数将一种数据类型的值转换为另一种类型的值，在MaxCompute SQL中支持的显式类型转换，如下表所示。 关于CAST的介绍请参见CAST。

From/To	BIGINT	DOUBLE	STRING	DATETIME	BOOLEAN	DECIMAL	FLOAT
BIGINT	不涉及	Y	Y	N	Y	Y	Y
DOUBLE	Y	不涉及	Y	N	Y	Y	Y
STRING	Y	Y	不涉及	Y	Y	Y	Y
DATETIME	N	N	Y	不涉及	N	N	N
BOOLEAN	Y	Y	Y	N	不涉及	Y	Y
DECIMAL	Y	Y	Y	N	Y	不涉及	Y
FLOAT	Y	Y	Y	N	Y	Y	不涉及

其中，Y表示可以转换，N表示不可以转换，不涉及表示不需要转换。不支持的显式类型转换会失败并报错退出。

示例

```
SELECT CAST(user_id AS DOUBLE) AS new_id;
SELECT CAST('2015-10-01 00:00:00' AS DATETIME) AS new_date;
SELECT CAST(ARRAY(1,2,3) AS ARRAY<STRING>);
SELECT CONCAT_WS(',', CAST(ARRAY(1, 2) AS ARRAY<STRING>));
```

使用说明和限制

- 将DOUBLE类型转为BIGINT类型时，小数部分会被截断，例如 `CAST(1.6 AS BIGINT) = 1` 。
- 满足DOUBLE格式的STRING类型转换为BIGINT时，会先将STRING转换为DOUBLE，再将DOUBLE转换为BIGINT，因此，小数部分会被截断，例如 `CAST("1.6" AS BIGINT) = 1` 。
- 满足BIGINT格式的STRING类型可以被转换为DOUBLE类型，小数点后保留一位，例如 `CAST("1" AS DOUBLE) = 1.0` 。
- 日期类型转换时采用默认格式 `yyyy-mm-dd hh:mi:ss` 。
- 部分类型之间不可以通过显式的类型转换，但可以通过SQL内建函数进行转换，例如从BOOLEAN类型转换到STRING类型，可使用函数 `TO_CHAR` ，详情请参见TO_CHAR。而 `TO_DATE` 函数同样支持从STRING类型到DATETIME类型的转换，详情请参见TO_DATE。
- DECIMAL超出值域， `CAST STRING TO DECIMAL` 可能会出现最高位溢出报错、最低位溢出截断等情况。
- DECIMAL类型显示转换为DOUBLE、FLOAT等类型会产生精度损失，对于精度有要求的场景，例如计算金额、费率等，建议使用DECIMAL类型。
- MaxCompute支持复杂类型的类型转换功能。其中复杂类型的隐式类型转换要求子类型能够隐式转换，而显示转换要求子类型能够显示转换。STRUCT类型转换不要求字段名称一致，但是要求字段的数量一致，且对应的字段能够隐式或显示转换。例如：
  - `ARRAY<BIGINT>` 能隐式转换或显示转换为 `ARRAY<STRING>` 。
  - `ARRAY<BIGINT>` 能显示转换为 `ARRAY<INT>` ，但是不能隐式转换。
  - `ARRAY<BIGINT>` 不能隐式转换或显示转换为 `ARRAY<DATETIME>` 。
  - `STRUCT<a:BIGINT,b:INT>` 能隐式转换为 `STRUCT<col1:STRING,col2:BIGINT>` ，但是不能隐式或显示转换为 `STRUCT<a:STRING>` 。

隐式类型转换及其作用域

隐式类型转换是指在运行时，由MaxCompute依据上下文使用环境及类型转换规则自动进行的类型转换。MaxCompute支持的隐式类型转换规则，如下表所示。

From/To	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	FLOAT
BOOLEAN	Y	N	N	N	N	N
TINYINT	N	Y	Y	Y	Y	Y
SMALLINT	N	N	Y	Y	Y	Y
INT	N	N	Y	Y	Y	Y
BIGINT	N	N	N	N	Y	Y
FLOAT	N	N	N	N	Y	Y

From/To	DOUBLE	DECIMAL	STRING	VARCHAR	TIMESTAMP	BINARY
DOUBLE	Y	Y	Y	Y	N	N
DECIMAL	N	Y	Y	Y	N	N
STRING	Y	Y	Y	Y	N	N
VARCHAR	Y	Y	N	N	不涉及	不涉及
TIMESTAMP	N	N	Y	Y	Y	N
BINARY	N	N	N	N	N	Y

其中，Y表示可以转换，N表示不可以转换，不涉及表示不需要转换。不支持的隐式类型转换会导致异常。如果在执行时转换失败，也会导致异常。

> 说明
> - MaxCompute 2.0新增了DECIMAL类型与DATETIME的常量定义方式，100BD是数值为100的DECIMAL， `2017-11-11 00:00:00` 是DATETIME类型的常量。VALUES子句和VALUES表中可以直接使用常量定义。
> - 由于隐式类型转换是MaxCompute依据上下文使用环境自动进行的类型转换，因此推荐您在类型不匹配时，显式地用CAST进行转换。
> - 隐式类型转换规则是有发生作用域的。在某些作用域中，只有一部分规则可以生效。

示例

```
SELECT user_id+age+'12345', CONCAT(user_name,user_id,age) FROM user;
```

不同运算符作用下的隐式转换如下：

- **关系运算符作用下的隐式转换**

  关系运算符包括 `=、<>、<、<=、>、>=、IS NULL、IS NOT NULL、LIKE、RLIKE、IN` 。由于 `LIKE、RLIKE、IN` 的隐式类型转换规则不同于其他关系运算符，将单独对其进行说明。此处的说明不包含这三种特殊的关系运算符。

当不同类型的数据共同参与关系运算时，按照下述原则进行隐式类型转换。

From/To	BIGINT	DOUBLE	STRING	DATETIME	BOOLEAN	DECIMAL
BIGINT	不涉及	DOUBLE	DOUBLE	N	N	DECIMAL
DOUBLE	DOUBLE	不涉及	DOUBLE	N	N	DECIMAL
STRING	DOUBLE	DOUBLE	不涉及	DATETIME	N	DECIMAL
DATETIME	N	N	DATETIME	不涉及	N	N
BOOLEAN	N	N	N	N	不涉及	N
DECIMAL	DECIMAL	DECIMAL	DECIMAL	N	N	不涉及

> 说明
> - 如果进行比较的两个类型间不能进行隐式类型转换，则该关系运算不能完成，报错退出。
> - 关系运算符的更多详情，请参见运算符。

- **特殊的关系运算符作用下的隐式转换**

  特殊的关系运算符包括 `LIKE、RLIKE、IN` 。

  - LIKE和RLIKE的使用方式，如下所示。

    ```
    source LIKE pattern;
    source RLIKE pattern;
    ```

    > 说明
    > - LIKE和RLIKE的source和pattern参数均仅接受STRING类型。
    > - 其他类型不允许参与运算，也不能进行到STRING类型的隐式类型转换。

  - IN的使用方式如下。

    ```
    key IN (value1, value2, …)
    ```

    > 说明
    > - IN右侧的VALUE值列表中的数据类型必须一致。
    > - 当KEY与VALUES之间比较时，如果数据类型包含BIGINT、DOUBLE、STRING，建议统一转为DOUBLE类型；如果数据类型包含DATETIME、STRING，建议统一转为DATETIME类型。除此之外不允许其它类型之间的转换。

- **算术运算符作用下的隐式转换**

  算术运算符包括 `+、-、*、/、%` ，其隐式转换规则如下：

  - 只有STRING、BIGINT、DOUBLE和DECIMAL才能参与算术运算。
  - STRING在参与运算前会进行隐式类型转换到DOUBLE。
  - BIGINT和DOUBLE共同参与计算时，会将BIGINT隐式转换为DOUBLE。
  - 日期型和布尔型不允许参与算数运算。

- **逻辑运算符作用下的隐式转换**

  逻辑运算符包括 `and、or、not` ，其隐式转换规则，如下：

  - 只有BOOLEAN才能参与逻辑运算。
  - 其他类型不允许参与逻辑运算，也不允许其他类型的隐式类型转换。

内建函数涉及到的隐式转换

MaxCompute SQL提供了大量的系统函数，方便您对任意行的一列或多列进行计算，输出任意种的数据类型。其隐式转换规则如下：

- 在调用函数时，如果输入参数的数据类型与函数定义的参数数据类型不一致，把输入参数的数据类型转换为函数定义的数据类型。
- 每个MaxCompute SQL内建函数的参数对于允许的隐式类型转换的要求不同，详情请参见内建函数概述。

CASE WHEN作用下的隐式转换

CASE WHEN的详情介绍请参见CASE WHEN表达式。它的隐式转换规则，如下：

- 如果返回类型只有BIGINT、DOUBLE，统一转换为DOUBLE。
- 如果返回类型中有STRING类型，统一转换为STRING，如果不能转换（如BOOLEAN类型）则报错。

- 除此之外不允许其它类型之间的转换。

STRING与DATETIME类型之间的转换

MaxCompute支持STRING类型和DATETIME类型之间的相互转换。转换时使用的格式为 `yyyy-mm-dd hh:mi:ss` 。

单位	字符串（忽略大小写）	有效值域
年	yyyy	0001~9999
月	mm	01~12
日	dd	01~28|29|30|31
时	hh	00~23
分	mi	00~59
秒	ss	00~59

> **说明**
> - 各个单位的值域中，如果首位为0，不可省略。例如 `2014-1-9 12:12:12` 为非法的DATETIME格式，无法从STRING类型数据转换为DATETIME类型，必须写为 `2014-01-09 12:12:12` 。
> - 只有符合上述格式描述的STRING类型才能够转换为DATETIME类型，例如 `CAST("2013-12-31 02:34:34" AS DATETIME)` ，将会把STRING类型 `2013-12-31 02:34:34` 转换为DATETIME类型。同理，DATETIME转换为STRING时，默认转换为 `yyyy-mm-dd hh:mi:ss` 格式。

MaxCompute提供了TO_DATE函数，用于将不满足日期格式的STRING类型数据转换为DATETIME类型。详情请参见TO_DATE。

3.15.8. UDAF和UDTF动态参数说明

本文为您介绍MaxCompute的UDAF和UDTF使用Resolve注解时支持的动态参数语法。

Resolve注解语法扩展

MaxCompute的UDAF和UDTF使用Resolve注解决定函数的Signature，示例如下。这种方式的局限性在于输入参数和输出参数是固定的，无法实现方法重载。

```
@com.aliyun.odps.udf.annotation.Resolve("BIGINT->DOUBLE")
public class UDTFClass extends UDTF {
    ...
}
```

> **说明** 示例定义了一个UDTF，该UDTF接受的参数类型为BIGINT，返回值类型为DOUBLE。

现在MaxCompute对Resolve注解语法做了如下扩展：

- 您可以在参数列表中使用 `*` ，表示接受任意长度、任意类型的输入参数。例如 `@Resolve('double,*->String')` 表示接受第一个参数是DOUBLE类型，后接任意长度、任意类型的参数列表。此时，您需要自己编写代码判断输入的个数和参数类型，然后对它们进行相应的操作（您可以对比C语言里面的 `printf` 函数来理解此操作）。

  > **说明** 星号用在返回值列表中时，表示的是不同的含义。

- 您可以在参数列表中使用 `any` 关键字，表示任意类型的参数。例如 `@Resolve('double,any->string')` 表示接受第一个是DOUBLE类型，后接任意类型的参数列表。

  > **说明** `any` 不能用在返回值列表中，也不能用在复杂类型的子类型中（例如ARRAY）。

- UDTF的返回值可以使用星号，表示返回任意个STRING类型。返回值的个数与调用函数时设置的别名个数有关。例如 `@Resolve("ANY,ANY->DOUBLE,*")` ，调用方式是 `UDTF(x, y) as (a, b, c)` ，此处 `as` 后面设置了三个别名，即a、b、c。编辑器会认定a为DOUBLE类型（Annotation中返回值第一列的类型是给定的），b和c为STRING类型。因为这里给出了三个返回值，所以UDTF在调用 `forward` 时，必须 `forward` 长度为3的数组，否则会出现运行时报错。

  > **说明** 这种错误无法在编译时报出，因此UDTF的调用者在SQL中设置alias个数时，必须遵循该UDTF定义的规则。由于聚合函数的返回值个数固定是1，所以这个功能对UDAF来说并无意义。

UDTF示例

```
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.annotation.Resolve;
import org.json.JSONException;
import org.json.JSONObject;
@Resolve("STRING,*->STRING,*")
public class JsonTuple extends UDTF {
  private Object[] result = null;
  @Override
  public void process(Object[] input) throws UDFException {
    if (result == null) {
      result = new Object[input.length];
    }
    try {
      JSONObject obj = new JSONObject((String)input[0]);
      for (int i = 1; i < input.length; i++) {
        // 返回值要求变长部分都是STRING。
        result[i] = String.valueOf(obj.get((String)(input[i])));
      }
      result[0] = null;
    } catch (JSONException ex) {
      for (int i = 1; i < result.length; i++) {
        result[i] = null;
      }
      result[0] = ex.getMessage();
    }
    forward(result);
  }
}
```

以上UDTF示例中，返回值个数会根据输入参数的个数来决定。输出参数中的第一个参数是一个JSON文本，后面的参数是需要从JSON中解析的Key。返回值中的第一个返回值是解析JSON过程中的出错信息，如果没有出错，则会根据输入的Key依次输出从JSON中解析出来的内容。使用示例如下。

```
-- 根据输入参数的个数定制输出别名个数。
SELECT my_json_tuple(json, 'a', 'b') as exceptions, a, b FROM jsons;
-- 变长部分可以一列都没有。
SELECT my_json_tuple(json) as exceptions, a, b FROM jsons;
-- 下面这个SQL会出现运行时错误，因为别名个数与实际输出个数不符。
-- 注意编译时无法发现这个错误。
SELECT my_json_tuple(json, 'a', 'b') as exceptions, a, b, c FROM jsons;
```

当本文介绍的这些扩展无法满足您的业务需求时，建议您使用UDT实现聚合函数和UDTF的功能。详情请参见UDT概述。

4.MaxCompute SQLML

4.1. 概述

SQLML是MaxCompute提供的应用机器学习能力的SQL语言入口。MaxCompute SQLML在底层依赖机器学习PAI平台做模型创建、预测、评估等操作。本文为您介绍SQLML的功能、支持的机器学习模型、支持的预测模型函数和评估模型函数。

功能介绍

MaxCompute SQLML功能依赖MaxCompute和机器学习PAI阿里云产品。您可以通过客户端开发MaxCompute SQLML作业，基于机器学习PAI对MaxCompute上的数据进行学习，并利用机器学习模型对数据进行预测，进而为业务规划提供指导。其中：

- MaxCompute：提供应用机器学习能力的SQL语言入口SQLML。
- 客户端：SQL语言操作平台，您可以自行选择DataWorks（推荐）、MaxCompute SDK（Java SDK或Python SDK）、MaxCompute客户端（odpscmd）或MaxCompute Studio。
- 机器学习PAI：提供机器学习模型。

MaxCompute SQLML可以帮助数据开发工程师、分析师和数据科学家，使用SQL语言创建、训练和应用机器学习模型。同时助力SQL从业人员利用现有的SQL技能就可以应用机器学习，无需做数据迁移，实现机器学习的能力。

使用说明

MaxCompute SQLML功能的使用流程如下：

1. 开通MaxCompute、DataWorks（基础版）及机器学习PAI服务（PAI（Studio、DSW、EAS）后付费）。
2. 准备数据集

   用于模型训练、预测的数据信息。
3. 创建或配置DataWorks工作空间，选择计算引擎服务为**MaxCompute**，选择机器学习服务为**PAI Studio**。
4. 通过DataWorks创建待导入数据集信息的表并导入数据。
5. 对导入的数据按照模型要求进行处理，并创建训练数据集和测试数据集。训练数据集用于模型训练，测试数据集用于预测。
6. 创建机器学习模型，并通过MaxCompute提供的预测模型函数进行预测。
7. 通过MaxCompute提供的评估模型函数评估预测结果的准确性。

快速使用示例请参见快速入门。

支持的机器学习模型

MaxCompute SQLML当前支持如下机器学习模型：

- 逻辑回归二分类：模型名称为logisticregression_binary。更多信息，请参见线性支持向量机。
- 逻辑回归多分类：模型名称为logisticregression_multi。更多信息，请参见PS-SMART多分类。
- 线性回归：模型名称为linearregression。更多信息，请参见GBDT回归。

支持的预测模型函数

MaxCompute SQLML当前支持的预测模型函数为 `ml_predict` ，语法格式如下：

```
ml_predict(model <model_name>, table <data_source>[, map<string, string> <parameters>])
```

- model_name：必填。新建的模型名称。
- data_source：必填。用于预测的数据，可以为表或一条SELECT语句。
- parameters：可选。用于预测的参数，参数和机器学习PAI平台的参数一致，请参见线性支持向量机、PS-SMART多分类或GBDT回归。

支持的评估模型函数

MaxCompute SQLML当前支持如下评估模型函数，用于评估预测结果的准确性：

- 二分类评估：通过内建函数 `ml_evaluate` 实现。支持计算AUC、KS及F1 score等。语法格式如下：

  ```
  ml_evaluate(table <data_source>[, map<string, string> <parameters>])
  ```
- 多分类评估：通过内建函数 `ml_multiclass_evaluate` 实现。基于模型的预测结果和原始结果，评价多分类算法模型的优劣，指标包括Accuracy、kappa、F1-Score等。语法格式如下：

  ```
  ml_multiclass_evaluate(table <data_source>[, map<string, string> <parameters>])
  ```
- 线性回归评估：通过内建函数 `ml_regression_evaluate` 实现。基于模型的预测结果和原始结果，评价回归算法模型的优劣，包含指标和残差直方图。其中：指标包括SST、SSE、SSR、R2、R、 MSE、RMSE、MAE、MAD、MAPE、count、yMean和predictMean。语法格式如下：

  ```
  ml_regression_evaluate(table <data_source>[, map<string, string> <parameters>])
  ```

上述语法格式中：

- data_source：必填。待评估的数据。需要包含label结果和prediction result，可以为表或一条SELECT语句。
- parameters：可选。预测的参数，参数和机器学习平台PAI平台的参数一致，请参见线性支持向量机、PS-SMART多分类或GBDT回归。

4.2. 快速入门

本文以开源数据集（Mushroom Data Set）为例，为您介绍如何快速使用MaxCompute SQLML及机器学习的逻辑回归二分类模型预测蘑菇是否有毒。

前提条件

已注册阿里云账号，并完成实名认证。更多信息，请参见准备阿里云账号。

如果您需要使用RAM用户身份进行操作，请确认账号可用并已授权，详情请参见准备RAM用户。

操作步骤

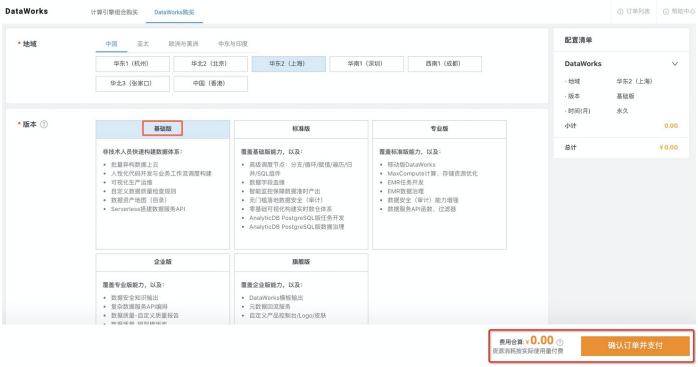
1. （可选）开通MaxCompute按量计费服务、DataWorks（基础版）服务及机器学习PAI（PAI（Studio、DSW、EAS）后付费）服务，三种服务的开通区域保持一致。
   i. 进入阿里云MaxCompute产品首页，单击**立即开通**。

   更多MaxCompute服务开通信息，请参见开通MaxCompute和DataWorks。

   > **说明**
   > - 如果您未开通过MaxCompute服务，通过该方式开通MaxCompute服务时，默认会为您开通DataWorks基础版服务（免费）和MaxCompute按量计费服务。
   > - 如果您已开通MaxCompute按量计费服务，请忽略本步骤。

   ii. 进入DataWorks购买页面，购买**基础版**服务。

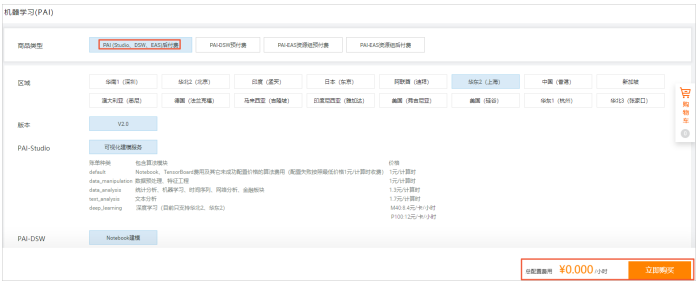
   更多DataWorks服务开通信息，请参见开通DataWorks。

   > **说明** 如果您已开通DataWorks基础版服务，请忽略本步骤。

   iii. 进入机器学习PAI购买页面，购买PAI（Studio、DSW、EAS）后付费服务。

   更多机器学习PAI服务开通信息，请参见开通PAI。

   > **说明** 如果您已开通PAI（Studio、DSW、EAS）后付费服务，请忽略本步骤。

2. 下载Mushroom Data Set数据集文件agaricus-lepiota.data，并保存为TXT、CSV或LOG文件类型。例如agaricus-lepiota.data.txt。
3. 登录DataWorks控制台，创建或配置DataWorks工作空间。
   - 如果已有DataWorks工作空间，请进入工作空间列表页面，在目标工作空间的操作列，单击**修改服务配置**，配置计算引擎服务为**MaxCompute**（按量付费），机器学习服务为**PAI Studio**（按量付费）。



- 如果没有DataWorks工作空间，请创建DataWork工作空间。配置计算引擎服务为**MaxCompute**（按量付费），机器学习服务为**PAI Studio**（按量付费）。更多创建DataWork工作空间信息，请参见创建工作空间。



4. 通过DataWorks创建表mushroom_classification并导入准备好的数据集信息。

i. 单击目标DataWorks工作空间**操作**列的**进入数据开发**，创建表mushroom_classification。

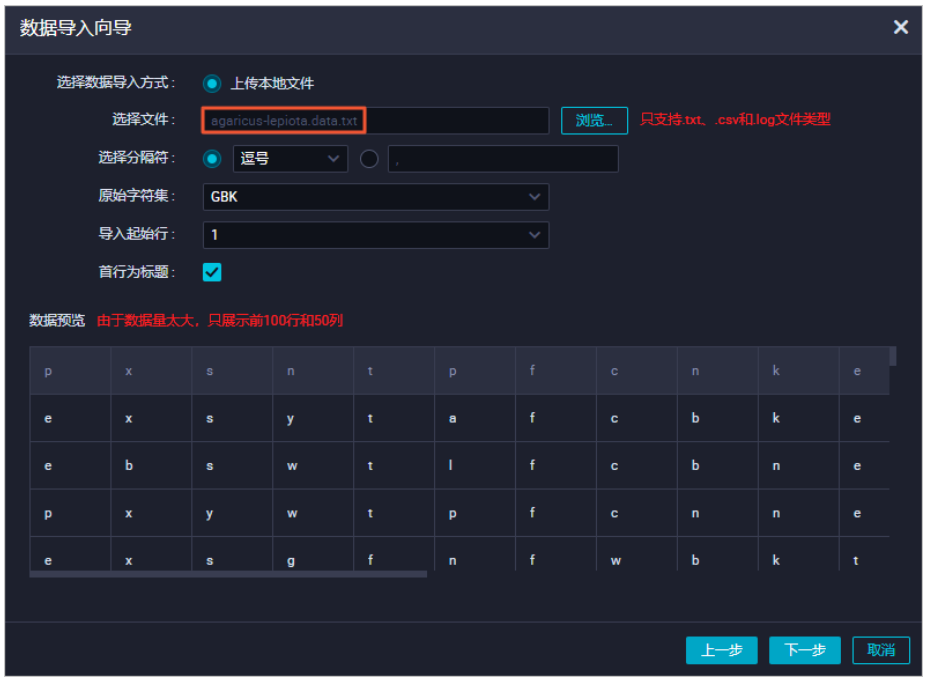
更多创建表操作信息，请参见创建MaxCompute表。

创建表的DDL语句示例如下：

```
create table mushroom_classification (
    label      string              comment 'poisonous=p,edible=e',
    cap_shape  string              comment 'bell=b,conical=c,convex=x,flat=f,knobbed=k,sunken=s',
    cap_surface string             comment 'fibrous=f,grooves=g,scaly=y,smooth=s',
    cap_color string               comment 'brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yello
w=y',
    bruises string                 comment 'bruises=t,no=f',
    odor string                    comment 'almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s'
,
    gill_attachment string         comment 'attached=a,descending=d,free=f,notched=n',
    gill_spacing string            comment 'close=c,crowded=w,distant=d',
    gill_size string               comment 'broad=b,narrow=n',
    gill_color string              comment 'black=k,brown=n,buff=b,chocolate=h,gray=g,green=r,orange=o,pink=p,purple=u,r
ed=e,white=w,yellow=y',
    stalk_shape string             comment 'enlarging=e,tapering=t',
    stalk_root string              comment 'bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?',
    stalk_surface_above_ring string comment 'fibrous=f,scaly=y,silky=k,smooth=s',
    stalk_surface_below_ring string comment 'fibrous=f,scaly=y,silky=k,smooth=s',
    stalk_color_above_ring string  comment 'brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y',
    stalk_color_below_ring string  comment 'brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y',
    veil_type string               comment 'partial=p,universal=u',
    veil_color string              comment 'brown=n,orange=o,white=w,yellow=y',
    ring_number string             comment 'none=n,one=o,two=t',
    ring_type string               comment 'cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=
z',
    spore_print_color string       comment 'black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=
y',
    population string              comment 'abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y',
    habitat string                 comment 'grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d'
);
```

ii. 将数据集文件agaricus-lepiota.data.txt的信息导入表mushroom_classification中，字段匹配方式选择**按位置匹配**。

更多上传数据操作信息，请参见上传本地数据。



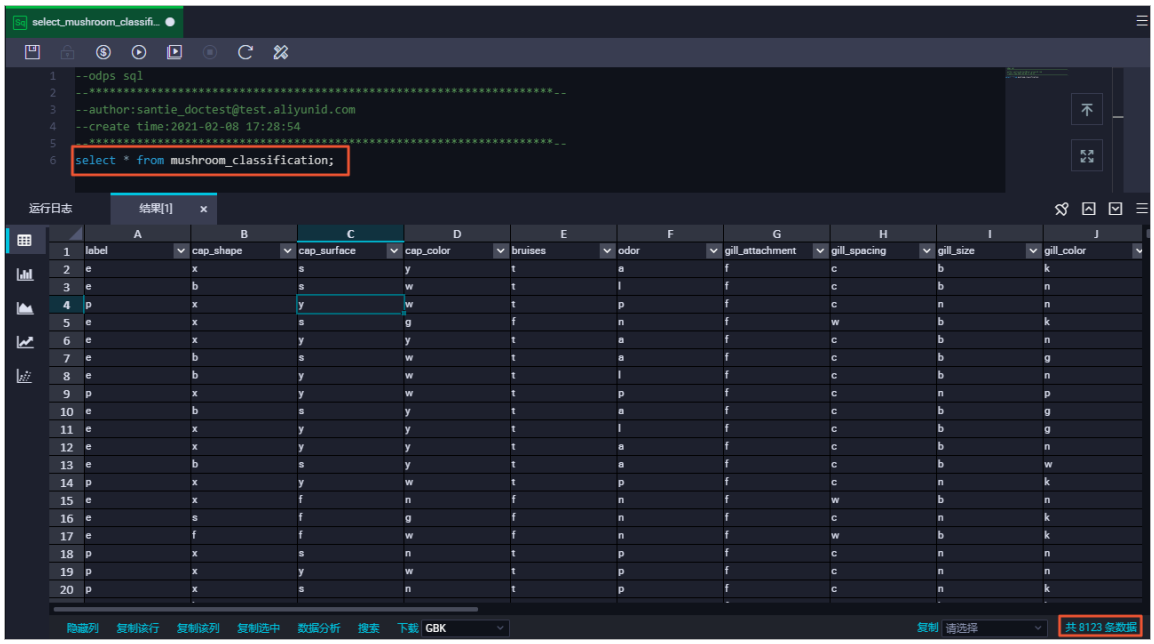
iii. 使用DataWorks的临时查询功能，新建MaxCompute ODPS SQL节点，执行SQL命令验证数据导入结果。

更多临时查询操作信息，请参见使用临时查询运行SQL语句（可选）。

命令示例如下：

```
select * from mushroom_classification;
```

返回结果如下：



5. 对导入表mushroom_classification中的数据通过one-hot编码方式进行处理。

由于逻辑回归二分类模型要求字段为数值类型，此处通过one-hot编码方式，将枚举类型的值转为数值类型。例如cap_shape对应的值为 `b、c、x、f、k、s` 6个值，one-hot编码方式会将这6个枚举值转为6列，每一列对应一个枚举值，当cap_shape的值与对应列的枚举值相等时填1，否则填0。

i. （可选）创建业务流程。例如mc_test。

更多创建业务流程操作信息，请参见创建业务流程。

> **说明** 如果您已有创建好的业务流程，可直接使用，请忽略本步骤。

ii. 新建MaxCompute ODPS Script节点，编写代码，对导入的数据按照one-hot编码方式进行处理并写入新表mushroom_classification_one_hot中。

更多创建ODPS Script节点信息，请参见创建ODPS Script节点。

命令示例如下：

```
create temporary function one_hot as 'onehot.OneHotEncoding' using
#CODE ('lang'='JAVA')
package onehot;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.annotation.Resolve;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
@Resolve({"string,string,string,string,string,string,string,string,string,string," +
        "string,string,string,string,string,string,string,string,string,string,string" +
        "->" +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,"+
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
```

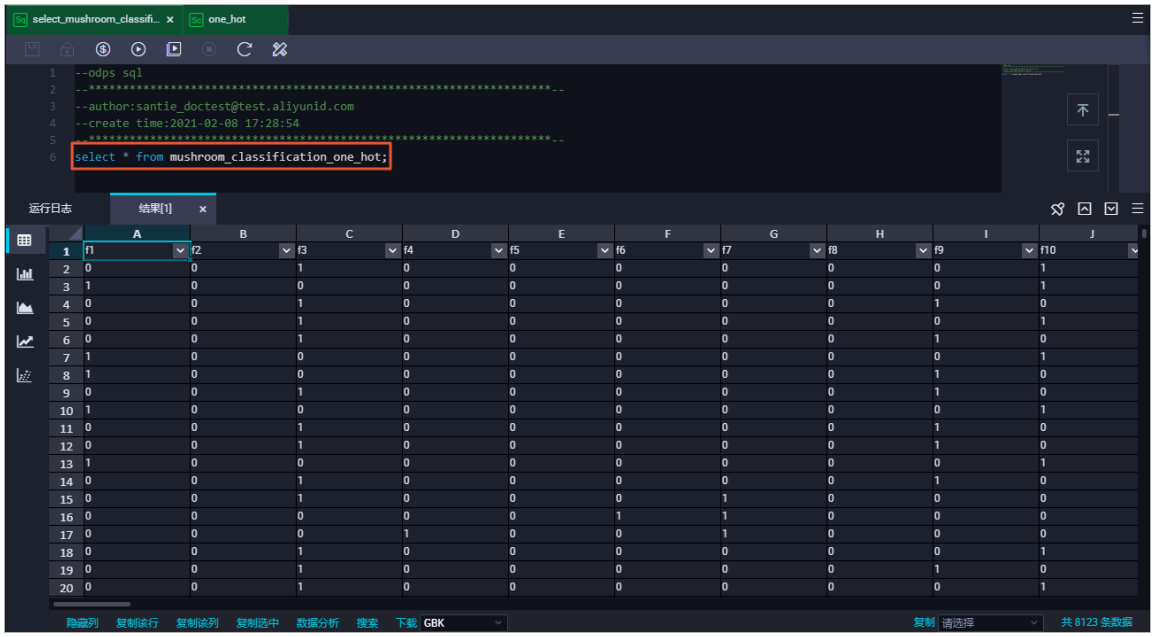
```
        "bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint,bigint," +
        "bigint,bigint,bigint,bigint,bigint,bigint"})
public class OneHotEncoding extends UDTF {
  private static char[][] features = {
          { 'b','c','x','f','k','s'}, //cap-shape
          { 'f','g','y','s'}, //cap-surface
          { 'n','b','c','g','r','p','u','e','w','y'}, //cap-color
          { 't','f'}, //bruises
          { 'a','l','c','y','f','m','n','p','s'}, //odor
          { 'a','d','f','n'}, //gill-attachment
          { 'c','w','d'}, //gill-spacing
          { 'b','n'}, //gill-size
          { 'k','n','b','h','g','r','o','p','u','e','w','y'}, //gill-color
          { 'e','t'}, //stalk-shape
          { 'b','c','u','e','z','r','?'}, //stalk-root
          { 'f','y','k','s'}, //stalk-surface-above-ring
          { 'f','y','k','s'}, //stalk-surface-below-ring
          { 'n','b','c','g','o','p','e','w','y'}, //stalk-color-above-ring
          { 'n','b','c','g','o','p','e','w','y'}, //stalk-color-below-ring
          { 'p','u'}, //veil-type
          { 'n','o','w','y'}, //veil-color
          { 'n','o','t'}, //ring-number
          { 'c','e','f','l','n','p','s','z'}, //ring-type
          { 'k','n','b','h','r','o','u','w','y'}, //spore-print-color
          { 'a','c','n','s','v','y'}, //population
          { 'g','l','m','p','u','w','d'}, //habitat
  };
  @Override
  public void process(Object[] objects) throws UDFException, IOException {
    List<Long> featuresEncoding = new ArrayList<>(126);
    for (int i = 0; i < objects.length; i++) {
      String value = (String)objects[i];
      char[] feature = features[i];
      for (char c : feature) {
        featuresEncoding.add(value.charAt(0) == c ? 1L : 0L);
      }
    }
    forward(featuresEncoding.toArray());
  }
}
#END CODE;
create table mushroom_classification_one_hot as
select t.*, label
from mushroom_classification
lateral view
one_hot(cap_shape,cap_surface,cap_color,bruises,odor,gill_attachment,
        gill_spacing, gill_size, gill_color, stalk_shape,stalk_root ,
        stalk_surface_above_ring,stalk_surface_below_ring,stalk_color_above_ring,
        stalk_color_below_ring,veil_type,veil_color,ring_number,ring_type,spore_print_color,
        population,habitat) t
AS f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,f16,f17,f18,f19,f20,
f21,f22,f23,f24,f25,f26,f27,f28,f29,f30,f31,f32,f33,f34,f35,f36,f37,f38,f39,f40,
f41,f42,f43,f44,f45,f46,f47,f48,f49,f50,f51,f52,f53,f54,f55,f56,f57,f58,f59,f60,
f61,f62,f63,f64,f65,f66,f67,f68,f69,f70,f71,f72,f73,f74,f75,f76,f77,f78,f79,f80,
f81,f82,f83,f84,f85,f86,f87,f88,f89,f90,f91,f92,f93,f94,f95,f96,f97,f98,f99,f100,
f101,f102,f103,f104,f105,f106,f107,f108,f109,f110,f111,f112,f113,f114,f115,f116,
f117,f118,f119,f120,f121,f122,f123,f124,f125,f126;
```

iii. 使用DataWorks的临时查询功能，新建MaxCompute ODPS SQL节点，执行SQL命令验证one-hot处理结果。

命令示例如下：

```
select * from mushroom_classification_one_hot;
```

返回结果如下：



iv. 使用DataWorks的临时查询功能，新建MaxCompute ODPS SQL节点，基于表mushroom_classification_one_hot中的数据创建训练数据集和测试数据集。

命令示例如下：

```
--训练数据集。1/4的数据用于模型训练。
create table mushroom_training as
select * from mushroom_classification_one_hot where sample(4,1);
--测试数据集。其余3/4的数据用于预测和评估。
create table mushroom_predict as
select * from mushroom_classification_one_hot except all select * from mushroom_training;
```

6. 创建机器学习模型并做预测。

i. 使用DataWorks的临时查询功能，新建MaxCompute ODPS SQL节点，基于训练数据集创建逻辑回归二分类模型lr_test_model。

命令示例如下：

```
create model lr_test_model
with properties('model_type'='logisticregression_binary', 'goodValue'='p','maxIter'='1000')
as select * from mushroom_training;
```

> **说明**
>
> - `properties` 中还可以指定更多参数，参数和机器学习PAI平台保持一致，请参见线性支持向量机。
> - SQL引擎会把 `as` 后的查询语句提取出来单独运行，结果存放在一个临时表中，您可以在作业的Logview的Summary信息中查看。临时表的生命周期为1天，超时会自动回收。
> - 如果后续需要删除模型，可以执行 `drop offlinemodel lr_test_model` 命令。

ii. 使用DataWorks的临时查询功能，新建MaxCompute ODPS SQL节点，基于模型lr_test_model，通过内置函数 `ml_predict` 对测试数据集中的数据进行预测。

命令示例如下：

```
create table mushroom_predict_result as
select * from ml_predict(
    lr_test_model,
    (select * from mushroom_predict)
);
```

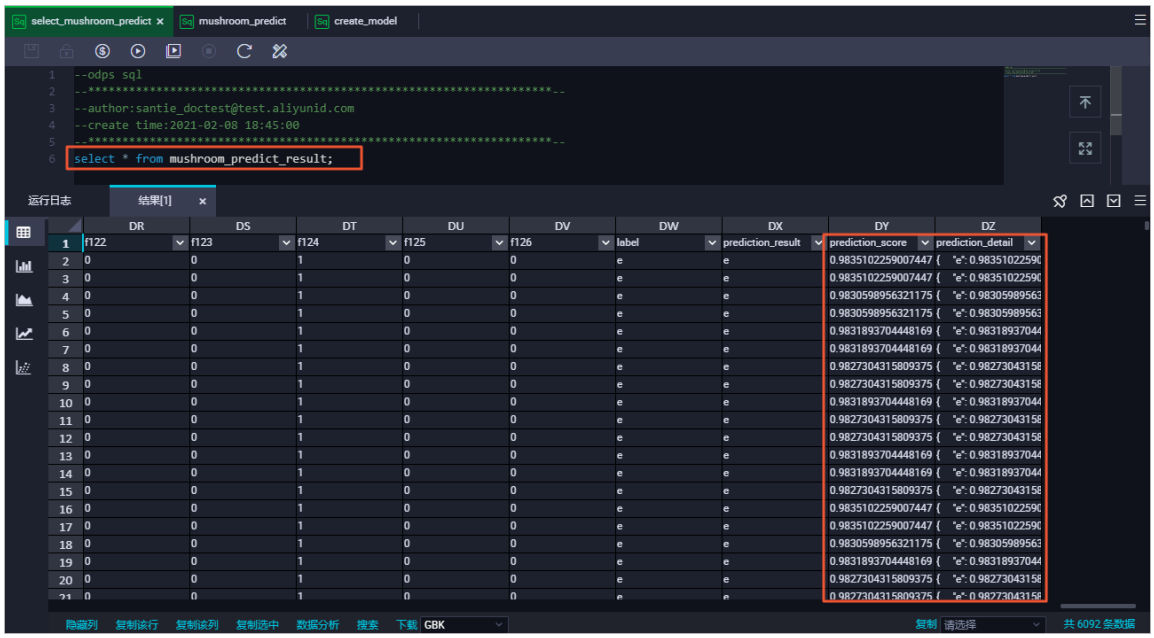
> **说明**
>
> - SQL引擎会把 `ml_predict` 函数下的子查询结果保存到临时表。临时表的生命周期为1天，超时会自动回收。
> - `ml_predict` 的结果可以直接放在SQL查询 `from` 子句中，也可以通过 `insert` 或 `create table as` 语句存到另一个表中。更多 `ml_predict` 信息，请参见支持的预测模型函数。

iii. 使用DataWorks的临时查询功能，新建MaxCompute ODPS SQL节点，执行SQL命令查看表mushroom_predict_result中的预测结果。

命令示例如下：

```
select * from mushroom_predict_result;
```

返回结果如下：



7. 通过内建函数 `ml_evaluate` 评估模型的预测准确度。

更多 `ml_evaluate` 信息，请参见支持的评估模型函数。

4.3. 计量计费

本文为您介绍如何估算MaxCompute SQLML作业的费用。

背景信息

通常情况下，每个算法组件由多个子任务组成。因此，计算费用时，需要先计算各算法组件下的子任务费用，再累计求和。MaxCompute SQLML作业使用了MaxCompute和机器学习PAI，所以计算费用包含这两个部分。

费用计算规则

MaxCompute SQLML底层的机器学习能力，是通过阿里云机器学习平台PAI产品独立计费的，所以MaxCompute SQLML费用分为以下两部分：

- SQL作业：计算费用和存储费用按照MaxCompute SQL计费规则计费，请参见计费项与计费方式概述。
- 机器学习模型：按照PAI Studio计费规则计费，规则如下，更多机器学习PAI计费信息，请参见PAI-Studio/Designer计费说明：

机器学习模型	单价（CNY/计算时）	MaxCompute支持的地域

机器学习模型	单价（CNY/计算时）	MaxCompute支持的地域
逻辑回归二分类	1.3	○ 华北2（北京） ○ 华东2（上海） ○ 华东1（杭州） ○ 华南1（深圳） ○ 中国（香港）
逻辑回归多分类		
线性回归		

说明 系统将1个小时消耗1个CPU Core和4 GB内存，折算为1个计算时。计算时数量的计算规则如下：

```
计算时数量=max(CPU Core数量×时长,内存×时长/4)
```

例如1小时消耗了2 个CPU Core和5 GB内存，则计算时数量为：

```
计算时数量=max(2×1,5×1/4)=2
```

账单金额为：

```
账单金额=计算时数量×单价
本例中账单金额为：
2（计算时）x1.3（单价金额） =2.6元
```

计费示例

基于MaxCompute SQLML费用计算规则，需要获取CPU Core和内存使用量，进而估算费用。

以快速入门中已有的预测数据集为例，估算费用的步骤如下：

1. 使用DataWorks的临时查询功能，基于预测数据集mushroom_predict创建逻辑回归二分类模型lr_test_model_1，并获取Logview。

   命令示例如下：

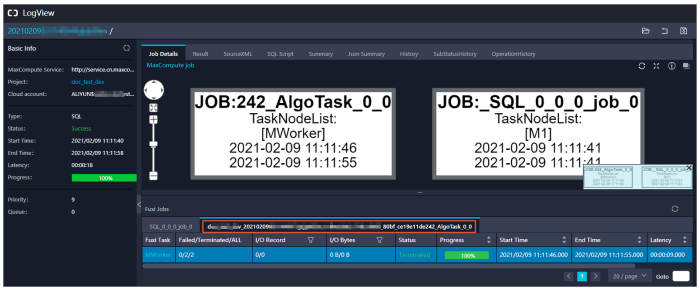
   ```
   create model lr_test_model_1
   with properties('model_type'='logisticregression_binary', 'goodValue'='p','maxIter'='1000')
   as select * from mushroom_predict;
   ```

   返回结果如下：

2. 基于Logview链接，获取AlgoTask作业的InstanceID。

   机器学习的作业在MaxCompute中被定义为AlgoTask。作业的命名规则是 `Projectname+'_'+InstanceID+"AlgoTask_x_x"` ，获取InstanceID。

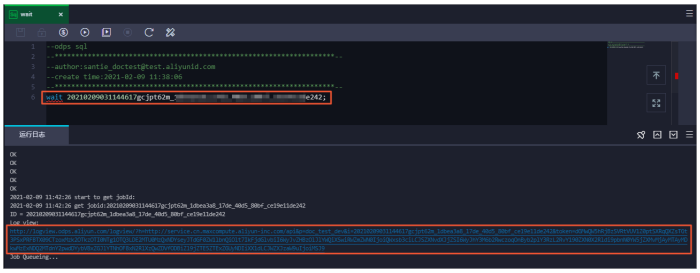
   

3. 使用DataWorks的临时查询功能，基于InstanceID获取机器学习作业的Logview。

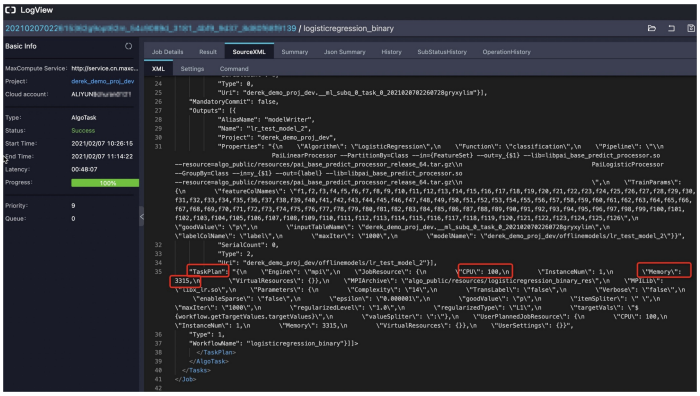
   命令示例如下：

   ```
   wait 20210209031144617gcjpt62m_1dbea3a8_17de_40d5_80bf_ce19e11d****;
   ```

   返回结果如下：



4. 基于机器学习作业的Logview，在**SourceXML**页签获取TaskPlan的CPU Core和内存使用量信息。



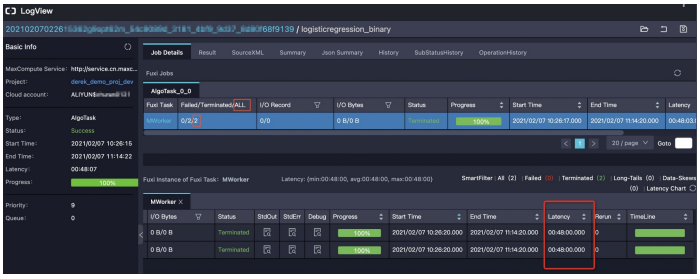
其中：

- CPU除以100表示使用的CPU Core数量，即该作业使用了1个CPU Core。
- Memory单位为MB，即使用了3.2 GB。

> **注意** 这是一个子任务使用的CPU Core和Memory，一个作业可能会有多个子任务，具体任务数请在下一步骤中获取。

5. 基于机器学习作业的Logview，在**Job Details**页签获取任务数和运行时长。

如图任务数为2，运行时长为48分钟。



综上：本例中使用CPU Core数是2x1=2，Memory为2x3.2 GB=6.4 GB，运行48分钟。

6. 预估总费用。
   i. 计算使用的计算时数。

   计算时=max(CPU Core**数量**×**时长**,**内存**×**时长**/4)=max[2×1×(48x60/3600),2×3.2×(48x60/3600)/4]≈1.6**计算时**

   ii. 计算机器学习作业费用。

   机器学习部分总费用=**子任务计算时数量**×**单价**≈1.6×1.3=2.08 CNY

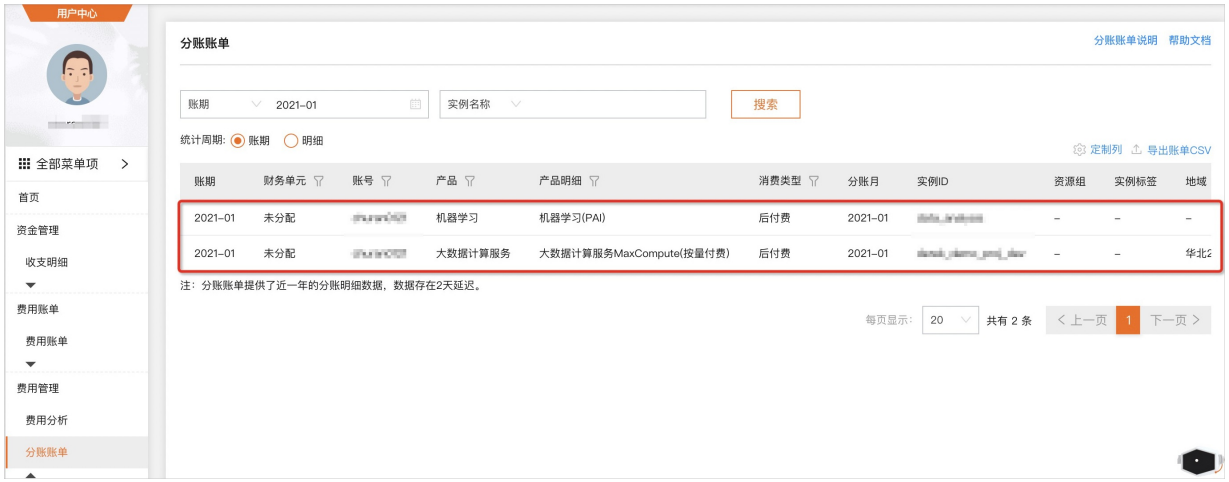
   即机器学习作业费用为2.08元。

   iii. 计算SQL作业费用，请参见计算费用（包年包月）或计算费用（按量计费）。

   iv. 机器学习作业费用与SQL作业费用相加即为MaxCompute SQLML作业费用。

查看账单信息

MaxCompute SQLML作业，在账单中体现的是大数据计算服务MaxCompute+机器学习（PAI）两个产品的费用。


用户中心
全部菜单项
首页
资金管理
收支明细
费用账单
费用账单
费用管理
费用分析
分账账单
分账账单
分账账单说明 帮助文档
账期 2021-01
实例名称
搜索
统计周期: 账期 明细
定制列 导出账单CSV
账期 财务单元 账号 产品 产品明细 消费类型 分账月 实例ID 资源组 实例标签 地域
2021-01 未分配 机器学习 机器学习(PAI) 后付费 2021-01
2021-01 未分配 大数据计算服务 大数据计算服务MaxCompute(按量付费) 后付费 2021-01
注：分账账单提供了近一年的分账明细数据，数据存在2天延迟。
每页显示: 20 共有 2 条
< 上一页 1 下一页 >

5.PyODPS

5.1. 快速入门

PyODPS是MaxCompute的Python版本的SDK，提供简单方便的Python编程接口。PyODPS支持类似Pandas的快速、灵活和富有表现力的数据结构。您可以通过PyODPS提供的DataFrame API使用Pandas的数据结果处理功能。本文用于帮助您快速开始使用PyODPS，并且能够用于实际项目。

前提条件

- 已开通MaxCompute服务，详情请参见开通MaxCompute和DataWorks。
- 已开通DataWorks服务，并创建工作空间，详情请参见创建MaxCompute项目。

背景信息

MaxCompute大部分开发都可以通过SQL语句实现，但对于复杂的业务场景以及自定义函数（UDF）都需要使用Python，例如：

- 接口对接

  MaxCompute表的每条数据，都需要通过身份证号码和外部HTTP的验证接口进行连通。

- 异步调用

  对于上千个处理逻辑相似的任务，如果对不同任务都单独创建一个节点，不利于管理，并且在同一时间会占用过多资源。利用PyODPS可以异步生成SQL执行任务队列，通过队列实现任务数的并发，方便统一管理所有节点。

- UDF开发

  当MaxCompute内置函数无法满足需求时，需要快速开发自定义函数（UDF）。例如，对DECIMAL类型进行千分位数字格式化展示。

- SQL性能提升

  您需要计算库存事务是否遵循先进先出的规则。由于只有一张表，每条数据都需要跟历史数据进行比对，而通常库存事务表体量巨大，时间复杂度是$O(N^2)$，往往无法运行并处理历史至今的结果。用Python结合Cache循环每条记录，只需遍历一次即可，时间复杂度是O(N)，可极大提升效率。

- 其他SQL很难解决的场景

  金额分摊模型。例如，10元由3个人均分，把尾差摊到其中1人身上。

更多PyODPS应用场景请参见：

- DataFrame操作：DataFrame快速入门。
- 读取分区表数据：PyODPS读取分区表数据。
- 参数传递：PyODPS参数传递。
- 使用第三方包：PyODPS使用第三方包。
- 查看一级分区：PyODPS查看一级分区。
- 条件查询：PyODPS条件查询。
- DataFrame Sequence及执行：PyODPS的Sequence及执行操作。

操作步骤

1. 新建PyODPS节点。

   为方便您快速开始，本文中使用DataWorks PyODPS节点进行开发，详情请参见创建PyODPS 2节点。

   > **说明**
   > - PyODPS节点底层的Python版本为2.7。
   > - PyODPS节点获取本地处理的数据量不能超过50 MB，节点运行时占用的内存不能超过1 GB，否则节点任务会被系统中止。因此请避免在PyODPS任务中写入数据量较大的Python处理代码。
   > - 在DataWorks上编写代码并进行调试效率较低，为提升运行效率，建议本地安装IDEA进行代码开发。

   i. 新建业务流程。

   进入数据开发页面，右键单击**业务流程**，选择**新建业务流程**。

   ii. 新建PyODPS节点。

   右键单击新建的业务流程，选择**新建 > MaxCompute > PyODPS 2**，输入节点名称，单击**提交**。

2. 编辑PyODPS节点。

i. 编写程序代码。

在PyODPS节点的编辑框中输入测试代码。以下是一个完整的使用PyODPS接口执行表操作的示例，更多关于表操作以及SQL操作的方法请参见表和SQL。

```
from odps import ODPS
import sys
reload (sys)
#修改系统默认编码。数据中存在中文字符时需要执行此操作。
sys.setdefaultencoding('utf8')
#以直接指定字段名以及字段类型的方式创建非分区表my_new_table。
#DataWorks的PyODPS节点中默认包含一个全局变量odps或者o，即为ODPS入口。您不需要手动定义ODPS入口，直接使用即可。更多信息，请参见在DataWorks上使用PyODPS。
table = o.create_table('my_new_table', 'num bigint, id string', if_not_exists=True)
#向非分区表my_new_table中插入数据。
records = [[111, 'aaa'],
           [222, 'bbb'],
           [333, 'ccc'],
           [444, '中文']]
o.write_table(table, records)
#读取非分区表my_new_table中的数据。
for record in o.read_table(table):
    print record[0],record[1]
#以运行SQL的方式读取表中的数据。
result = o.execute_sql('select * from my_new_table;',hints={'odps.sql.allow.fullscan': 'true'})
#读取SQL执行结果。
with result.open_reader() as reader:
    for record in reader:
        print record[0],record[1]
#删除表以清除资源。
table.drop()
```

ii. 运行代码。

完成编辑后，单击图标。运行结束后，您可以在下方的**运行日志**中看到运行结果。输出如下日志代表执行成功。



5.2. 安装指南及使用限制

PyODPS是MaxCompute的Python版本的SDK，提供对MaxCompute对象的基本操作及DataFrame框架，帮助您在MaxCompute上使用Python进行数据分析。本文为您介绍如何安装PyODPS及使用限制。

前提条件

安装PyOPDS环境前，您的Python环境需要满足以下条件：

- setuptools 3.0或以上版本。
- requests 2.4.0或以上版本。

安装步骤

1. 安装PIP工具。推荐使用Python包管理工具（PIP）安装PyODPS环境，详情请参见PIP安装说明安装PIP工具。参考安装命令如下。

```
pip install setuptools>=3.0
pip install requests>=2.4.0
pip install greenlet>=0.4.10  # 可选，安装后能加速Tunnel上传。
pip install cython>=0.19.0  # 可选，不建议Windows用户安装。
```

> 说明
> - 如果您想加快下载速度，推荐使用 阿里云镜像 。
> - 如果您的服务器是Windows系统，请确认您安装合适版本的Visual C++ 和Cython，否则无法实现Tunnel上传加速。

2. 执行如下命令安装PyODPS。

```
pip install pyodps
```

3. 执行如下命令检查安装是否成功。

```
python -c "from odps import ODPS"
```

4. 如果您使用的Python不是系统默认的Python版本，安装完PIP后，您可以执行如下命令进行Python版本切换。

```
/home/tops/bin/python2.7 -m pip install setuptools>=3.0
```

使用限制

- SQL使用限制项。
- DataWorks PyODPS节点使用限制。
- 由于沙箱的限制，部分Pandas计算后端执行本地调试通过的程序，无法在MaxCompute上调试通过。

后续步骤

建议您安装以下工具，提升Tunnel上传的速度：

- greenlet 0.4.10或以上版本。
- cython 0.19.0或以上版本。

5.3. 工具平台使用指南

5.3.1. 工具平台使用概述

PyODPS可在DataWorks等数据开发平台中作为数据开发节点调用。这些平台提供了PyODPS运行环境和调度执行的能力，无需您手动创建ODPS入口对象。

如果您想从上述平台迁移到自行部署的PyODPS环境，请仔细阅读下列章节：

- 从平台到自行部署
- DataWorks用户使用指南

5.3.2. 从平台到自行部署

如果您需要在本地调试PyODPS，或平台的资源不能满足您的要求，此时您可以从平台迁移到自己部署的PyODPS环境中。

自行部署的方法如下：

1. 在本地环境中安装PyODPS，详细步骤请参见安装指南及使用限制。
2. 在本地环境中手动创建MaxCompute入口对象。

   您可以在之前的平台中执行如下语句生成创建MaxCompute入口对象所需要的语句模板，然后手动修改为可用的代码。

   ```
   print("\nfrom odps import ODPS\no = ODPS(%r, '<access-key>', %r, '<endpoint>')\n" % (o.account.access_id, o.project))
   ```

   命令说明如下：

   - <access-key>：账号的AccessKey。
   - <endpoint>：MaxCompute服务的Endpoint，详情请参见Endpoint。
   - 项目名称：使用上述命令无需指定该项目名称。

     当您从DataWorks平台迁移至本地时，如果手动填写该参数（而非使用上述语句自动生成），请注意DataWorks中的项目名称（下称<DataWorks Project Name>）与实际的MaxCompute项目名称并不是同一个概念。在DataWorks中，<DataWorks Project Name>_dev 表示DataWorks开发环境中作业运行的MaxCompute项目名称，而 <DataWorks Project Name> 表示DataWorks生产环境中作业运行的MaxCompute项目名称。详情请参见简单模式和标准模式的区别。
3. 修改完成后，将上述代码放置到所有代码的最顶端即可。

5.3.3. 在DataWorks上使用PyODPS

本文为您介绍如何在DataWorks上使用PyODPS。

功能入口

您可以进入DataWorks的**数据开发**页面创建PyODPS节点。

PyODPS节点分为PyODPS 2和PyODPS 3两种，二者的区别在于底层Python语言版本不同。PyODPS 2底层Python语言版本为Python 2，PyODPS 3底层Python语言版本为Python 3，请您根据实际使用的Python语言版本创建PyODPS节点。

更多创建PyODPS节点信息，请参见创建PyODPS 2节点和创建PyODPS 3节点。

使用示例，请参见PyODPS节点实现结巴中文分词。



使用限制

- PyODPS节点获取本地处理的数据不能超过50 MB，节点运行时占用内存不能超过1 GB，否则节点任务会被系统中止。请避免在PyODPS任务中写额外的Python数据处理代码。
- 在DataWorks上编写代码并进行调试效率较低，为提升运行效率，建议本地安装IDE进行代码开发。
- 在DataWorks上使用PyODPS时，为了防止对DataWorks的Gate Way造成压力，对内存和CPU都有限制，该限制由DataWorks统一管理。如果您发现有**Got killed**报错，即表明内存使用超限，进程被中止。因此，请尽量避免本地的数据操作。通过PyODPS发起的SQL和DataFrame任务（除to_pandas外）不受此限制。
- 由于缺少matplotlib等包，如下功能可能受限：
  - DataFrame的plot函数。
  - DataFrame自定义函数需要提交到MaxCompute执行。由于Python沙箱限制，第三方库只支持所有的纯粹Python库以及Numpy，因此不能直接使用Pandas。
  - DataWorks中执行的非自定义函数代码可以使用平台预装的Numpy和Pandas。不支持其他带有二进制代码的三方包。
- 由于兼容性原因，在DataWorks中，options.tunnel.use_instance_tunnel默认设置为False。如果需要全局开启instance tunnel，需要手动将该值设置为True。
- 由于实现的原因，Python的atexit包不被支持，请使用try-finally结构实现相关功能。

ODPS入口

DataWorks的PyODPS节点中，将会包含一个全局变量 `odps` 或者 `o` ，即为ODPS入口。您不需要手动定义ODPS入口。

```
print(o.exist_table('pyodps_iris'))
```

执行SQL

您可以在PyODPS节点中执行SQL，详情请参见SQL。

Dataworks上默认未开启instance tunnel，即instance.open_reader默认使用Result接口（最多一万条记录）。您可以通过reader.count获取记录数。如果您需要迭代获取全部数据，则需要关闭 `limit` 限制。您可以通过下列语句在全局范围内打开Instance Tunnel并关闭 `limit` 限制。

```
options.tunnel.use_instance_tunnel = True
options.tunnel.limit_instance_tunnel = False  # 关闭limit限制，读取全部数据。
with instance.open_reader() as reader:
    # 通过Instance Tunnel可读取全部数据。
```

您也可以通过在open_reader上添加 `tunnel=True` ，实现仅对本次open_reader开启instance tunnel。同时，您还可以添加 `limit=False` ，实现仅对本次关闭 `limit` 限制。

```
with instance.open_reader(tunnel=True, limit=False) as reader:
# 本次open_reader使用Instance Tunnel接口，且能读取全部数据。
```

> **说明** 若您未开启Instance Tunnel，可能导致获取数据格式错误，解决方法请参见Python SDK常见问题。

DataFrame

- 执行

  在DataWorks的环境里，DataFrame的执行需要显式调用立即执行的方法。

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
for record in iris[iris.sepal_width < 3].execute():  # 调用立即执行的方法，处理每条Record。
```

如果您需要在Print时调用立即执行，需要开启 `options.interactive` 。

```
from odps import options
from odps.df import DataFrame
options.interactive = True  # 在开始处打开开关。
iris = DataFrame(o.get_table('pyodps_iris'))
print(iris.sepal_width.sum())  # Print时会立即执行。
```

- 打印详细信息

  通过设置 `options.verbose` 选项。在DataWorks上，默认已经处于打开状态，运行过程会打印Logview等详细过程。

获取调度参数

与DataWorks中的SQL节点不同，为了避免影响代码，PyODPS节点不会在代码中替换 ${param_name}这样的字符串，而是在执行代码前，在全局变量中增加一个名为 `args` 的dict，调度参数可以在此获取。例如，在**节点基本属性 > 参数**中设置 `ds=${yyyymmdd}` ，则可以通过以下方式在代码中获取该参数。

```
print('ds=' + args['ds'])
ds=20161116
```

> **说明** 如果您需要获取名为 `ds=${yyyymmdd}` 的分区，则可以使用如下方法。
>
> ```
> o.get_table('table_name').get_partition('ds=' + args['ds'])
> ```

5.4. 基本操作

5.4.1. 基本操作概述

PyODPS提供直接针对ODPS对象的基本操作接口，您可以使用符合Python习惯的编程方式操作MaxCompute。

PyODPS的基本操作主要包括以下几个部分：

- 项目空间
- 表
- SQL
- 任务实例
- 资源
- 函数

5.4.2. 项目空间

本文为您介绍如何使用PyODPS对项目空间进行基本操作。

项目空间是MaxCompute的基本组织单元，更多详情请参见项目。

项目空间的基本操作如下：

- 获取项目空间：使用MaxCompute入口对象的 `get_project()` 方法获取项目空间。

  ```
  project = o.get_project('my_project')  # 指定项目名称时，获取特定项目。
  project = o.get_project()              # 不指定项目名称时，获取当前项目。
  ```

  该方法需要提供project_name即项目空间名称。指定项目空间名称时，则获取特定项目；如果不指定项目空间名称，则获取当前项目空间名称。

- 验证项目空间是否存在：使用 `exist_project()` 方法，检验某个项目空间是否存在。

5.4.3. 表

PyODPS支持对MaxCompute表的基本操作，包括创建表、创建表的Schema、同步表更新、获取表数据、删除表、表分区操作以及如何将表转换为DataFrame对象。

背景信息

PyODPS提供对MaxCompute表的基本操作方法。

操作	说明
基本操作	列出项目空间下的所有表、判断表是否存在、获取表等基本操作。
创建表	使用PyODPS创建表。
创建表的Schema	使用PyODPS创建表的Schema。
同步表更新	使用PyODPS同步表更新。
向表中插入一行记录	使用PyODPS向表中插入一行记录。
写入数据	使用PyODPS向表中写入数据。
获取表数据	使用PyODPS获取表中数据。
删除表	使用PyODPS删除表。
转换表为DataFrame	使用PyODPS转换表为DataFrame。
表分区	使用PyODPS判断是否为分区表、遍历表全部分区、判断分区是否存在、创建分区等。
数据上传下载通道	使用PyODPS操作Tunnel向MaxCompute中上传或者下载数据。

> **说明** 更多PyODPS方法说明，请参见Python SDK方法说明。

基本操作

- 列出项目空间下的所有表：入口对象的 `list_tables()` 方法可以列出项目空间下的所有表。

```
#列出项目空间下的所有表。
for table in o.list_tables():
```

- 判断表是否存在：入口对象的 `exist_table()` 方法可以判断表是否存在。
- 获取表：入口对象的 `get_table()` 方法可以获取表。

```
t = o.get_table('table_name')
t.schema
odps.Schema {
  c_int_a                 bigint
  c_int_b                 bigint
  c_double_a              double
  c_double_b              double
  c_string_a              string
  c_string_b              string
  c_bool_a                boolean
  c_bool_b                boolean
  c_datetime_a            datetime
  c_datetime_b            datetime
}
t.lifecycle
-1
print(t.creation_time)
2014-05-15 14:58:43
t.is_virtual_view
False
t.size
1408
t.comment
'table_name Table Comment'
t.schema.columns
[<column c_int_a, type bigint>,
 <column c_int_b, type bigint>,
 <column c_double_a, type double>,
 <column c_double_b, type double>,
 <column c_string_a, type string>,
 <column c_string_b, type string>,
 <column c_bool_a, type boolean>,
 <column c_bool_b, type boolean>,
 <column c_datetime_a, type datetime>,
 <column c_datetime_b, type datetime>]
t.schema['c_int_a']
<column c_int_a, type bigint>
t.schema['c_int_a'].comment
'Comment of column c_int_a'
```

您可以通过 `project` 参数，跨项目获取表。

```
t = o.get_table('table_name', project='other_project')
```

> **说明** 表是MaxCompute的数据存储单元，详情请参见表。

创建表的Schema

初始化方法有如下两种：

- 通过表的列以及可选的分区进行初始化。

```
from odps.models import Schema, Column, Partition
columns = [Column(name='num', type='bigint', comment='the column'),
           Column(name='num2', type='double', comment='the column2')]
partitions = [Partition(name='pt', type='string', comment='the partition')]
schema = Schema(columns=columns, partitions=partitions)
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
schema.partitions
[<partition pt, type string>]
schema.names #获取非分区字段的字段名。
['num', 'num2']
schema.types #获取非分区字段的字段类型。
[bigint, double]
```

- 使用 `Schema.from_lists()` 方法。该方法更容易调用，但无法直接设置列和分区的注释。

```
schema = Schema.from_lists(['num', 'num2'], ['bigint', 'double'], ['pt'], ['string'])
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
```

创建表

使用 `create_table()` 方法创建表的方式有如下两种：

- 使用表Schema创建表，方法如下。

```
table = o.create_table('my_new_table', schema)
#只有不存在表时，才创建表。
table = o.create_table('my_new_table', schema, if_not_exists=True)
#设置生命周期。
table = o.create_table('my_new_table', schema, lifecycle=7)
```

- 采用字段名及字段类型字符串创建表，方法如下。

```
#创建非分区表。
table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
#创建分区表可传入（表字段列表，分区字段列表）。
table = o.create_table('my_new_table', ('num bigint, num2 double', 'pt string'), if_not_exists=True)
```

未打开新数据类型开关时，创建表的数据类型只允许为BIGINT、DOUBLE、DECIMAL、STRING、DATETIME、BOOLEAN、MAP和ARRAY类型。如果您需要支持TINYINT和STRUCT等新数据类型，可以打开 `options.sql.use_odps2_extension = True` 开关，示例如下。

```
from odps import options
options.sql.use_odps2_extension = True
table = o.create_table('my_new_table', 'cat smallint, content struct<title:varchar(100), body:string>')
```

同步表更新

当一个表被其他程序更新，例如改变了Schema，可以调用 `reload()` 方法同步表的更新。

```
table.reload()
```

向表中插入一行记录

Record表示表的一行记录，对表对象调用 `new_record()` 方法即可创建一个新的Record。

```
t = o.get_table('mytable')
r = t.new_record(['val0', 'val1'])  # 值的个数必须等于表Schema的字段数。
r2 = t.new_record()      # 可以不传入值。
r2[0] = 'val0' # 通过偏移设置值。
r2['field1'] = 'val1'  # 通过字段名设置值。
r2.field1 = 'val1'  # 通过属性设置值。
print(record[0])  # 取第0个位置的值。
print(record['c_double_a'])  # 通过字段取值。
print(record.c_double_a)  # 通过属性取值。
print(record[0: 3])  # 切片操作。
print(record[0, 2, 3])  # 取多个位置的值。
print(record['c_int_a', 'c_double_a'])  # 通过多个字段取值。
```

写入数据

- 使用入口对象的 `write_table()` 方法写入数据。

```
records = [[111, 'aaa', True],                # 此处可以是list。
           [222, 'bbb', False],
           [333, 'ccc', True],
           [444, '中文', False]]
o.write_table('test_table', records, partition='pt=test', create_partition=True)
```

> 说明
>
> - 每次调用 `write_table()` 方法，MaxCompute都会在服务端生成一个文件。该操作耗时较长，同时文件过多会降低后续的查询效率。因此，建议您在使用此方法时，一次性写入多组数据，或者传入一个生成器对象。
> - 调用 `write_table()` 方法向表中写入数据时会追加到原有数据中。PyODPS不提供覆盖数据的选项，如果需要覆盖数据，请手动清除原有数据。对于非分区表，需要调用 `table.truncate()` 方法；对于分区表，需要删除分区后再建立新的分区。

- 对表对象调用 `open_writer()` 方法写入数据。

```
with t.open_writer(partition='pt=test') as writer:
records = [[111, 'aaa', True],                #此处可以是List。
          [222, 'bbb', False],
          [333, 'ccc', True],
          [444, '中文', False]]
writer.write(records)  #这里Records可以是可迭代对象。
with t.open_writer(partition='pt1=test1,pt2=test2') as writer: #多级分区写法。
records = [t.new_record([111, 'aaa', True]),   #也可以是Record对象。
           t.new_record([222, 'bbb', False]),
           t.new_record([333, 'ccc', True]),
           t.new_record([444, '中文', False])]
writer.write(records)
```

如果分区不存在，可以使用create_partition参数指定创建分区，如下所示。

```
with t.open_writer(partition='pt=test', create_partition=True) as writer:
     records = [[111, 'aaa', True],                # 这里可以是List。
                [222, 'bbb', False],
                [333, 'ccc', True],
                [444, '中文', False]]
   writer.write(records)  # 此处的Records可以是可迭代对象。
```

- 使用多进程并行写数据。

  每个进程写数据时共享同一个Session_ID，但是有不同的Block_ID。每个Block对应服务端的一个文件。主进程执行Commit，完成数据上传。

```
import random
from multiprocessing import Pool
from odps.tunnel import TableTunnel
def write_records(tunnel, table, session_id, block_id):
    # 对使用指定的ID创建Session。
    local_session = tunnel.create_upload_session(table.name, upload_id=session_id)
    # 创建Writer时指定Block_ID。
    with local_session.open_record_writer(block_id) as writer:
        for i in range(5):
            # 生成数据并写入对应Block。
            record = table.new_record([random.randint(1, 100), random.random()])
            writer.write(record)
if __name__ == '__main__':
    N_WORKERS = 3
    table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
    tunnel = TableTunnel(o)
    upload_session = tunnel.create_upload_session(table.name)
    # 每个进程使用同一个Session_ID。
    session_id = upload_session.id
    pool = Pool(processes=N_WORKERS)
    futures = []
    block_ids = []
    for i in range(N_WORKERS):
        futures.append(pool.apply_async(write_records, (tunnel, table, session_id, i)))
        block_ids.append(i)
    [f.get() for f in futures]
    # 最后执行Commit，并指定所有Block。
    upload_session.commit(block_ids)
```

获取表数据

获取表数据的方法有多种，常用方法如下：

- 使用入口对象的 `read_table()` 方法。

```
# 处理一条记录。
for record in o.read_table('test_table', partition='pt=test'):
```

- 如果您仅需要查看每个表最开始的小于1万条数据，可以对表对象调用 `head()` 方法。

```
t = o.get_table('table_name')
# 处理每个Record对象。
for record in t.head(3):
```

- 调用 `open_reader()` 方法读取数据。

- 使用 `with` 表达式的写法如下。

```
with t.open_reader(partition='pt=test') as reader:
count = reader.count
for record in reader[5:10]  # 可以执行多次，直到将Count数量的Record读完，此处可以改造成并行操作。
# 处理一条记录。
```

- 不使用 `with` 表达式的写法如下。

```
reader = t.open_reader(partition='pt=test')
count = reader.count
for record in reader[5:10]  # 可以执行多次，直到将Count数量的Record读完，此处可以改造成并行操作。
# 处理一条记录。
```

删除表

使用 `delete_table()` 方法删除已经存在的表。

```
o.delete_table('my_table_name', if_exists=True)  # 只有表存在时，才删除表。
 t.drop()  # Table对象存在时，直接调用Drop方法删除。
```

转换表为DataFrame

PyODPS提供了DataFrame框架，支持以更方便的方式查询和操作MaxCompute数据。使用 `to_df()` 方法，即可转化为DataFrame对象。

```
table = o.get_table('my_table_name')
df = table.to_df()
```

表分区

- 判断是否为分区表。

```
if table.schema.partitions:
 print('Table %s is partitioned.' % table.name)
```

- 遍历表全部分区。

```
for partition in table.partitions:
    print(partition.name)
for partition in table.iterate_partitions(spec='pt=test'): # 遍历二级分区。
```

- 判断分区是否存在。

```
table.exist_partition('pt=test,sub=2015')
```

- 获取分区。

```
partition = table.get_partition('pt=test')
print(partition.creation_time)
partition.size
```

- 创建分区。

```
t.create_partition('pt=test', if_not_exists=True)  # 指定if_not_exists参数，分区不存在时才创建分区。
```

- 删除分区。

```
t.delete_partition('pt=test', if_exists=True)  # 指定if_exists参数，分区存在时才删除分区。
partition.drop()  # 分区对象存在时，直接对分区对象调用Drop方法删除。
```

数据上传下载通道

Tunnel是MaxCompute的数据通道，用户可以通过Tunnel向MaxCompute中上传或者下载数据。

- 上传数据示例

```
from odps.tunnel import TableTunnel
table = o.get_table('my_table')
tunnel = TableTunnel(odps)
upload_session = tunnel.create_upload_session(table.name, partition_spec='pt=test')
with upload_session.open_record_writer(0) as writer:
    record = table.new_record()
    record[0] = 'test1'
    record[1] = 'id1'
    writer.write(record)
    record = table.new_record(['test2', 'id2'])
    writer.write(record)
upload_session.commit([0])
```

- 下载数据示例

```
from odps.tunnel import TableTunnel
tunnel = TableTunnel(odps)
download_session = tunnel.create_download_session('my_table', partition_spec='pt=test')
# 处理每条记录。
with download_session.open_record_reader(0, download_session.count) as reader:
    for record in reader:
```

> 说明
> - PyODPS不支持上传外部表，例如OSS和OTS的表。
> - 不推荐直接使用Tunnel接口，推荐您直接使用表对象的写和读接口。
> - 如果您安装了CPython，在安装PyODPS时会编译C代码，加速Tunnel的上传和下载。

5.4.4. SQL

PyODPS支持对MaxCompute SQL的基本操作，本文为您介绍如何在PyODPS中使用SQL。

背景信息

PyODPS提供对MaxCompute SQL的基本操作方法，方法如下所示。

方法名称	方法说明
execute_sql()/run_sql()	执行SQL语句。
open_reader()	读取SQL执行结果。

> 说明 在MaxCompute客户端中可以执行的SQL语句并非都可以通过入口对象的 `execute_sql()` 和 `run_sql()` 方法执行。在调用非DDL或非DML语句时，请使用其它方法。例如，调用GRANT或REVOKE语句时，请使用 `run_security_query` 方法；调用API命令时，请使用 `run_xflow` 或 `execute_xflow` 方法。

在IPython和Jupyter里也支持使用SQL插件的方式运行SQL和参数化查询，详情请参见IPython增强和Jupyter Notebook增强。

在Python UDF编写过程中，如果某个UDF引用的资源是动态变化的，您可以在execute_sql()中设置alias给旧的资源一个别名作为新的资源，无需重新删除或创建新的UDF。详情请参见设置alias。

执行SQL语句

PyODPS对MaxCompute SQL操作的具体说明如下。

- 参数说明
  - statement：需要执行的SQL语句。
  - hints：设置运行时参数，参数类型是DICT。
- 返回值说明

  执行 `execute_sql()` 和 `run_sql()` 后的返回值是任务实例。详情请参见任务实例。
- 使用示例
  - 示例1

    执行SQL语句。

```
o.execute_sql('select * from table_name')  #同步的方式执行，会阻塞直到SQL语句执行完成。
instance = o.run_sql('select * from table_name')  #异步的方式执行。
print(instance.get_logview_address())  # 获取Logview地址。
instance.wait_for_success()  # 阻塞直到完成。
```

- 示例2

  执行SQL语句时，运行参数。

  ```
  o.execute_sql('select * from pyodps_iris', hints={'odps.sql.mapper.split.size': 16})
  ```

  您也可以通过如下示例，设置 `sql.settings` ，对运行的参数进行全局配置，则在每次运行语句时都会执行对应参数。

  ```
  from odps import options
  options.sql.settings = {'odps.sql.mapper.split.size': 16}
  o.execute_sql('select * from pyodps_iris')  # 会根据全局配置添加hints。
  ```

读取SQL执行结果

您可以通过 `open_reader` 操作读取SQL执行结果。有以下两种情况：

- 读取表数据，返回结构化数据，通过 `for` 语句遍历即可。

  ```
  with o.execute_sql('select * from table_name').open_reader() as reader:
      for record in reader:
      # 处理每一个record。
  ```

- 执行 `desc` 等命令，返回非结构化数据，需要通过 `reader.raw` 获取执行结果。

  ```
  with o.execute_sql('desc table_name').open_reader() as reader:
      print(reader.raw)
  ```

在调用 `open_reader()` 时，PyODPS会默认调用旧的Result接口，可能会出现获取数据超时或获取数据受限等问题。您可以按照如下方法指定PyODPS调用Instance Tunnel。

- 在脚本中设置 `options.tunnel.use_instance_tunnel =True` 。
- 按照如下示例，设置 `open_reader(tunnel=True)` 。从PyODPS v0.7.7.1开始，您可以通过 `open_reader()` 方法读取全量数据。

  ```
  with o.execute_sql('select * from table_name').open_reader(tunnel=True) as reader:
      for record in reader:
          print(record)
  ```

> 说明
>
> - 如果您使用了较低版本的MaxCompute服务，或者调用Instance Tunnel出现了问题，PyODPS会给出警告并自动降级到旧的Result接口，您可根据警告信息判断导致降级的原因。
> - 如果您使用的MaxCompute只能支持旧Result接口，并且需要读取所有的数据，您可将SQL结果写入另一张表后用读表接口读取（可能受到Project安全设置的限制）。
> - 更多Instance Tunnel说明，请参见Instance tunnel。

PyODPS对于受保护的项目，通过Tunnel下载数据将受限。您可以通过以下方法，将数据下载限制打开，可下载的数据条数最多被限制为10000条。

- 在脚本中设置 `options.tunnel.limit_instance_tunnel = True` ，限制下载数据的规模。
- 在 `open_reader()` 方法设置limit参数，限制下载数据的规模。

  例如 `open_reader(tunnel=True, limit=1000)` 表示限制下载1000行。

设置alias

如果某个UDF引用的资源是动态变化的，您可以通过设置alias给旧的资源一个别名作为新的资源，无需重新删除或创建新的UDF。

```
from odps.models import Schema
myfunc = '''\
from odps.udf import annotate
from odps.distcache import get_cache_file
@annotate('bigint->bigint')
class Example(object):
    def __init__(self):
        self.n = int(get_cache_file('test_alias_res1').read())
    def evaluate(self, arg):
        return arg + self.n
'''
res1 = o.create_resource('test_alias_res1', 'file', file_obj='1')
o.create_resource('test_alias.py', 'py', file_obj=myfunc)
o.create_function('test_alias_func',
                  class_type='test_alias.Example',
                  resources=['test_alias.py', 'test_alias_res1'])
table = o.create_table(
    'test_table',
    schema=Schema.from_lists(['size'], ['bigint']),
    if_not_exists=True
)
data = [[1, ], ]
# 写入一行数据，只包含一个值1。
o.write_table(table, 0, [table.new_record(it) for it in data])
with o.execute_sql(
    'select test_alias_func(size) from test_table').open_reader() as reader:
    print(reader[0][0])
res2 = o.create_resource('test_alias_res2', 'file', file_obj='2')
# 把内容为1的资源别名设置成内容为2的资源。您不需要修改UDF或资源。
with o.execute_sql(
    'select test_alias_func(size) from test_table',
    aliases={'test_alias_res1': 'test_alias_res2'}).open_reader() as reader:
    print(reader[0][0])
```

在少数情形下，提交SQL时需要同时提交biz_id，否则执行会报错。此时，您可以通过设置全局options里的biz_id解决此类报错。

```
from odps import options
options.biz_id = 'my_biz_id'
o.execute_sql('select * from pyodps_iris')
```

5.4.5. 任务实例

Task是MaxCompute的基本计算单元，例如SQL Task。本文为您介绍PyODPS中任务实例的基本操作。

基本操作

Task在执行时会被实例化， 以MaxCompute实例（下文简称为实例或Instance）的形式存在。实例常见的操作如下：

- `list_instances()`：获取项目空间下的所有的Instance。
- `exist_instance()`：判断某实例是否存在。
- `get_instance()`：获取实例。
- `stop_instance()`：停止实例。

> **说明** 通过Instance对象调用 `stop` 方法也可以用于停止Instance。

示例

```
for instance in o.list_instances():
    print(instance.id)
    o.exist_instance('my_instance_id')
```

获取Logview地址

- 对于SQL等任务，通过Instance对象调用 `get_logview_address` 方法即可获取Logview地址。

```
#从已有的instance对象获取Logview地址。
instance = o.run_sql('desc pyodps_iris')
print(instance.get_logview_address())
#从instance id获取Logview地址。
instance = o.get_instance('20160426055209459g9k5pvyi2')
print(instance.get_logview_address())
```

- 对于XFlow任务，需要先枚举其子任务，再获取子任务的Logview。

```
instance = o.run_xflow('AppendID', 'algo_public', {'inputTableName': 'input_table', 'outputTableName': 'output_table'})
for sub_inst_name, sub_inst in o.get_xflow_sub_instances(instance).items():
    print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address()))
```

任务实例状态

一个Instance的状态可以是`Running`、`Suspended`或者`Terminated`。关于任务实例状态的方法有以下几种：

- `status`：属性来获取状态。
- `is_terminated`：返回当前Instance是否已经执行完毕的结果。
- `is_successful`：返回当前Instance是否正确地执行完毕的结果。任务处于运行中或者执行失败都会返回False。

```
instance = o.get_instance('20160426055209459g9k5pvyi2')
instance.status
<Status.TERMINATED: 'Terminated'>
    from odps.models import Instance
    instance.status == Instance.Status.TERMINATED
    True
    instance.status.value
    'Terminated'
```

> **说明** 调用`wait_for_completion`方法会阻塞直到Instance执行完成。`wait_for_success`方法同样会阻塞，但如果最终任务执行失败，会抛出相关异常。

子任务操作

一个Instance在运行时，可能包含一个或者多个子任务，这些子任务称为Task（注意：这里的Task不同于MaxCompute的计算单元）。关于Task的方法有以下几种：

- `get_task_names`：通过此方法获取所有的Task任务。该方法返回一个所有子任务的名称列表。

```
instance.get_task_names()
['SQLDropTableTask']
```

- `get_task_result`：指定Task名称，获取该Task的执行结果。该方法是以字典的形式返回每个Task的执行结果。

```
instance = o.execute_sql('select * from pyodps_iris limit 1')
instance.get_task_names()
['AnonymousSQLTask']
instance.get_task_result('AnonymousSQLTask')
'"sepallength","sepalwidth","petallength","petalwidth","name"\n5.1,3.5,1.4,0.2,"Iris-setosa"\n'
instance.get_task_results()
OrderedDict([('AnonymousSQLTask', "sepallength","sepalwidth","petallength","petalwidth","name"\n5.1,3.5,1.4,0.2,"Iris-setosa"\
n')])
```

- `get_task_progress`：该方法获取在任务实例运行时Task当前的运行进度。

```
while not instance.is_terminated():
    for task_name in instance.get_task_names():
        print(instance.id, instance.get_task_progress(task_name).get_stage_progress_formatted_string())
        time.sleep(10)
20160519101349613gzbzufck2 2016-05-19 18:14:03 M1_Stg1_job0:0/1/1[100%]
```

5.4.6. 资源

PyODPS主要支持两种资源类型，即文件和表。本文为您介绍PyODPS中资源的常见操作。

基本操作

- `list_resources()`：列出该项目空间下的所有资源。
- `exist_resource()`：判断资源是否存在。
- `delete_resource()`：删除资源。也可以通过Resource对象调用`drop`方法实现。
- `create_resource()`：创建资源。
- `open_resource()`：读取资源。

文件资源

文件资源包括基础的*File*类型，以及*Py*、*Jar*、*Archive*类型。

- 创建文件资源

创建文件资源可以通过给定资源名、文件类型和一个file-like的对象（或字符串对象）调用 `create_resource()` 来创建。

```
resource = o.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file'))  # 使用file-like的对象创建文件资源。
resource = o.create_resource('test_py_resource', 'py', file_obj='import this')  # 使用字符串创建文件资源。
```

- 读取和修改文件资源

  打开一个资源有如下两种方法：

  - 对文件资源调用 `open` 方法。
  - 在ODPS入口调用 `open_resource()` 方法。

  打开后的对象是file-like的对象。 类似于Python内置的 `open()` 方法，文件资源也支持打开的模式，示例如下。

```
with resource.open('r') as fp:  # 以读模式打开资源。
     content = fp.read()  # 读取全部的内容。
     fp.seek(0)  # 回到资源开头。
     lines = fp.readlines()  # 读成多行。
     fp.write('Hello World')  # 报错，读模式下无法写资源。
 with o.open_resource('test_file_resource', mode='r+') as fp:  # 读写模式打开资源。
     fp.read()
     fp.tell()  # 定位当前位置。
     fp.seek(10)
     fp.truncate()  # 截断后面的内容。
     fp.writelines(['Hello\n', 'World\n'])  # 写入多行。
     fp.write('Hello World')
     fp.flush()  # 手动调用会将更新提交到ODPS。
```

  所有支持的打开类型包括：

  - `r` ：读模式，只能打开不能写。
  - `w` ：写模式，只能写入而不能读文件，注意用写模式打开，文件内容会先被清空。
  - `a` ：追加模式，只能写入内容到文件末尾。
  - `r+` ：读写模式，可以任意读写内容。
  - `w+` ：类似于 `r+` ，但会先清空文件内容。
  - `a+` ：类似于 `r+` ，但写入时只能写入文件末尾。

  同时，PyODPS中文件资源支持以二进制模式打开，例如一些压缩文件需要以这种模式打开。

  - `rb` ：指以二进制读模式打开文件。
  - `r+b` ：指以二进制读写模式打开。

表资源

- 创建表资源

```
o.create_resource('test_table_resource', 'table', table_name='my_table', partition='pt=test')
```

- 更新表资源

```
table_resource = o.get_resource('test_table_resource')
table_resource.update(partition='pt=test2', project_name='my_project2')
```

- 获取表及分区

```
table_resource = o.get_resource('test_table_resource')
table = table_resource.table
print(table.name)
partition = table_resource.partition
print(partition.spec)
```

- 读写内容

```
table_resource = o.get_resource('test_table_resource')
with table_resource.open_writer() as writer:
    writer.write([0, 'aaaa'])
    writer.write([1, 'bbbbb'])
    with table_resource.open_reader() as reader:
        for rec in reader:
            print(rec)
```

5.4.7. 函数

您可以编写自定义函数并在MaxCompute SQL中使用它们。

基本操作

- `list_functions()`：获取项目空间下的所有函数。
- `exist_function()`：用于判断是否存在某个函数。
- `get_function()`：用于获取函数对象。
- `create_function()`：创建函数。
- `delete_function()`：删除函数。

创建函数

使用入口对象的`create_function()`方法即可创建函数。举例如下。

```
# 引用当前project中的资源。
resource = o.get_resource('my_udf.py')
function = o.create_function('test_function', class_type='my_udf.Test', resources=[resource])
# 引用其他project中的资源。
resource2 = o.get_resource('my_udf.py', project='another_project')
function2 = o.create_function('test_function2', class_type='my_udf.Test', resources=[resource2])
```

删除函数

使用入口对象的`delete_function()`方法即可删除函数，也可以使用函数对象调用`drop`方法删除函数。举例如下。

```
o.delete_function('test_function')
function.drop()  # Function对象存在时直接调用drop方法。
```

更新函数

对函数调用`update`方法即可更新函数。举例如下。

```
function = o.get_function('test_function')
new_resource = o.get_resource('my_udf2.py')
function.class_type = 'my_udf2.Test'
function.resources = [new_resource, ]
function.update()  # 更新函数。
```

5.4.8. 集成SQLAlchemy

PyODPS集成了SQLAlchemy，可以使用SQLAlchemy查询MaxCompute数据。本文为您介绍如何创建连接，并调用SQLAlchemy接口。

创建连接

创建连接语法格式如下。

```
from sqlalchemy import create_engine
engine = create_engine('odps://<access_id>:<access_key>@<project>/?endpoint=<endpoint>')
conn = engine.connect()
```

- access_id：具备访问目标MaxCompute项目权限的AccessKey ID。

  您可以进入AccessKey管理页面获取AccessKey ID。
- access_key：AccessKey ID对应的AccessKey Secret。

  您可以进入AccessKey管理页面获取AccessKey Secret。
- project：目标MaxCompute项目的名称。

  此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
- endpoint：目标MaxCompute项目所属地域的Endpoint。

  各地域的Endpoint信息，请参见Endpoint。

对于已有的ODPS对象o，调用`o.to_global()`设为全局账号后，在连接串中就不需要指定上述参数。命令示例如下。

```
from sqlalchemy import create_engine
o.to_global()  # set ODPS object as global one
engine = create_engine('odps://')
```

调用SQLAlchemy接口

创建连接之后，就可以正常调用SQLAlchemy接口。建表、写入数据、查询示例如下。

- 建表

```
from sqlalchemy import Table, Column, Integer, String, MetaData
metadata = MetaData()
users = Table('users', metadata,
    Column('id', Integer),
    Column('name', String),
    Column('fullname', String),
)
metadata.create_all(engine)
```

- 写入数据

```
ins = users.insert().values(id=1, name='jack', fullname='Jack Jones')
conn.execute(ins)
```

- 查询数据

```
from sqlalchemy.sql import select
s = select([users])
result = conn.execute(s)
for row in result:
print(row)
(1, 'jack', 'Jack Jones')
```

5.5. 配置选项

本文为您介绍PyODPS提供的配置选项。

您可以通过 `odps.options` 获得PyODPS提供的配置选项。

```
from odps import options
# 设置所有输出表的生命周期（lifecycle选项）。
options.lifecycle = 30
# 使用Tunnel下载string类型时使用bytes（tunnel.string_as_binary选项）。
options.tunnel.string_as_binary = True
# 用ODPS执行PyODPS DataFrame时，可以参照下面dataframe相关配置，在sort时将limit设置为一个比较大的值。
options.df.odps.sort.limit = 100000000
```

通用配置

选项	说明	默认值
end_point	ODPS Endpoint。	None
default_project	默认Project。	None
log_view_host	LogView主机名。	None
log_view_hours	LogView保持时间（小时）。	24
local_timezone	使用的时区。True表示本地时间，False表示UTC，也可用 `pytz` 的时区。	None
lifecycle	所有表生命周期。	None
temp_lifecycle	临时表生命周期。	1
biz_id	用户ID。	None
verbose	是否打印日志。	False
verbose_log	日志接收器。	None
chunk_size	写入缓冲区大小。	1496
retry_times	请求重试次数。	4
pool_connections	缓存在连接池的连接数。	10
pool_maxsize	连接池最大容量。	10
connect_timeout	连接超时。	5
read_timeout	读取超时。	120

选项	说明	默认值
api_proxy	API代理服务器。	None
data_proxy	数据代理服务器。	None
completion_size	对象补全列举条数限制。	10
notebook_repr_widget	使用交互式图表。	True
sql.settings	ODPS SQL运行全局hints。	None
sql.use_odps2_extension	启用MaxCompute 2.0语言扩展。	False

数据上传下载配置

选项	说明	默认值
tunnel.endpoint	Tunnel的Endpoint。	None
tunnel.use_instance_tunnel	使用Instance Tunnel获取执行结果。	True
tunnel.limit_instance_tunnel	是否限制Instance Tunnel获取结果的条数。	None
tunnel.string_as_binary	在STRING类型中使用Bytes而非Unicode。	False

DataFrame配置

选项	说明	默认值
interactive	是否在交互式环境。	根据检测值。
df.analyze	是否启用非ODPS内置函数。	True
df.optimize	是否开启DataFrame全部优化。	True
df.optimizes.pp	是否开启DataFrame谓词下推优化。	True
df.optimizes.cp	是否开启DataFrame列剪裁优化。	True
df.optimizes.tunnel	是否开启DataFrame使用Tunnel优化执行。	True
df.quote	ODPS SQL后端是否用 `` 来标记字段和表名。	True
df.libraries	DataFrame运行使用的第三方库（资源名）。	None
df.supersede_libraries	使用自行上传的Numpy替换服务中的版本。	False
df.odps.sort.limit	DataFrame有排序操作时，默认添加的Limit条数。	10000

机器学习配置

选项	说明	默认值
ml.xflow_settings	Xflow执行配置。	None
ml.xflow_project	默认Xflow工程名。	algo_public
ml.use_model_transfer	是否使用ModelTransfer获取模型 PMML。	False
ml.model_volume	在使用ModelTransfer时使用的 Volume名称。	pyodps_volume

5.6. DataFrame

5.6.1. DataFrame概述

PyODPS提供了DataFrame API，它提供了类似Pandas的接口，但是能充分利用MaxCompute的计算能力。同时能在本地使用同样的接口，用Pandas进行计算。

- 快速入门：为您介绍如何创建和操作DataFrame对象，以及使用Dataframe完成基本的数据处理。
- 创建DataFrame：为您介绍如何创建DataFrame，用于引用数据源。

- Sequence：为您介绍Sequence。Sequence Expr代表二维数据集中的一列。SequenceExpr只可以从一个Collection中获取，不支持手动创建SequenceExpr。
- Collection：为您介绍Collection。CollectionExpr中包含针对二维数据集的列操作、筛选、变换等大量操作。
- 执行：为您介绍DataFrame操作支持的执行方法。
- MapReduce API：本文为您介绍DataFrame下MapReduce API的使用。
- 列运算：本文为您介绍DataFrame API中的列运算。
- 聚合操作：本文为您介绍DataFrame支持的聚合操作以及如何实现分组聚合、编写自定义聚合。
- 排序、去重、采样、数据变换：本文为您介绍DataFrame对象执行排序、去重、采样、数据变换操作。
- 数据合并：本文向您介绍DataFrame支持的数据表的JOIN操作、UNION操作等数据合并操作。
- 窗口函数：本文为您介绍DataFrame API支持使用窗口函数。
- 绘图：本文为您介绍PyODPS DataFrame提供的绘图方法。
- 调试指南：由于PyODPS DataFrame本身会对整个操作执行优化，为了更直观地反应整个过程， 您可以使用可视化的方式显示整个表达式的计算过程。本文为您介绍DataFrame的调试过程。

您可以参见Python数据处理库pandas入门教程了解Python数据处理库Pandas的更多信息。

5.6.2. 快速入门

本文为您介绍如何创建和操作DataFrame对象，以及使用DataFrame完成基本的数据处理。

DataFrame对象操作

本文将以movielens 100K进行举例。现在已经有了三张表，分别是 `pyodps_ml_100k_movies` （电影相关的数据）、 `pyodps_ml_100k_users` （用户相关的数据）、 `pyodps_ml_100k_ratings` （评分有关的数据）。

1. 当运行环境没有提供ODPS对象时，您需要自己创建该对象。

```
from odps import ODPS
o = ODPS('**your-access-id**', '**your-secret-access-key**',
         project='**your-project**', endpoint='**your-end-point**')
```

2. 创建一个DataFrame对象非常容易，只需传入Table对象即可。

```
from odps.df import DataFrame
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

3. 您可以通过 `dtypes` 属性查看这个DataFrame的字段及字段类型。

```
users.dtypes
odps.Schema {
  user_id             int64
  age                 int64
  sex                 string
  occupation          string
  zip_code            string
}
```

4. 通过 `head` 方法，您可以取前N条数据并进行快速预览。

```
users.head(10)
   user_id  age  sex     occupation  zip_code
0        1   24    M     technician     85711
1        2   53    F          other     94043
2        3   23    M         writer     32067
3        4   24    M     technician     43537
4        5   33    F          other     15213
5        6   42    M      executive     98101
6        7   57    M  administrator     91344
7        8   36    M  administrator     05201
8        9   29    M        student     01002
9       10   53    M         lawyer     90703
```

5. 如果您不需要看到所有字段，则可以进行如下操作：
   - 从中筛选出一部分字段。

```
users[['user_id', 'age']].head(5)
   user_id  age
0        1   24
1        2   53
2        3   23
3        4   24
4        5   33
```

- 只做排除个别字段操作。

```
users.exclude('zip_code', 'age').head(5)
   user_id  sex  occupation
0        1    M  technician
1        2    F       other
2        3    M      writer
3        4    M  technician
4        5    F       other
```

- 排除掉一些字段，通过计算得到一些新的列。例如将 `sex` 为 `M` 的置为True，否则为False，并取名为 `sex_bool` 。

```
users.select(users.exclude('zip_code', 'sex'), sex_bool=users.sex == 'M').head(5)
   user_id  age  occupation  sex_bool
0        1   24  technician      True
1        2   53       other     False
2        3   23      writer      True
3        4   24  technician      True
4        5   33       other     False
```

6. 查看男用户和女用户的个数。

```
users.groupby(users.sex).agg(count=users.count())
   sex  count
0    F    273
1    M    670
```

7. 将用户按职业划分，从高到底进行排序，查看人数最多的前10职业。

```
df = users.groupby('occupation').agg(count=users['occupation'].count())
df.sort(df['count'], ascending=False)[:10]
      occupation  count
0        student    196
1          other    105
2       educator     95
3  administrator     79
4       engineer     67
5     programmer     66
6      librarian     51
7         writer     45
8      executive     32
9      scientist     31
```

或者通过 `value_counts` 方法快速实现。该方法返回的行数受到 `options.df.odps.sort.limit` 的限制，详情请参见配置选项。

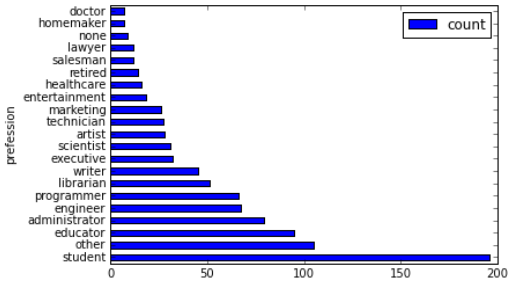
```
users.occupation.value_counts()[:10]
      occupation  count
0        student    196
1          other    105
2       educator     95
3  administrator     79
4       engineer     67
5     programmer     66
6      librarian     51
7         writer     45
8      executive     32
9      scientist     31
```

8. 通过更直观的图查看这份数据。

```
%matplotlib inline
```

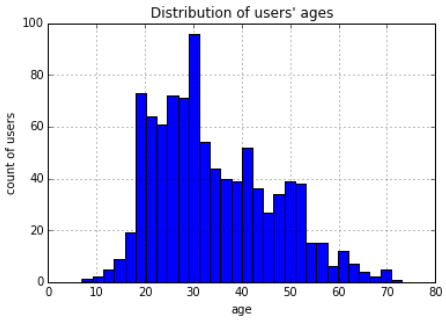
9. 通过横向的柱状图实现可视化。

```
users['occupation'].value_counts().plot(kind='barh', x='occupation', ylabel='prefession')
<matplotlib.axes._subplots.AxesSubplot at 0x10653cfd0>
```



10. 将年龄分为30组，查看年龄分布的直方图。

```
users.age.hist(bins=30, title="Distribution of users' ages", xlabel='age', ylabel='count of users')
<matplotlib.axes._subplots.AxesSubplot at 0x10667a510>
```



11. 此时，只需要使用 `join` 将这三张表联合起来，然后保存为一张新的表。

```
movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
o.delete_table('pyodps_ml_100k_lens', if_exists=True)
lens = movies.join(ratings).join(users).persist('pyodps_ml_100k_lens')
lens.dtypes
odps.Schema {
  movie_id                       int64
  title                          string
  release_date                   string
  video_release_date             string
  imdb_url                       string
  user_id                        int64
  rating                         int64
  unix_timestamp                 int64
  age                            int64
  sex                            string
  occupation                     string
  zip_code                       string
}
```

12. 将年龄（0到80岁）分成8个年龄段。

```
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
 cut_lens = lens[lens, lens.age.cut(range(0, 81, 10), right=False, labels=labels).rename('年龄分组')]
```

13. 取分组和年龄唯一的前10条查看。

```
cut_lens['年龄分组', 'age'].distinct()[:10]
   年龄分组  age
0       0-9    7
1     10-19   10
2     10-19   11
3     10-19   13
4     10-19   14
5     10-19   15
6     10-19   16
7     10-19   17
8     10-19   18
9     10-19   19
```

14. 最后，查看在各个年龄分组下，用户的评分总数和评分均值。

```
cut_lens.groupby('年龄分组').agg(cut_lens.rating.count().rename('评分总数'), cut_lens.rating.mean().rename('评分均值'))
```

	年龄分组	评分均值	评分总数
0	0-9	3.767442	43
1	10-19	3.486126	8181
2	20-29	3.467333	39535
3	30-39	3.554444	25696
4	40-49	3.591772	15021
5	50-59	3.635800	8704
6	60-69	3.648875	2623
7	70-79	3.649746	197

Dataframe数据处理

请您首先下载鸢尾花数据集保存到本地作为示例数据。本文使用DataWorks PyODPS节点功能，详情请参见创建PyODPS 2节点。

1. 创建测试数据表

   使用DataWorks表管理功能新建表，完成后单击**DDL模式**。



   输入建表语句如下，完成后提交表。

```
CREATE TABLE pyodps_iris (
    sepallength double COMMENT '片长度(cm)',
    sepalwidth double COMMENT '片宽度(cm)',
    petallength double COMMENT '瓣长度(cm)',
    petalwidth double COMMENT '瓣宽度(cm)',
    name string COMMENT '种类'
) ;
```

2. 上传测试数据

   在新建表上单击右键，选择**导入数据**。



   输入表名称后，上传您刚下载的数据集。



单击**按位置匹配**后导入数据。



3. 新建一个PyODPS节点，用于存放和运行代码。



4. 输入代码后，单击运行，运行后可在下方**运行日志**处查看结果。



代码详情如下。

```
from odps.df import DataFrame
from odps.df import output
iris = DataFrame(o.get_table('pyodps_iris')) #从ODPS表创建DataFrame对象iris。
print iris.head(10)
print iris.sepallength.head(5) #打印iris部分内容。
#使用自定义函数求iris的两列之和。
print iris.apply(lambda row: row.sepallength + row.sepalwidth, axis=1, reduce=True, types='float').rename('sepaladd').head(3)
#指定函数的输出名称和类型。
@output(['iris_add', 'iris_sub'], ['float', 'float'])
def handle(row):
#使用yield关键字可返回多行结果。
    yield row.sepallength - row.sepalwidth,row.sepallength + row.sepalwidth
    yield row.petallength - row.petalwidth,row.petallength + row.petalwidth
#打印前5行结果，axis=1表示列的轴沿着水平的方向。
print iris.apply(handle,axis=1).head(5)
```

5.6.3. 创建DataFrame

本文为您介绍如何创建DataFrame引用数据源。

背景信息

在使用DataFrame时，您需要了解 `Collection`（`DataFrame`）、`Sequence` 和 `Scalar` 三类对象的操作。三类对象分别表示表结构（或者二维结构）、列（一维结构）和标量。

您使用Pandas数据创建的对象包含实际数据，但通过MaxCompute表创建的对象并不包含实际数据，仅包含对数据的操作。MaxCompute完成数据存储和计算。

创建DataFrame

您唯一需要直接创建的Collection对象是DataFrame。DataFrame用于引用MaxCompute表、MaxCompute分区、Pandas DataFrame或Sqlalchemy Table（数据库表）数据源。这几种数据源的操作相同，您可以不更改数据处理代码，仅修改输入和输出指向，便可以将本地运行的小数据量测试代码迁移到MaxCompute，迁移正确性由PyODPS保证。

创建DataFrame，只需要将Table对象、Pandas DataFrame对象或者Sqlalchemy Table对象传入即可。

```
from odps.df import DataFrame
# 从ODPS表创建DataFrame。
 iris = DataFrame(o.get_table('pyodps_iris'))
 iris2 = o.get_table('pyodps_iris').to_df()  # 使用表的to_df方法。
# 从MaxCompute分区创建DataFrame。
 pt_df = DataFrame(o.get_table('partitioned_table').get_partition('pt=20171111'))
 pt_df2 = o.get_table('partitioned_table').get_partition('pt=20171111').to_df()  # 使用分区的to_df方法。
# 从Pandas DataFrame创建DataFrame。
 import pandas as pd
 import numpy as np
 df = DataFrame(pd.DataFrame(np.arange(9).reshape(3, 3), columns=list('abc')))
# 从Sqlalchemy Table创建DataFrame。
 engine = sqlalchemy.create_engine('mysql://root:123456@localhost/movielens')
 metadata = sqlalchemy.MetaData(bind=engine) # 需要绑定到Engine。
 table = sqlalchemy.Table('top_users', metadata, extend_existing=True, autoload=True)
 users = DataFrame(table)
```

用Pandas DataFrame初始化时：

- PyODPS DataFrame会尝试对NUMPY OBJECT或STRING类型进行推断。如果一整列都为空，则会报错。为避免报错，您可以设置 `unknown_as_string` 值为True，将这些列指定为STRING类型。
- 您可以通过 `as_type` 参数，强制转换类型。如果类型为基本类型，会在创建PyODPS DataFrame时强制转换类型。如果Pandas DataFrame中包含LIST或DICT列，系统不会推断该列的类型，必须手动使用 `as_type` 指定类型。`as_type` 参数类型必须是DICT。

```
df2 = DataFrame(df, unknown_as_string=True, as_type={'null_col2': 'float'})
df2.dtypes
odps.Schema {
  sepallength           float64
  sepalwidth            float64
  petallength           float64
  petalwidth            float64
  name                  string
  null_col1             string   # 无法识别，通过unknown_as_string设置成STRING类型。
  null_col2             float64  # 强制转换成FLOAT类型。
}
 df4 = DataFrame(df3, as_type={'list_col': 'list<int64>'})
 df4.dtypes
odps.Schema {
  id        int64
  list_col  list<int64>  # 无法识别且无法自动转换，通过as_type设置。
}
```

> **说明** PyODPS不支持上传外部表OSS或OTS。

5.6.4. Sequence

SequenceExpr代表二维数据集中的一列。SequenceExpr只可以从一个Collection中获取，不支持手动创建SequenceExpr。

获取列

您可以使用 `collection.column_name` 取出一列。

```
iris.sepallength.head(5)
   sepallength
0          5.1
1          4.9
2          4.7
3          4.6
4          5.0
```

当列名存储在一个字符串变量中，可以使用 `df[column_name]` 实现取出一列。

```
iris['sepallength'].head(5)
   sepallength
0          5.1
1          4.9
2          4.7
3          4.6
4          5.0
```

列类型

DataFrame拥有自己的类型系统，在使用表初始化时，MaxCompute的类型会被转换。这样可以支持更多类型的计算后端。目前，DataFrame的执行后端支持MaxCompute SQL、Pandas和数据库（MySQL和Postgres）。

MaxCompute和DataFrame的数据类型映射关系如下。

MaxCompute类型	DataFrame类型
BIGINT	INT64
DOUBLE	FLOAT64
STRING	STRING
DATETIME	DATETIME
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL
ARRAY<VALUE_TYPE>	LIST<VALUE_TYPE>
MAP<KEY_TYPE, VALUE_TYPE>	DICT<KEY_TYPE, VALUE_TYPE>
当 `options.sql.use_odps2_extension=True` 时，还支持以下数据类型。	
TINYINT	INT8
SMALLINT	INT16
INT	INT32
FLOAT	FLOAT32

数据类型说明如下：

- PyODPS DataFrame支持的数据类型有INT8、INT16、INT32、INT64、FLOAT32、FLOAT64、BOOLEAN、STRING、DECIMAL、DATETIME、LIST、DICT。
- LIST和DICT必须填写其包含值的类型，否则会报错。
- 目前DataFrame暂不支持MaxCompute 2.0中新增的TIMESTAMP及STRUCT类型，后续版本将会陆续支持新增的类型。
- 在Sequence中可以通过 `sequence.dtype` 获取数据类型。

  ```
  iris.sepallength.dtype
  float64
  ```

- 如果要修改一列的类型，可以使用 `astype` 方法。该方法输入一个类型，并返回类型转换后的Sequence。

  ```
  iris.sepallength.astype('int')
     sepallength
  0            5
  1            4
  2            4
  3            4
  4            5
  ```

列名

在DataFrame的计算过程中，每个Sequence必须要有列名。通常，DataFrame会为每个Sequence起一个名字。

```
iris.groupby('name').sepalwidth.max()
   sepalwidth_max
0             4.4
1             3.4
2             3.8
```

上述示例中，`sepalwidth`取最大值后被命名为`sepalwidth_max`。除部分操作外（例如，给指定Sequence加上一个Scalar时，结果会自动被命名为这个Sequence的名字），为Sequence命名的操作需要您自己手动完成。

Sequence提供`rename`方法对一列进行重命名。

```
iris.sepalwidth.rename('sepal_width').head(5)
   sepal_width
0          3.5
1          3.0
2          3.2
3          3.1
4          3.6
```

简单的列变换

您可以对一个Sequence进行运算，返回一个新的Sequence，这种操作类似于对简单的Python变量进行运算。Sequence支持对数值列进行四则运算，而对字符串则仅支持字符串的相加操作。

```
(iris.sepallength + 5).head(5)
   sepallength
0         10.1
1          9.9
2          9.7
3          9.6
4         10.0
```

两列共同参与运算时，PyODPS无法确定最终显示的列名，需要您手动指定。更多列变换说明，请参见列运算。

```
(iris.sepallength + iris.sepalwidth).rename('sum_sepal').head(5)
   sum_sepal
0        8.6
1        7.9
2        7.9
3        7.7
4        8.6
```

5.6.5. Collection

DataFrame中所有二维数据集上的操作都属于CollectionExpr，可视为一张MaxCompute表或一张电子表单，DataFrame对象也是CollectionExpr的特例。CollectionExpr中包含针对二维数据集的列操作、筛选、变换等大量操作。

获取类型

`dtypes`可以用来获取CollectionExpr中所有列的类型。`dtypes`返回的是Schema类型。

```
iris.dtypes
odps.Schema {
  sepallength           float64
  sepalwidth            float64
  petallength           float64
  petalwidth            float64
  name                  string
}
```

列选择和增删

- 列选择

  如果您需要从一个CollectionExpr中选取部分列，产生新的数据集，可以使用`expr[columns]`语法。

```
iris['name', 'sepallength'].head(5)
          name  sepallength
0  Iris-setosa          5.1
1  Iris-setosa          4.9
2  Iris-setosa          4.7
3  Iris-setosa          4.6
4  Iris-setosa          5.0
```

> **说明**
>
> 如果需要选择的列只有一列，需要在Columns后加上逗号或者显示标记为列表，例如 `iris[iris.sepal_length, ]` 或 `iris[[iris.sepal_length]]` ，否则返回的将是一个Sequence对象，而不是Collection。

- 列删除

  如果您需要在新的数据集中排除已有数据集的某些列，可使用 `exclude` 方法。

```
iris.exclude('sepallength', 'petallength')[:5]
   sepalwidth  petalwidth         name
0         3.5         0.2  Iris-setosa
1         3.0         0.2  Iris-setosa
2         3.2         0.2  Iris-setosa
3         3.1         0.2  Iris-setosa
4         3.6         0.2  Iris-setosa
```

  PyODPS 0.7.2版（及以上版本）支持另一种写法，即在数据集上直接排除相应的列。

```
del iris['sepallength']
del iris['petallength']
iris[:5]
   sepalwidth  petalwidth         name
0         3.5         0.2  Iris-setosa
1         3.0         0.2  Iris-setosa
2         3.2         0.2  Iris-setosa
3         3.1         0.2  Iris-setosa
4         3.6         0.2  Iris-setosa
```

- 列增加

  如果您需要在已有Collection中添加某一列变换的结果，可以使用 `expr[expr, new_sequence]` 语法， 新增的列会作为新Collection的一部分。

  以下示例将 `iris` 中的 `sepalwidth` 列加一后重命名为 `sepalwidthplus1` 并追加到数据集末尾，形成新的数据集。

```
iris[iris, (iris.sepalwidth + 1).rename('sepalwidthplus1')].head(5)
   sepallength  sepalwidth  petallength  petalwidth         name  \
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
   sepalwidthplus1
0              4.5
1              4.0
2              4.2
3              4.1
4              4.6
```

  使用 `expr[expr, new_sequence]` 语法需要注意的是，变换后的列名与原列名可能相同。如果需要与原Collection合并， 请将该列重命名。

  PyODPS 0.7.2版（及以上版本）支持直接在当前数据集中追加。

```
iris['sepalwidthplus1'] = iris.sepalwidth + 1
iris.head(5)
   sepallength  sepalwidth  petallength  petalwidth         name  \
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
   sepalwidthplus1
0              4.5
1              4.0
2              4.2
3              4.1
4              4.6
```

- 同时增删列

  您也可以先将原列通过 `exclude` 方法进行排除，再将变换后的新列并入，而不必担心重名。

```
iris[iris.exclude('sepalwidth'), iris.sepalwidth * 2].head(5)
   sepallength  petallength  petalwidth         name  sepalwidth
0          5.1          1.4         0.2  Iris-setosa         7.0
1          4.9          1.4         0.2  Iris-setosa         6.0
2          4.7          1.3         0.2  Iris-setosa         6.4
3          4.6          1.5         0.2  Iris-setosa         6.2
4          5.0          1.4         0.2  Iris-setosa         7.2
```

  PyODPS 0.7.2版（及以上版本）支持直接在当前数据集上覆盖的操作，写法如下。

```
iris['sepalwidth'] = iris.sepalwidth * 2
iris.head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         7.0          1.4         0.2  Iris-setosa
1          4.9         6.0          1.4         0.2  Iris-setosa
2          4.7         6.4          1.3         0.2  Iris-setosa
3          4.6         6.2          1.5         0.2  Iris-setosa
4          5.0         7.2          1.4         0.2  Iris-setosa
```

  以创建新Collection的方式实现增删列的另一种方法是调用 `select` ，将需要选择的列作为参数输入。如果您需要重命名，可以使用 `keyword` 参数输入，并将新的列名作为参数名。

```
iris.select('name', sepalwidthminus1=iris.sepalwidth - 1).head(5)
          name  sepalwidthminus1
0  Iris-setosa               2.5
1  Iris-setosa               2.0
2  Iris-setosa               2.2
3  Iris-setosa               2.1
4  Iris-setosa               2.6
```

  您也可以传入一个Lambda表达式，它接收的参数为上一步的运算结果。在执行时，PyODPS会检查这些Lambda表达式，传入上一步生成的Collection并将其替换为正确的列。

```
iris['name', 'petallength'][[lambda x: x.name]].head(5)
          name
0  Iris-setosa
1  Iris-setosa
2  Iris-setosa
3  Iris-setosa
4  Iris-setosa
```

  PyODPS 0.7.2版（及以上版本）支持对数据进行条件赋值。

```
iris[iris.sepallength > 5.0, 'sepalwidth'] = iris.sepalwidth * 2
iris.head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1        14.0          1.4         0.2  Iris-setosa
1          4.9         6.0          1.4         0.2  Iris-setosa
2          4.7         6.4          1.3         0.2  Iris-setosa
3          4.6         6.2          1.5         0.2  Iris-setosa
4          5.0         7.2          1.4         0.2  Iris-setosa
```

引入常数和随机数

- DataFrame支持在Collection中追加一列常数。追加常数需要使用Scalar，引入时需要手动指定列名。

```
from odps.df import Scalar
iris[iris, Scalar(1).rename('id')][:5]
   sepallength  sepalwidth  petallength  petalwidth         name  id
0          5.1         3.5          1.4         0.2  Iris-setosa   1
1          4.9         3.0          1.4         0.2  Iris-setosa   1
2          4.7         3.2          1.3         0.2  Iris-setosa   1
3          4.6         3.1          1.5         0.2  Iris-setosa   1
4          5.0         3.6          1.4         0.2  Iris-setosa   1
```

  如果需要指定一个空值列，您可以使用NullScalar，但需要提供字段类型。

```
from odps.df import NullScalar
iris[iris, NullScalar('float').rename('fid')][:5]
   sepal_length  sepal_width  petal_length  petal_width     category   fid
0           5.1          3.5           1.4          0.2  Iris-setosa  None
1           4.9          3.0           1.4          0.2  Iris-setosa  None
2           4.7          3.2           1.3          0.2  Iris-setosa  None
3           4.6          3.1           1.5          0.2  Iris-setosa  None
4           5.0          3.6           1.4          0.2  Iris-setosa  None
```

在PyODPS 0.7.12（及以上版本）中，引入了简化写法。

```
iris['id'] = 1
iris
   sepallength  sepalwidth  petallength  petalwidth         name  id
0          5.1         3.5          1.4         0.2  Iris-setosa   1
1          4.9         3.0          1.4         0.2  Iris-setosa   1
2          4.7         3.2          1.3         0.2  Iris-setosa   1
3          4.6         3.1          1.5         0.2  Iris-setosa   1
4          5.0         3.6          1.4         0.2  Iris-setosa   1
```

需要注意的是，这种写法无法自动识别空值的类型，所以在增加空值列时，仍然要使用如下代码。

```
iris['null_col'] = NullScalar('float')
iris
   sepallength  sepalwidth  petallength  petalwidth         name  null_col
0          5.1         3.5          1.4         0.2  Iris-setosa      None
1          4.9         3.0          1.4         0.2  Iris-setosa      None
2          4.7         3.2          1.3         0.2  Iris-setosa      None
3          4.6         3.1          1.5         0.2  Iris-setosa      None
4          5.0         3.6          1.4         0.2  Iris-setosa      None
```

- DataFrame也支持在Collection中增加一列随机数列，该列类型为FLOAT，范围为0~1，每行数值均不同。追加随机数列需要使用RandomScalar，参数为随机数种子，可省略。

```
from odps.df import RandomScalar
iris[iris, RandomScalar().rename('rand_val')][:5]
   sepallength  sepalwidth  petallength  petalwidth         name  rand_val
0          5.1         3.5          1.4         0.2  Iris-setosa  0.000471
1          4.9         3.0          1.4         0.2  Iris-setosa  0.799520
2          4.7         3.2          1.3         0.2  Iris-setosa  0.834609
3          4.6         3.1          1.5         0.2  Iris-setosa  0.106921
4          5.0         3.6          1.4         0.2  Iris-setosa  0.763442
```

过滤数据

Collection提供了数据过滤的功能。查询 `sepallength` 大于5的数据，示例如下。

```
iris[iris.sepallength > 5].head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          5.4         3.9          1.7         0.4  Iris-setosa
2          5.4         3.7          1.5         0.2  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.7         4.4          1.5         0.4  Iris-setosa
```

多种查询方式的示例：

- 与（&）条件

```
iris[(iris.sepallength < 5) & (iris['petallength'] > 1.5)].head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          4.8         3.4          1.6         0.2     Iris-setosa
1          4.8         3.4          1.9         0.2     Iris-setosa
2          4.7         3.2          1.6         0.2     Iris-setosa
3          4.8         3.1          1.6         0.2     Iris-setosa
4          4.9         2.4          3.3         1.0  Iris-versicolor
```

- 或（|）条件

```
iris[(iris.sepalwidth < 2.5) | (iris.sepalwidth > 4)].head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.7         4.4          1.5         0.4     Iris-setosa
1          5.2         4.1          1.5         0.1     Iris-setosa
2          5.5         4.2          1.4         0.2     Iris-setosa
3          4.5         2.3          1.3         0.3     Iris-setosa
4          5.5         2.3          4.0         1.3 Iris-versicolor
```

> **说明** 与和或条件必须使用 `&` 和 `|`，不能使用 `and` 和 `or`。

- 非（~）条件

```
iris[~(iris.sepalwidth > 3)].head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          4.9         3.0          1.4         0.2 Iris-setosa
1          4.4         2.9          1.4         0.2 Iris-setosa
2          4.8         3.0          1.4         0.1 Iris-setosa
3          4.3         3.0          1.1         0.1 Iris-setosa
4          5.0         3.0          1.6         0.2 Iris-setosa
```

- 显式调用 `filter` 方法，提供多个与条件。

```
iris.filter(iris.sepalwidth > 3.5, iris.sepalwidth < 4).head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.0         3.6          1.4         0.2 Iris-setosa
1          5.4         3.9          1.7         0.4 Iris-setosa
2          5.4         3.7          1.5         0.2 Iris-setosa
3          5.4         3.9          1.3         0.4 Iris-setosa
4          5.7         3.8          1.7         0.3 Iris-setosa
```

- 对于连续的操作，使用Lambda表达式。

```
iris[iris.sepalwidth > 3.8]['name', lambda x: x.sepallength + 1]
          name  sepallength
0  Iris-setosa          6.4
1  Iris-setosa          6.8
2  Iris-setosa          6.7
3  Iris-setosa          6.4
4  Iris-setosa          6.2
5  Iris-setosa          6.5
```

- 对于Collection，如果它包含一个列是BOOLEAN类型，则可以直接使用该列作为过滤条件。

```
df.dtypes
odps.Schema {
  a boolean
  b int64
}
df[df.a]
      a  b
0  True  1
1  True  3
```

因此，对Collection取单个Sequence的操作时，只有BOOLEAN列是合法的，即对Collection作过滤操作。

```
df[df.a, ]       # 取列操作。
df[[df.a]]       # 取列操作。
df.select(df.a)  # 显式取列。
df[df.a]         # a列是boolean列，执行过滤操作。
df.a             # 取单列。
df['a']          # 取单列。
```

- 同时，您可以使用Pandas中的 `query` 方法，通过查询语句做数据的筛选，在表达式中直接使用列名如 `sepallength` 进行操作。在查询语句中，`&` 和 `and` 都表示与操作，`|` 和 `or` 都表示或操作。

```
iris.query("(sepallength < 5) and (petallength > 1.5)").head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          4.8         3.4          1.6         0.2     Iris-setosa
1          4.8         3.4          1.9         0.2     Iris-setosa
2          4.7         3.2          1.6         0.2     Iris-setosa
3          4.8         3.1          1.6         0.2     Iris-setosa
4          4.9         2.4          3.3         1.0 Iris-versicolor
```

当表达式中需要使用到本地变量时，需要在该变量前加一个 `@` 前缀。

```
var = 4
iris.query("(iris.sepalwidth < 2.5) | (sepalwidth > @var)").head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.7         4.4          1.5         0.4     Iris-setosa
1          5.2         4.1          1.5         0.1     Iris-setosa
2          5.5         4.2          1.4         0.2     Iris-setosa
3          4.5         2.3          1.3         0.3     Iris-setosa
4          5.5         2.3          4.0         1.3  Iris-versicolor
```

目前 `query` 支持的语法，如下表所示。

语法	说明
name	没有 `@` 前缀的都当做列名处理，有前缀的会获取本地变量。
operator	支持部分运算符：`+`、`-`、`*`、`/`、`//`、`%`、`**`、`==`、`!=`、`<`、`<=`、`>`、`>=`、`in`、`not in`。
bool	与或非操作，其中 `&` 和 `and` 表示与，`|` 和 `or` 表示或。
attribute	取对象属性。
index, slice, subscript	切片操作。

并列多行输出

对于LIST及MAP类型的列，`explode` 方法会将该列转换为多行输出。您也可以使用 `apply` 方法实现多行输出。为了进行聚合等操作，常常需要将这些输出和原表中的列合并。此时可以使用DataFrame提供的并列多行输出功能， 写法为将多行输出函数生成的集合与原集合中的列名一起映射。

并列多行输出的示例如下。

```
df
   id         a             b
0   1  [a1, b1]  [a2, b2, c2]
1   2      [c1]      [d2, e2]
df[df.id, df.a.explode(), df.b]
   id   a             b
0   1  a1  [a2, b2, c2]
1   1  b1  [a2, b2, c2]
2   2  c1      [d2, e2]
df[df.id, df.a.explode(), df.b.explode()]
   id   a   b
0   1  a1  a2
1   1  a1  b2
2   1  a1  c2
3   1  b1  a2
4   1  b1  b2
5   1  b1  c2
6   2  c1  d2
7   2  c1  e2
```

如果多行输出方法对某个输入不产生任何输出，默认输入行将不在最终结果中出现。如果需要在结果中出现该行，可以设置 `keep_nulls=True`。此时，与该行并列的值将输出为空值。

```
df
   id         a
0   1  [a1, b1]
1   2        []
df[df.id, df.a.explode()]
   id   a
0   1  a1
1   1  b1
df[df.id, df.a.explode(keep_nulls=True)]
   id     a
0   1    a1
1   1    b1
2   2  None
```

使用 `explode` 方法实现并列多行输出的示例，请参见集合类型相关操作。

限制条数

输出前三条数据。

```
iris[:3]
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
```

目前对于MaxCompute SQL，后端切片不支持 `start` 和 `step` 方法，但支持 `limit` 方法。

```
iris.limit(3)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
```

> **说明** 切片操作只能作用于Collection，不能作用于Sequence。

5.6.6. 执行

本文为您介绍DataFrame操作支持的执行方法。

延迟执行

DataFrame上的所有操作并不会立即执行，只有当显式调用 `execute` 方法，或者调用立即执行的方法时（内部调用的也是 `execute` ），才会执行这些操作。立即执行的方法如下表所示。

方法	说明	返回值
persist	将执行结果保存到MaxCompute表。	PyODPS DataFrame
execute	执行并返回全部结果。	ResultFrame
head	查看开头N行数据，这个方法会执行所有结果，并取开头N行数据。	ResultFrame
tail	查看结尾N行数据，这个方法会执行所有结果，并取结尾N行数据。	ResultFrame
to_pandas	转化为Pandas DataFrame或者Series，wrap参数为True的时候，返回PyODPS DataFrame对象。	• wrap为True时，返回PyODPS DataFrame。 • wrap为False时，返回Pandas DataFrame。False为默认值。
plot，hist，boxplot	画图有关。	不涉及

> **说明** 在交互式环境下，PyODPS DataFrame会在打印或者 `repr` 的时候，调用 `execute` 方法，您无需手动调用 `execute` 。

示例

```
>>> iris[iris.sepallength < 5][:5]
   sepallength  sepalwidth  petallength  petalwidth         name
0          4.9         3.0          1.4         0.2  Iris-setosa
1          4.7         3.2          1.3         0.2  Iris-setosa
2          4.6         3.1          1.5         0.2  Iris-setosa
3          4.6         3.4          1.4         0.3  Iris-setosa
4          4.4         2.9          1.4         0.2  Iris-setosa
```

如果您需要关闭自动调用执行，请进行手动设置。

```
>>> from odps import options
>>> options.interactive = False
>>>
>>> iris[iris.sepallength < 5][:5]
Collection: ref_0
  odps.Table
    name: odps_test_sqltask_finance.`pyodps_iris`
    schema:
      sepallength           : double
      sepalwidth            : double
      petallength           : double
      petalwidth            : double
      name                  : string
Collection: ref_1
  Filter[collection]
    collection: ref_0
    predicate:
      Less[sequence(boolean)]
        sepallength = Column[sequence(float64)] 'sepallength' from collection ref_0
        Scalar[int8]
          5
Slice[collection]
  collection: ref_1
  stop:
    Scalar[int8]
      5
```

此时打印 `repr` 对象，会显示整个抽象语法树。

> **说明** ResultFrame是结果集合，不能参与后续计算。

ResultFrame可以迭代取出每条记录。

```
>>> result = iris.head(3)
>>> for r in result:
>>>     print(list(r))
[5.0999999999999996, 3.5, 1.3999999999999999, 0.20000000000000001, u'Iris-setosa']
[4.9000000000000004, 3.0, 1.3999999999999999, 0.20000000000000001, u'Iris-setosa']
[4.7000000000000002, 3.2000000000000002, 1.3, 0.20000000000000001, u'Iris-setosa']
```

ResultFrame也支持在安装有Pandas的前提下转换为Pandas DataFrame，或使用Pandas后端的PyODPS DataFrame。

```
pd_df = iris.head(3).to_pandas()  # 返回Pandas DataFrame。
 wrapped_df = iris.head(3).to_pandas(wrap=True)  # 返回使用Pandas后端的PyODPS DataFrame。
```

保存执行结果为MaxCompute表

对于Collection，您可以调用 `persist` 方法，用于返回一个新的DataFrame对象，参数为表名。

```
>>> iris2 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris2')
>>> iris2.head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          4.5         2.3          1.3         0.3     Iris-setosa
1          5.5         2.3          4.0         1.3  Iris-versicolor
2          4.9         2.4          3.3         1.0  Iris-versicolor
3          5.0         2.0          3.5         1.0  Iris-versicolor
4          6.0         2.2          4.0         1.0  Iris-versicolor
```

`persist` 可以传入 `partitions` 参数，用于创建一个分区表。它的分区是 `partitions` 所指定的字段。

```
iris3 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris3', partitions=['name'])
 iris3.data
odps.Table
  name: odps_test_sqltask_finance.`pyodps_iris3`
  schema:
    sepallength           : double
    sepalwidth            : double
    petallength           : double
    petalwidth            : double
  partitions:
    name                  : string
```

如果您需要写入已经存在的表的某个分区，`persist`可以传入`partition`参数，指明写入表的哪个分区（例如`ds=******`）。该DataFrame的每个字段的类型都必须相同，且都存在于该表中。`drop_partition`和`create_partition`参数只在此时有效，分别表示是否要删除（如果分区存在）或创建（如果分区不存在）该分区。

```
iris[iris.sepalwidth < 2.5].persist('pyodps_iris4', partition='ds=test', drop_partition=True, create_partition=True)
```

写入表时，您还可以指定表的生命周期。例如下列语句将表的生命周期指定为10天。

```
iris[iris.sepalwidth < 2.5].persist('pyodps_iris5', lifecycle=10)
```

如果数据源中没有ODPS对象，例如数据源仅为Pandas，在`persist`时需要手动指定ODPS入口对象，或者将需要的入口对象标明为全局对象。

```
# 假设入口对象为o。
# 指定入口对象。
df.persist('table_name', odps=o)
# 或者可将入口对象标记为全局。
o.to_global()
df.persist('table_name')
```

保存执行结果为Pandas DataFrame

您可以使用`to_pandas`方法，如果`wrap`参数为True，将返回PyODPS DataFrame对象。

```
type(iris[iris.sepalwidth < 2.5].to_pandas())
pandas.core.frame.DataFrame
type(iris[iris.sepalwidth < 2.5].to_pandas(wrap=True))
odps.df.core.DataFrame
```

将表中的数据读出，PyODPS可以执行`open_reader`方法，通过`reader.to_pandas()`转成Pandas DataFrame。

立即运行设置运行参数

对于立即执行的方法，例如`execute`、`persist`、`to_pandas`等，您可以通过以下方法设置它们运行时的参数（仅对ODPS SQL后端有效）：

- 设置全局参数。详情请参见SQL。
- 在这些立即执行的方法上，使用`hints`参数，可以确保这些参数只作用于当前的计算过程。

```
iris[iris.sepallength < 5].to_pandas(hints={'odps.sql.mapper.split.size': 16})
```

运行时显示详细信息

如果您需要查看运行时Instance的Logview，则应该修改全局配置。

```
from odps import options
 options.verbose = True
 iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
Sql compiled:
SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`
FROM odps_test_sqltask_finance.`pyodps_iris` t1
WHERE t1.`sepallength` < 5
LIMIT 5
logview:
http://logview
   sepalwidth  petallength  petalwidth         name
0         3.0          1.4         0.2  Iris-setosa
1         3.2          1.3         0.2  Iris-setosa
2         3.1          1.5         0.2  Iris-setosa
3         3.4          1.4         0.3  Iris-setosa
4         2.9          1.4         0.2  Iris-setosa
```

您可以指定自己的日志记录函数。

```
my_logs = []
 def my_logger(x):
     my_logs.append(x)
 options.verbose_log = my_logger
 iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
   sepalwidth  petallength  petalwidth         name
0         3.0          1.4         0.2  Iris-setosa
1         3.2          1.3         0.2  Iris-setosa
2         3.1          1.5         0.2  Iris-setosa
3         3.4          1.4         0.3  Iris-setosa
4         2.9          1.4         0.2  Iris-setosa
 print(my_logs)
['Sql compiled:', 'SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name` \nFROM odps_test_sqltask_finance.`pyodps_
iris` t1 \nWHERE t1.`sepallength` < 5 \nLIMIT 5', 'logview:', u'http://logview']
```

缓存中间Collection计算结果

DataFrame的计算过程中，部分Collection被多处使用。如果您需要查看中间过程的执行结果， 可以使用 `cache` 标记某个需要被优先计算的Collection。

> **说明** `cache` 会延迟执行，调用 `cache` 不会触发立即计算。

```
cached = iris[iris.sepalwidth < 3.5].cache()
 df = cached['sepallength', 'name'].head(3)
 df
   sepallength         name
0          4.9  Iris-setosa
1          4.7  Iris-setosa
2          4.6  Iris-setosa
 cached.head(3)  # 由于cached已经被计算，所以能立刻取到计算结果。
   sepallength         name
0          4.9  Iris-setosa
1          4.7  Iris-setosa
2          4.6  Iris-setosa
```

异步和并行执行

DataFrame支持异步操作，对于立即执行的方法，包括 `execute` 、 `persist` 、 `head` 、 `tail` 、 `to_pandas` （其他方法不支持）， 传入 `async` 参数，即可以将这个操作异步执行， `timeout` 参数指定超时时间， 异步返回的是 Future对象。

```
from odps.df import Delay
 delay = Delay()  # 创建Delay对象。
 df = iris[iris.sepal_width < 5].cache()  # 有一个共同的依赖。
 future1 = df.sepal_width.sum().execute(delay=delay)  # 立即返回future对象，此时并没有执行。
 future2 = df.sepal_width.mean().execute(delay=delay)
 future3 = df.sepal_length.max().execute(delay=delay)
 delay.execute(n_parallel=3)  # 并发度是3，此时才真正执行。
|========================================|   1 /  1  (100.00%)        21s
 future1.result()
 458.10000000000014
 future2.result()
3.0540000000000007
```

上述示例中，共同依赖的对象会先执行，然后再以并发度为3分别执行 `future1` 到 `future3` 。当 `n_parallel` 为1时，执行时间会达到37s。

`delay.execute` 也接受 `async` 操作指定是否异步执行，当异步执行的时候，也可以用 `timeout` 参数指定超时时间。

5.6.7. 列运算

本文为您介绍DataFrame API中的列运算。

列运算

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
lens = DataFrame(o.get_table('pyodps_ml_100k_lens'))
```

为一个Sequence加上一个常量或执行sin函数时，这些操作将作用于Sequence中的每个元素。

NULL相关（isnull，notnull，fillna）

DataFrame API提供了几个和NULL相关的内置函数，例如 `isnull` 用于判断某字段是否为NULL， `notnull` 用于判断某字段是否为非NULL， `fillna` 用于将NULL填充为您指定的值。

```
>>> iris.sepallength.isnull().head(5)
   sepallength
0        False
1        False
2        False
3        False
4        False
```

逻辑判断（ifelse，switch）

`ifelse` 作用于BOOLEAN类型的字段，当条件成立时，返回第0个参数，否则返回第1个参数。

```
>>> (iris.sepallength > 5).ifelse('gt5', 'lte5').rename('cmp5').head(5)
   cmp5
0   gt5
1  lte5
2  lte5
3  lte5
4  lte5
```

`switch` 用于多条件判断的情况。

```
>>> iris.sepallength.switch(4.9, 'eq4.9', 5.0, 'eq5.0', default='noeq').rename('equalness').head(5)
   equalness
0       noeq
1      eq4.9
2       noeq
3       noeq
4      eq5.0
```

```
>>> from odps.df import switch
>>> switch(iris.sepallength == 4.9, 'eq4.9', iris.sepallength == 5.0, 'eq5.0', default='noeq').rename('equalness').head(5)
   equalness
0       noeq
1      eq4.9
2       noeq
3       noeq
4      eq5.0
```

PyODPS 0.7.8以上版本支持根据条件修改数据集某一列的一部分值，写法如下。

```
>>> iris[iris.sepallength > 5, 'cmp5'] = 'gt5'
>>> iris[iris.sepallength <= 5, 'cmp5'] = 'lte5'
>>> iris.head(5)
   cmp5
0   gt5
1  lte5
2  lte5
3  lte5
4  lte5
```

数学运算

对于数字类型的字段，支持加法（+）、减法（-）、乘法（*）和除法（/）等操作，也支持log、sin等数学计算。

```
>>> (iris.sepallength * 10).log().head(5)
   sepallength
0     3.931826
1     3.891820
2     3.850148
3     3.828641
4     3.912023
```

```
>>> fields = [iris.sepallength,
>>>           (iris.sepallength / 2).rename('sepallength除以2'),
>>>           (iris.sepallength ** 2).rename('sepallength的平方')]
>>> iris[fields].head(5)
   sepallength  sepallength除以2  sepallength的平方
0          5.1              2.55             26.01
1          4.9              2.45             24.01
2          4.7              2.35             22.09
3          4.6              2.30             21.16
4          5.0              2.50             25.00
```

算术运算支持如下操作。

算术操作	说明
abs	绝对值
sqrt	平方根
sin	无
sinh	无
cos	无
cosh	无
tan	无
tanh	无
arccos	无
arccosh	无
arcsin	无
arcsinh	无
arctan	无
arctanh	无
exp	指数函数
expm1	指数减1
log	传入参数表示底是几。
log2	无
log10	无
log1p	log(1+x)
radians	给定角度计算弧度。
degrees	给定弧度计算角度。
ceil	不小于输入值的最小整数。
floor	向下取整，返回比输入值小的整数值。
trunc	将输入值截取到指定小数点位置。

Sequence也支持自身与其它Sequence或Scalar的比较。

```
>>> (iris.sepallength < 5).head(5)
   sepallength
0        False
1         True
2         True
3         True
4        False
```

虽然DataFrame API不支持连续操作，例如 `3 <= iris.sepallength <= 5` ，但是 `between` 函数可以用于判断 `iris.sepallength` 是否在某个区间。

```
>>> (iris.sepallength.between(3, 5)).head(5)
   sepallength
0        False
1         True
2         True
3         True
4         True
```

默认情况下， `between` 包含两边的区间，如果计算开区间，则需要设置 `inclusive=False` 。

```
>>> (iris.sepallength.between(3, 5, inclusive=False)).head(5)
   sepallength
0        False
1         True
2         True
3         True
4        False
```

STRING相关操作

DataFrame API提供了一系列针对STRING类型的Sequence或者Scalar的操作。

```
>>> fields = [
>>>     iris.name.upper().rename('upper_name'),
>>>     iris.name.extract('Iris(.*)', group=1)
>>> ]
>>> iris[fields].head(5)
    upper_name     name
0  IRIS-SETOSA  -setosa
1  IRIS-SETOSA  -setosa
2  IRIS-SETOSA  -setosa
3  IRIS-SETOSA  -setosa
4  IRIS-SETOSA  -setosa
```

STRING的相关操作如下。

STRING操作	说明
capitalize	无
contains	包含某个字符串，如果 `regex` 参数为True，则是包含某个正则表达式，默认为True。
count	指定字符串出现的次数。
endswith	以某个字符串结尾。
startswith	以某个字符串开头。
extract	抽取出某个正则表达式，如果group不指定，则返回满足整个pattern的子串；否则，返回第几个group。
find	返回第一次出现的子串位置，若不存在则返回-1。
rfind	从右查找返回子串第一次出现的位置，不存在则返回-1。
replace	将某个pattern的子串全部替换成另一个子串， n参数若指定，则替换n次。
get	返回某个位置上的字符串。
len	返回字符串的长度。
ljust	若未达到指定的 `width` 的长度，则在右侧填充 `fillchar` 指定的字符串（默认空格）。
rjust	若未达到指定的 `width` 的长度，则在左侧填充 `fillchar` 指定的字符串（默认空格）。
lower	变为全部小写。

STRING操作	说明
upper	变为全部大写。
lstrip	在左侧删除空格（包括空行符）。
rstrip	在右侧删除空格（包括空行符）。
strip	在左右两侧删除空格（包括空行符）。
split	将字符串按分隔符拆分为若干个字符串（返回 list<string>类型）。
pad	在指定的位置（left，right或者both）用指定填充字符（用 `fillchar` 指定，默认空格）来对齐。
repeat	重复指定n次。
slice	切片操作。
swapcase	对调大小写。
title	同 `str.title` 。
zfill	长度没达到指定 `width` ，则左侧填充0。
isalnum	同 `str.isalnum` 。
isalpha	同 `str.isalpha` 。
isdigit	是否都是数字，同 `str.isdigit` 。
isspace	是否都是空格，同 `str.isspace` 。
islower	是否都是小写，同 `str.islower` 。
isupper	是否都是大写，同 `str.isupper` 。
istitle	同 `str.istitle` 。
isnumeric	同 `str.isnumeric` 。
isdecimal	同 `str.isdecimal` 。
todict	将字符串按分隔符拆分为一个Dict，传入的两个参数分别为项目分隔符和Key-Value分隔符（返回dict<string, string>类型）。
strptime	格式化时间，时间格式和Python标准库相同，详情请参见 Python 时间格式化。

时间相关操作

对于DATETIME类型Sequence或者Scalar，可以调用时间相关的内置函数。

```
>>> df = lens[[lens.unix_timestamp.astype('datetime').rename('dt')]]
>>> df[df.dt,
>>>    df.dt.year.rename('year'),
>>>    df.dt.month.rename('month'),
>>>    df.dt.day.rename('day'),
>>>    df.dt.hour.rename('hour')].head(5)
                   dt  year  month  day  hour
0  1998-04-08 11:02:00  1998      4    8    11
1  1998-04-08 10:57:55  1998      4    8    10
2  1998-04-08 10:45:26  1998      4    8    10
3  1998-04-08 10:25:52  1998      4    8    10
4  1998-04-08 10:44:19  1998      4    8    10
```

与时间相关的属性如下。

时间相关属性	说明
year	无
month	无

时间相关属性	说明
day	无
hour	无
minute	无
second	无
weekofyear	返回日期位于那一年的第几周。周一作为一周的第一天。
weekday	返回日期当前周的第几天。
dayofweek	返回日期当前周的第几天。
strftime	格式化时间，时间格式和Python标准库相同，详情请参见 Python 时间格式化。

PyODPS也支持时间的加减操作，例如可以通过以下方法得到前3天的日期。两个日期列相减得到相差的毫秒数。

```
>>> df
                          a                          b
0 2016-12-06 16:43:12.460001 2016-12-06 17:43:12.460018
1 2016-12-06 16:43:12.460012 2016-12-06 17:43:12.460021
2 2016-12-06 16:43:12.460015 2016-12-06 17:43:12.460022
>>> from odps.df import day
>>> df.a - day(3)
                          a
0 2016-12-03 16:43:12.460001
1 2016-12-03 16:43:12.460012
2 2016-12-03 16:43:12.460015
>>> (df.b - df.a).dtype
int64
>>> (df.b - df.a).rename('a')
         a
0  3600000
1  3600000
2  3600000
```

支持的时间类型如下表所示。

属性	说明
year	无
month	无
day	无
hour	无
minute	无
second	无
millisecond	无

集合类型相关操作

PyODPS支持的集合类型有List和Dict。这两个类型都可以使用下标获取集合中的某个项目。`len`方法用于获得集合的大小。

同时，两种集合均有`explode`方法，用于展开集合中的内容。对于List，`explode`默认返回一列，当传入参数`pos`时，将返回两列，其中一列为值在数组中的编号（类似Python的`enumerate`函数）。对于Dict，`explode`会返回两列，分别表示keys及values。`explode`中也可以传入列名，作为最后生成的列。

示例如下。

```
>>> df
   id         a                             b
0   1  [a1, b1]  {'a2': 0, 'b2': 1, 'c2': 2}
1   2      [c1]           {'d2': 3, 'e2': 4}
>>> df[df.id, df.a[0], df.b['b2']]
   id   a    b
0   1  a1    1
1   2  c1  NaN
>>> df[df.id, df.a.len(), df.b.len()]
   id  a  b
0   1  2  3
1   2  1  2
>>> df.a.explode()
    a
0  a1
1  b1
2  c1
>>> df.a.explode(pos=True)
   a_pos   a
0      0  a1
1      1  b1
2      0  c1
>>> # 指定列名。
>>> df.a.explode(['pos', 'value'], pos=True)
   pos value
0    0    a1
1    1    b1
2    0    c1
>>> df.b.explode()
  b_key  b_value
0    a2        0
1    b2        1
2    c2        2
3    d2        3
4    e2        4
>>> # 指定列名。
>>> df.b.explode(['key', 'value'])
  key  value
0  a2      0
1  b2      1
2  c2      2
3  d2      3
4  e2      4
```

`explode` 也可以和并列多行输出结合，以将原有列和 `explode` 的结果相结合，示例如下。

```
>>> df[df.id, df.a.explode()]
   id   a
0   1  a1
1   1  b1
2   2  c1
>>> df[df.id, df.a.explode(), df.b.explode()]
   id   a b_key  b_value
0   1  a1    a2        0
1   1  a1    b2        1
2   1  a1    c2        2
3   1  b1    a2        0
4   1  b1    b2        1
5   1  b1    c2        2
6   2  c1    d2        3
7   2  c1    e2        4
```

除了下标 `len` 和 `explode` 两个共有方法以外，List还支持下列方法。

List 操作	说明
contains(v)	列表是否包含某个元素。
sort	返回排序后的列表（返回值为List）。

Dict还支持下列方法。

dict操作	说明
keys	获取Dict keys（返回值为List）。
values	获取Dict values（返回值为List）。

其它元素操作（isin，notin，cut）

`isin` 用于判断Sequence里的元素是否在某个集合元素里，`notin` 反之。

```
>>> iris.sepallength.isin([4.9, 5.1]).rename('sepallength').head(5)
   sepallength
0         True
1         True
2        False
3        False
4        False
```

`cut` 提供离散化的操作，可以将Sequence的数据拆成几个区段。

```
>>> iris.sepallength.cut(range(6), labels=['0-1', '1-2', '2-3', '3-4', '4-5']).rename('sepallength_cut').head(5)
   sepallength_cut
0             None
1              4-5
2              4-5
3              4-5
4              4-5
```

`include_under` 和 `include_over` 可以分别包括向下和向上的区间。

```
>>> labels = ['0-1', '1-2', '2-3', '3-4', '4-5', '5-']
>>> iris.sepallength.cut(range(6), labels=labels, include_over=True).rename('sepallength_cut').head(5)
   sepallength_cut
0               5-
1              4-5
2              4-5
3              4-5
4              4-5
```

调用MaxCompute内建或者已定义函数

如果您需要调用MaxCompute上的内建或者已定义函数来生成列，您可以使用 `func` 接口，该接口默认函数返回值为STRING，可以用 `rtype` 参数指定返回值。

```
>>> from odps.df import func
>>>
>>> iris[iris.name, func.rand(rtype='float').rename('rand')][:4]
>>> iris[iris.name, func.rand(10, rtype='float').rename('rand')][:4]
>>> # 调用ODPS上定义的UDF，列名无法确定时需要手动指定。
>>> iris[iris.name, func.your_udf(iris.sepalwidth, iris.sepallength, rtype='float').rename('new_col')]
>>> # 从其它Project调用UDF，也可通过name参数指定列名。
>>> iris[iris.name, func.your_udf(iris.sepalwidth, iris.sepallength, rtype='float', project='udf_project', name='new_col')]
```

> 说明 Pandas后端不支持执行带有 `func` 的表达式。

5.6.8. 聚合操作

本文为您介绍DataFrame支持的聚合操作，以及如何实现分组聚合和编写自定义聚合。DataFrame提供对列进行HyperLogLog计数的接口。

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
```

常用聚合操作如下：

- 使用 `describe` 函数，查看DataFrame里数字列的数量、最大值、最小值、平均值以及标准差。

  ```
  >>> print(iris.describe())
  ```

  返回结果如下。

```
    type  sepal_length  sepal_width  petal_length  petal_width
0  count    150.000000   150.000000    150.000000   150.000000
1   mean      5.843333     3.054000      3.758667     1.198667
2    std      0.828066     0.433594      1.764420     0.763161
3    min      4.300000     2.000000      1.000000     0.100000
4    max      7.900000     4.400000      6.900000     2.500000
```

- 使用单列执行聚合操作。

```
>>> iris.sepallength.max()
```

返回结果如下。

```
7.9
```

- 如果在消除重复后的列上进行聚合，可以先调用 `unique` 函数，再调用相应的聚合函数。

```
>>> iris.name.unique().cat(sep=',')
```

返回结果如下。

```
u'Iris-setosa,Iris-versicolor,Iris-virginica'
```

- 如果所有列支持同一种聚合操作，可以直接在整个DataFrame上执行聚合操作。

```
>>> iris.exclude('category').mean()
```

返回结果如下。

```
   sepal_length  sepal_width  petal_length  petal_width
1      5.843333     3.054000      3.758667     1.198667
```

- 使用 `count` 函数获取DataFrame的总行数。

```
>>> iris.count()
```

返回结果如下。

```
150
```

PyODPS支持的聚合操作，如下表所示。

聚合操作	说明
count（或size）	数量。
unique	不重复值数量。
min	最小值。
max	最大值。
sum	求和。
mean	均值。
median	中位数。
quantile(p)	p分位数，仅在整数值下可取得准确值。
var	方差。
std	标准差。
moment	n阶中心矩（或n阶矩）。
skew	样本偏度（无偏估计）。
kurtosis	样本峰度（无偏估计）。
cat	按sep做字符串连接操作。
tolist	组合为LIST。

> 说明 不同于Pandas，对于列上的聚合操作，无论是在MaxCompute还是Pandas后端，PyODPS DataFrame都会忽略空值。这一逻辑与SQL类似。

分组聚合

分组聚合操作如下：

- DataFrame提供了 `groupby` 函数执行分组操作，分组后通过调用 `agg` 或者 `aggregate` 方法，执行聚合操作。最终的结果列中会包含分组的列和聚合的列。

```
>>> iris.groupby('name').agg(iris.sepallength.max(), smin=iris.sepallength.min())
```

返回结果如下。

```
              name  sepallength_max  smin
0      Iris-setosa              5.8   4.3
1  Iris-versicolor              7.0   4.9
2   Iris-virginica              7.9   4.9
```

- DataFrame提供了 `value_counts` 函数，按某列分组后，将每个组的个数从大到小进行排列。
  - 使用 `groupby` 函数实现。

```
>>> iris.groupby('name').agg(count=iris.name.count()).sort('count', ascending=False).head(5)
```

返回结果如下。

```
              name  count
0   Iris-virginica     50
1  Iris-versicolor     50
2      Iris-setosa     50
```

  - 使用 `value_counts` 函数实现。

```
>>> iris['name'].value_counts().head(5)
```

返回结果如下。

```
              name  count
0   Iris-virginica     50
1  Iris-versicolor     50
2      Iris-setosa     50
```

- 对于聚合后的单列操作，您也可以直接取出列名。但此时只能使用聚合函数。

```
>>> iris.groupby('name').petallength.sum()
```

返回结果如下。

```
   petallength_sum
0             73.2
1            213.0
2            277.6
```

```
>>> iris.groupby('name').agg(iris.petallength.notnull().sum())
```

返回结果如下。

```
              name  petallength_sum
0      Iris-setosa               50
1  Iris-versicolor               50
2   Iris-virginica               50
```

- 分组时也支持对常量进行分组，但是需要使用Scalar初始化。

```
>>> from odps.df import Scalar
>>> iris.groupby(Scalar(1)).petallength.sum()
```

返回结果如下。

```
   petallength_sum
0            563.8
```

编写自定义聚合

对字段使用 `agg` 或者 `aggregate` 方法调用自定义聚合。自定义聚合需要提供一个类，这个类需要提供以下方法：

- `buffer()` ：返回一个Mutable的Object（例如LIST或DICT），`buffer` 大小不应随数据量增大而递增。
- `__call__(buffer, *val)` ：将值聚合到中间 `buffer` 。
- `merge(buffer, pbuffer)` ：将 `pbuffer` 聚合到 `buffer` 中。
- `getvalue(buffer)` ：返回最终值。

计算平均值的示例如下。

```
class Agg(object):
    def buffer(self):
        return [0.0, 0]
    def __call__(self, buffer, val):
        buffer[0] += val
        buffer[1] += 1
    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]
    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

```
>>> iris.sepalwidth.agg(Agg)
```

返回结果如下。

```
3.0540000000000007
```

编写自定义聚合还需要关注如下内容：

- 如果最终类型和输入类型发生了变化，则需要指定类型。

  ```
  >>> iris.sepalwidth.agg(Agg, 'float')
  ```

- 自定义聚合也可以用在分组聚合中。

  ```
  >>> iris.groupby('name').sepalwidth.agg(Agg)
  ```

  返回结果如下。

  ```
     petallength_aggregation
  0                    3.418
  1                    2.770
  2                    2.974
  ```

- 对多列可以使用 `agg` 方法调用自定义聚合。

  ```
  class Agg(object):
      def buffer(self):
          return [0.0, 0.0]
      def __call__(self, buffer, val1, val2):
          buffer[0] += val1
          buffer[1] += val2
      def merge(self, buffer, pbuffer):
          buffer[0] += pbuffer[0]
          buffer[1] += pbuffer[1]
      def getvalue(self, buffer):
          if buffer[1] == 0:
              return 0.0
          return buffer[0] / buffer[1]
  ```

  ```
  >>> from odps.df import agg
  >>> to_agg = agg([iris.sepalwidth, iris.sepallength], Agg, rtype='float')  # 对两列调用自定义聚合。
  >>> iris.groupby('name').agg(val=to_agg)
  ```

  返回结果如下。

  ```
               name       val
  0      Iris-setosa  0.682781
  1  Iris-versicolor  0.466644
  2   Iris-virginica  0.451427
  ```

- 如果您需要调用MaxCompute上已经存在的UDAF，指定函数名即可。

  ```
  >>> iris.groupby('name').agg(iris.sepalwidth.agg('your_func'))  # 对单列聚合。
  >>> to_agg = agg([iris.sepalwidth, iris.sepallength], 'your_func', rtype='float')
  >>> iris.groupby('name').agg(to_agg.rename('val'))  # 对多列聚合。
  ```

  > **说明** 目前，因受限于Python UDF，自定义聚合无法支持将LIST或DICT类型作为初始输入或最终输出结果。

HyperLogLog计数

DataFrame提供了对列进行HyperLogLog计数的接口 `hll_count` ，这个接口是近似个数的估计接口。当数据量很大时，它可以较快地估计去重后的数据量。

使用该接口计算海量用户UV时，可以快速得出估计值。

```
>>> df = DataFrame(pd.DataFrame({'a': np.random.randint(100000, size=100000)}))
>>> df.a.hll_count()
```

返回结果如下。

```
63270
```

```
>>> df.a.nunique()
```

返回结果如下。

```
63250
```

> 说明 `splitter` 参数会对每个字段进行分隔，再计算去重后的数据量。

5.6.9. 排序、去重、采样、数据变换

您可以对DataFrame对象执行排序、去重、采样、数据变换操作。

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
```

排序

排序操作只能作用于Collection。

- 调用 `sort` 或者 `sort_values` 方法进行排序。

```
>>> iris.sort('sepalwidth').head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.2         2.2          4.5         1.5  Iris-versicolor
2          6.0         2.2          5.0         1.5   Iris-virginica
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          5.5         2.3          4.0         1.3  Iris-versicolor
```

- 设置参数 `ascending=False;` 进行降序排列。

```
>>> iris.sort('sepalwidth', ascending=False).head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.7         4.4          1.5         0.4  Iris-setosa
1          5.5         4.2          1.4         0.2  Iris-setosa
2          5.2         4.1          1.5         0.1  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.4         3.9          1.3         0.4  Iris-setosa
```

- 设置 `-` 实现降序排列。

```
>>> iris.sort(-iris.sepalwidth).head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.7         4.4          1.5         0.4  Iris-setosa
1          5.5         4.2          1.4         0.2  Iris-setosa
2          5.2         4.1          1.5         0.1  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.4         3.9          1.3         0.4  Iris-setosa
```

- 实现多字段排序。

```
>>> iris.sort(['sepalwidth', 'petallength']).head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          4.0         1.0  Iris-versicolor
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          5.0         1.5   Iris-virginica
4          4.5         2.3          1.3         0.3      Iris-setosa
```

- 多字段排序时，如果是升序降序不同，`ascending`参数可以用于传入一个列表，长度必须等同于排序的字段，它们的值都是BOOLEAN类型。

```
>>> iris.sort(['sepalwidth', 'petallength'], ascending=[True, False]).head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          5.0         1.5   Iris-virginica
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          6.3         2.3          4.4         1.3  Iris-versicolor
```

下面代码可以实现同样的效果。

```
>>> iris.sort(['sepalwidth', -iris.petallength]).head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          5.0         1.5   Iris-virginica
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          6.3         2.3          4.4         1.3  Iris-versicolor
```

> **说明** 由于ODPS要求排序必须指定个数，所以在ODPS后端执行时，会通过`options.df.odps.sort.limit`指定排序个数，这个值默认是10000。如果需要排序尽量多的数据，可以把这个值设到较大的值。但是，这样可能会导致OOM。

去重

- 您可以通过以下三种方式调用`distinct`方法，对Collection进行去重操作。

```
>>> iris[['name']].distinct()
             name
0     Iris-setosa
1 Iris-versicolor
2  Iris-virginica
```

```
>>> iris.distinct('name')
             name
0     Iris-setosa
1 Iris-versicolor
2  Iris-virginica
```

```
>>> iris.distinct('name', 'sepallength').head(3)
          name  sepallength
0 Iris-setosa          4.3
1 Iris-setosa          4.4
2 Iris-setosa          4.5
```

- 您可以调用`unique`对Sequence进行去重操作，但是调用`unique`的Sequence不能用在列选择中。

```
>>> iris.name.unique()
             name
0     Iris-setosa
1 Iris-versicolor
2  Iris-virginica
```

错误示例如下。

```
>>> iris[iris.name, iris.name.unique()]
```

采样

要对一个Collection的数据采样，可以调用`sample`方法。PyODPS支持以下四种采样方式：

> **说明** 除了按份数采样外，其余方法如果要在ODPS DataFrame上执行，需要Project支持XFlow，否则，这些方法只能在Pandas DataFrame后端上执行。

- 按份数采样

  在这种采样方式下，数据被分为`parts`份，可选择选取的份数序号。

```
>>> iris.sample(parts=10)  # 分成10份，默认取第0份。
>>> iris.sample(parts=10, i=0)  # 手动指定取第0份。
>>> iris.sample(parts=10, i=[2, 5])  # 分成10份，取第2和第5份。
>>> iris.sample(parts=10, columns=['name', 'sepalwidth'])  # 根据name和sepalwidth的值做采样。
```

- 按比例、条数采样

在这种采样方式下，用户指定需要采样的数据条数或采样比例。指定replace参数为True可启用放回采样。

```
>>> iris.sample(n=100)  # 选取100条数据。
>>> iris.sample(frac=0.3)  # 采样30%的数据。
```

- 按权重列采样

  在这种采样方式下，您可以指定权重列和数据条数、采样比例。指定 `replace` 参数为True可启用放回采样。

```
>>> iris.sample(n=100, weights='sepal_length')
>>> iris.sample(n=100, weights='sepal_width', replace=True)
```

- 分层采样

  在这种采样方式下，您可以指定用于分层的标签列，同时为需要采样的每个标签指定采样比例（ `frac` 参数）或条数（ `n` 参数）。暂不支持放回采样。

```
>>> iris.sample(strata='category', n={'Iris Setosa': 10, 'Iris Versicolour': 10})
>>> iris.sample(strata='category', frac={'Iris Setosa': 0.5, 'Iris Versicolour': 0.4})
```

数据缩放

DataFrame支持通过最大、最小值或平均值、标准差对数据进行缩放。示例数据如下。

```
name  id  fid
0  name1   4  5.3
1  name2   2  3.5
2  name2   3  1.5
3  name1   4  4.2
4  name1   3  2.2
5  name1   3  4.1
```

- 使用 `min_max_scale` 方法进行归一化。

```
df.min_max_scale(columns=['fid'])
    name  id       fid
0  name1   4  1.000000
1  name2   2  0.526316
2  name2   3  0.000000
3  name1   4  0.710526
4  name1   3  0.184211
5  name1   3  0.684211
```

- `min_max_scale` 还支持使用 `feature_range` 参数指定输出值的范围。例如，需要使输出值在 (-1, 1) 范围内，示例如下。

```
df.min_max_scale(columns=['fid'], feature_range=(-1, 1))
    name  id       fid
0  name1   4  1.000000
1  name2   2  0.052632
2  name2   3 -1.000000
3  name1   4  0.421053
4  name1   3 -0.631579
5  name1   3  0.368421
```

- 如果需要保留原始值，可以使用 `preserve` 参数。此时，缩放后的数据将会以新增列的形式追加到数据中，列名默认为原列名追加_*scaled*后缀，该后缀可使用 `suffix` 参数更改。

```
df.min_max_scale(columns=['fid'], preserve=True)
    name  id  fid  fid_scaled
0  name1   4  5.3    1.000000
1  name2   2  3.5    0.526316
2  name2   3  1.5    0.000000
3  name1   4  4.2    0.710526
4  name1   3  2.2    0.184211
5  name1   3  4.1    0.684211
```

- `min_max_scale` 也支持使用 `group` 参数指定一个或多个分组列，在分组列中分别取最值进行缩放。

```
df.min_max_scale(columns=['fid'], group=['name'])
    name  id       fid
0  name1   4  1.000000
1  name1   4  0.645161
2  name1   3  0.000000
3  name1   3  0.612903
4  name2   2  1.000000
5  name2   3  0.000000
```

结果显示，`name1` 和 `name2` 两组均按组中的最值进行了缩放。

- 使用 `std_scale` 方法可以依照标准正态分布对数据进行调整。`std_scale` 同样支持 `preserve` 参数保留原始列以及使用 `group` 进行分组。

```
df.std_scale(columns=['fid'])
    name  id       fid
0  name1   4  1.436467
1  name2   2  0.026118
2  name2   3 -1.540938
3  name1   4  0.574587
4  name1   3 -0.992468
5  name1   3  0.496234
```

空值处理

DataFrame支持筛去空值以及填充空值的功能。示例数据如下。

```
id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
1   1  name1  2.0  NaN  NaN  1.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
```

- 使用 `dropna` ，删除 `subset` 中包含空值的行。

```
df.dropna(subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   5  name1  1.0  2.0  3.0  4.0
```

- 如果行中包含非空值则不删除，可以使用 `how='all'` 。

```
df.dropna(how='all', subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
1   1  name1  2.0  NaN  NaN  1.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
```

- 使用 `thresh` 参数指定行中至少要有多少个非空值。

```
df.dropna(thresh=3, subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
```

- 调用 `fillna` 方法，使用常数或已有的列来填充未知值。
  - 使用常数填充未知值。

```
df.fillna(100, subset=['f1', 'f2', 'f3', 'f4'])
   id   name     f1     f2     f3     f4
0   0  name1    1.0  100.0    3.0    4.0
1   1  name1    2.0  100.0  100.0    1.0
2   2  name1    3.0    4.0    1.0  100.0
3   3  name1  100.0    1.0    2.0    3.0
4   4  name1    1.0  100.0    3.0    4.0
5   5  name1    1.0    2.0    3.0    4.0
6   6  name1  100.0  100.0  100.0  100.0
```

- 使用一个已有的列填充未知值。

```
df.fillna(df.f2, subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
1   1  name1  2.0  NaN  NaN  1.0
2   2  name1  3.0  4.0  1.0  4.0
3   3  name1  1.0  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
```

- DataFrame提供了向前、向后填充的功能。您可以按照表格中的取值范围为 `method` 参数进行赋值。

取值	含义
bfill、backfill	向前填充。
ffill、pad	向后填充。

示例

```
df.fillna(method='bfill', subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  3.0  3.0  4.0
1   1  name1  2.0  1.0  1.0  1.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  1.0  1.0  2.0  3.0
4   4  name1  1.0  3.0  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
df.fillna(method='ffill', subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  1.0  3.0  4.0
1   1  name1  2.0  2.0  2.0  1.0
2   2  name1  3.0  4.0  1.0  1.0
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  1.0  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
```

您也可以使用 `ffill` 、 `bfill` 函数简化代码。 `ffill` 等价于 `fillna(method='ffill')` ； `bfill` 等价于 `fillna(method='bfill')` 。

5.6.10. 使用自定义函数及Python第三方库

本文为您介绍如何使用自定义函数及Python第三方库。

使用自定义函数

DataFrame函数支持对Sequence使用 `map` ，它会对它的每个元素调用自定义函数。

```
>>> iris.sepallength.map(lambda x: x + 1).head(5)
   sepallength
0          6.1
1          5.9
2          5.7
3          5.6
4          6.0
```

> **说明** 目前，自定义函数无法支持将List/Dict类型作为输入或输出。

如果 `map` 前后，Sequence的类型发生了变化，则需要显式指定 `map` 后的类型。

```
>>> iris.sepallength.map(lambda x: 't'+str(x), 'string').head(5)
   sepallength
0         t5.1
1         t4.9
2         t4.7
3         t4.6
4         t5.0
```

如果在函数中包含闭包，则函数外闭包变量值的变化会引起函数内该变量值的变化。

```
>>> dfs = []
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(lambda x: x + i))
```

结果为 `dfs` 中每个 `SequenceExpr` 均为 `df.sepal_length+9` 。为解决此问题，可以将函数作为另一函数的返回值，或者使用 `partial` 。两个示例如下。

```
>>> dfs = []
>>> def get_mapper(i):
>>>     return lambda x: x + i
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(get_mapper(i)))
```

```
>>> import functools
>>> dfs = []
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(functools.partial(lambda v, x: x + v, i)))
```

`map` 也支持使用现有的UDF函数，传入的参数是str类型（函数名）或者Function对象，详情请参见函数。

`map` 传入Python函数的实现使用了MaxCompute Python UDF。因此，如果您所在的Project不支持Python UDF，则 `map` 函数无法使用。除此以外，所有Python UDF的限制在此都适用。

目前，默认可使用的第三方库（包含C）只有numpy，第三方库使用详情请参见使用第三方Python库。

除了调用自定义函数，DataFrame还提供了很多内置函数，这些函数中部分使用了 `map` 函数来实现。因此，如果您所在Project未开通Python UDF，则无法使用这些函数（注意：阿里云公共服务暂不提供对Python UDF的支持）。

> **说明** 由于字节码定义的差异，Python 3下使用新语言特性（例如 `yield from` ）时，代码在使用Python 2.7的MaxCompute Worker上执行时会发生错误。因此，建议您在Python 3下使用MapReduce API编写生产作业前，先确认相关代码是否能正常执行。

示例程序：使用计数器

```
from odps.udf import get_execution_context
def h(x):
    ctx = get_execution_context()
    counters = ctx.get_counters()
    counters.get_counter('df', 'add_one').increment(1)
    return x + 1
df.field.map(h)
```

Logview的JSONSummary中即可找到计数器值。

对一行数据使用自定义函数

如果您需要对一行数据使用自定义函数，可以使用 `apply` 方法。参数 `axis` 的值必须设为1，表示对行进行操作。 `apply` 的自定义函数接收一个参数，参数为上一步Collection的一行数据。您可以通过属性或者偏移获得一个字段的数据。

- `reduce` 为True时，表示返回结果为Sequence，否则返回结果为Collection。 `names` 和 `types` 参数分别指定返回的Sequence或Collection的字段名和类型。 如果未指定类型，则会默认为STRING类型。

  ```
  >>> iris.apply(lambda row: row.sepallength + row.sepalwidth, axis=1, reduce=True, types='float').rename('sepaladd').head(3)
     sepaladd
  0       8.6
  1       7.9
  2       7.9
  ```

- 在 `apply` 的自定义函数中， `reduce` 为False时，您可以使用 `yield` 关键字返回多行结果。

  ```
  >>> iris.count()
  150
  >>>
  >>> def handle(row):
  >>>     yield row.sepallength - row.sepalwidth, row.sepallength + row.sepalwidth
  >>>     yield row.petallength - row.petalwidth, row.petallength + row.petalwidth
  >>>
  >>> iris.apply(handle, axis=1, names=['iris_add', 'iris_sub'], types=['float', 'float']).count()
  300
  ```

- 您也可以在函数中注释返回的字段和类型，无需在函数调用时再指定它们。

```
>>> from odps.df import output
>>>
>>> @output(['iris_add', 'iris_sub'], ['float', 'float'])
>>> def handle(row):
>>>     yield row.sepallength - row.sepalwidth, row.sepallength + row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength + row.petalwidth
>>>
>>> iris.apply(handle, axis=1).count()
300
```

- 您也可以使用 `map-only` 的 `map_reduce` ，该操作与 `axis=1` 的 `apply` 操作是等价的。

```
>>> iris.map_reduce(mapper=handle).count()
300
```

- 如果您想调用MaxCompute上已经存在的UDTF，函数指定为函数名即可。

```
>>> iris['name', 'sepallength'].apply('your_func', axis=1, names=['name2', 'sepallength2'], types=['string', 'float'])
```

- 使用 `apply` 对行操作，且 `reduce` 为False时，您可以使用并列多行输出与已有的行结合，用于后续聚合等操作。

```
>>> from odps.df import output
>>>
>>> @output(['iris_add', 'iris_sub'], ['float', 'float'])
>>> def handle(row):
>>>     yield row.sepallength - row.sepalwidth, row.sepallength + row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength + row.petalwidth
>>>
>>> iris[iris.category, iris.apply(handle, axis=1)]
```

对所有列调用自定义聚合

调用 `apply` 方法，当不指定 `axis` ，或者 `axis` 值为0时，您可以通过传入一个自定义聚合类对所有Sequence进行聚合操作。

```
class Agg(object):
    def buffer(self):
        return [0.0, 0]
    def __call__(self, buffer, val):
        buffer[0] += val
        buffer[1] += 1
    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]
    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

```
>>> iris.exclude('name').apply(Agg)
   sepallength_aggregation  sepalwidth_aggregation  petallength_aggregation  petalwidth_aggregation
0                 5.843333                   3.054                 3.758667                1.198667
```

> **说明** 目前，自定义函数无法支持将LIST/DICT类型作为初始输入或最终输出结果。

引用资源

自定义函数也能读取MaxCompute上的资源（表资源或文件资源），或者引用一个Collection作为资源。此时，自定义函数需要写成函数闭包或Callable的类。两个示例如下。

```
>>> file_resource = o.create_resource('pyodps_iris_file', 'file', file_obj='Iris-setosa')
>>>
>>> iris_names_collection = iris.distinct('name')[:2]
>>> iris_names_collection
       sepallength
0      Iris-setosa
1  Iris-versicolor
```

```
>>> def myfunc(resources):  # resources按调用顺序传入。
>>>     names = set()
>>>     fileobj = resources[0] # 文件资源是一个file-like的object。
>>>     for l in fileobj:
>>>         names.add(l)
>>>     collection = resources[1]
>>>     for r in collection:
>>>         names.add(r.name)  # 这里可以通过字段名或者偏移来取。
>>>     def h(x):
>>>         if x in names:
>>>             return True
>>>         else:
>>>             return False
>>>     return h
>>>
>>> df = iris.distinct('name')
>>> df = df[df.name,
>>>         df.name.map(myfunc, resources=[file_resource, iris_names_collection], rtype='boolean').rename('isin')]
>>>
>>> df
             name   isin
0     Iris-setosa   True
1 Iris-versicolor   True
2  Iris-virginica  False
```

> **说明** 分区表资源在读取时不包含分区字段。

当 `axis` 值为1，即在行上操作时，您需要写一个函数闭包或者Callable的类。 而对于列上的聚合操作，您只需在 `__init__` 函数里读取资源即可。

```
>>> words_df
                    sentence
0                Hello World
1               Hello Python
2  Life is short I use Python
>>>
>>> import pandas as pd
>>> stop_words = DataFrame(pd.DataFrame({'stops': ['is', 'a', 'I']}))
>>>
>>> @output(['sentence'], ['string'])
>>> def filter_stops(resources):
>>>     stop_words = set([r[0] for r in resources[0]])
>>>     def h(row):
>>>         return ' '.join(w for w in row[0].split() if w not in stop_words),
>>>     return h
>>>
>>> words_df.apply(filter_stops, axis=1, resources=[stop_words])
               sentence
0           Hello World
1          Hello Python
2  Life short use Python
```

> **说明** 这里的 `stop_words` 存放于本地，但在真正执行时会被上传到MaxCompute作为资源引用。

使用第三方Python库

您可以把第三方Python包作为资源上传到MaxCompute，支持的格式有*whl*、*egg*、*zip*以及*tar.gz*。在全局或者在立即执行的方法时，指定需要使用的包文件，即可以在自定义函数中使用第三方库。第三方库的依赖库，也必须指定，否则依然会导致导入错误。

- 您可以通过PyODPS的资源上传接口 `create_resource` 来完成资源的上传。

  下面以*python-dateutil*包进行举例：

  i. 使用 `pip download` 命令，下载包以及其依赖到某个路径。下载后会出现两个包：*six-1.10.0-py2.py3-none-any.whl*和*python_dateutil-2.5.3-py2.py3-none-any.whl*（注意：这里需要下载支持Linux环境的包）。

  ```
  $ pip download python-dateutil -d /to/path/
  ```

  ii. 分别把两个文件上传到MaxCompute资源。

```
>>> # 这里要确保资源名的后缀是正确的文件类型。
>>> odps.create_resource('six.whl', 'file', file_obj=open('six-1.10.0-py2.py3-none-any.whl', 'rb'))
>>> odps.create_resource('python_dateutil.whl', 'file', file_obj=open('python_dateutil-2.5.3-py2.py3-none-any.whl', 'rb'))
```

iii. 现在有个DataFrame，只有一个STRING类型字段。

```
>>> df
              datestr
0  2016-08-26 14:03:29
1  2015-08-26 14:03:29
```

iv. 全局配置使用到的三方库如下。

```
>>> from odps import options
>>>
>>> def get_year(t):
>>>     from dateutil.parser import parse
>>>     return parse(t).strftime('%Y')
>>>
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
>>> df.datestr.map(get_year)
   datestr
0     2016
1     2015
```

或者通过运行方法的 `libraries` 参数指定使用到的第三方库。

```
>>> def get_year(t):
>>>     from dateutil.parser import parse
>>>     return parse(t).strftime('%Y')
>>>
>>> df.datestr.map(get_year).execute(libraries=['six.whl', 'python_dateutil.whl'])
   datestr
0     2016
1     2015
```

PyODPS默认支持执行仅包含Python且不含文件操作的第三方库。在较新版本的MaxCompute服务下，PyODPS也支持执行带有二进制代码或带有文件操作的Python库。这些库的后缀必须是*cp27-cp27m-manylinux1_x86_64*，以 `archive` 方式上传，*whl*后缀的包需要重命名为*zip*。同时，作业需要开启 `odps.isolation.session.enable` 选项，或者在Project级别开启 `isolation` 。以下示例为您展示如何上传并使用 `scipy` 中的特殊函数。

```
>>> # 对于含有二进制代码的包，必须使用archive方式上传资源，whl后缀需要改为zip。
>>> odps.create_resource('scipy.zip', 'archive', file_obj=open('scipy-0.19.0-cp27-cp27m-manylinux1_x86_64.whl', 'rb'))
>>>
>>> # 如果Project开启了isolation，下面的选项不是必需的。
>>> options.sql.settings = { 'odps.isolation.session.enable': True }
>>>
>>> def psi(value):
>>>     # 建议在函数内部import第三方库，以防止不同操作系统下二进制包结构差异造成执行错误。
>>>     from scipy.special import psi
>>>     return float(psi(value))
>>>
>>> df.float_col.map(psi).execute(libraries=['scipy.zip'])
```

对于只提供源码的二进制包，可以在Linux Shell中打包成Wheel再上传，Mac和Windows中生成的Wheel包无法在MaxCompute中使用。

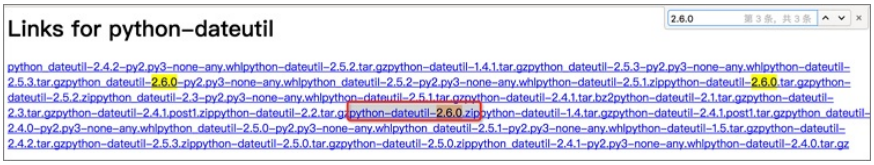
```
python setup.py bdist_wheel
```

- 您也可以通过MaxCompute Console上传资源。
  i. 现在主流的Python包都提供了whl包，提供了各平台包含二进制文件的包，因此找到可以在MaxCompute上运行的包是第一步。
  ii. 其次，要想在MaxCompute上运行，需要包含所有的依赖包，这个是比较繁琐的。各个包的依赖情况如下表所示。

包名	依赖
pandas	numpy、python-dateutil、pytz、six
scipy	numpy
scikit-learn	numpy、scipy

> **说明** 其中numpy已包含，您只需上传python-dateutil、pytz、pandas、scipy、sklearn、six包，pandas、scipy和scikit-learn即可使用。

  iii. 您可进入python-dateutils找到python-dateutil-2.6.0.zip进行下载。



iv. 重命名为*python-dateutil.zip*，通过MaxCompute Console上传资源。

```
add archive python-dateutil.zip;
```

> **说明** pytz和six的上传方式同上，分别找到 pytz-2017.2.zip和six-1.11.0.tar.gz进行下载和上传资源操作。

v. 对于Pandas这种包含c的包，需要找到名字中包含cp27-cp27m-manylinux1_x86_64的whl包，这样才能在MaxCompute上正确执行。因此，您需要找到pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.whl进行下载，然后把后缀改成zip，在MaxCompute Console中执行 `add archive pandas.zip;` 进行上传。scipy和scikit-learn包的操作同上。

所有包需要下载的资源如下表所示。

包名	文件名	上传资源名
python-dateutil	python-dateutil-2.6.0.zip	python-dateutil.zip
pytz	pytz-2017.2.zip	pytz.zip
six	six-1.11.0.tar.gz	six.tar.gz
pandas	pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.zip	pandas.zip
scipy	scipy-0.19.0-cp27-cp27m-manylinux1_x86_64.zip	scipy.zip
scikit-learn	scikit_learn-0.18.1-cp27-cp27m-manylinux1_x86_64.zip	sklearn.zip

指定第三方Python库

- 在全局指定使用的库。

```
>>> from odps import options
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

- 在立即执行的方法中，局部指定使用的库。

```
>>> df.apply(my_func, axis=1).to_pandas(libraries=['six.whl', 'python_dateutil.whl'])
```

5.6.11. MapReduce API

本文为您介绍MapReduce API。

PyODPS DataFrame支持MapReduce API，您可以分别编写 `map` 和 `reduce` 函数（ `map_reduce` 可以只有 `mapper` 或者 `reducer` 过程）。 `wordcount` 的示例如下。

```
>>> #encoding=utf-8
>>> from odps import ODPS
>>> from odps import options
>>> options.verbose = True
>>> o = ODPS('your-access-id', 'your-secret-access-key',project='DMP_UC_dev', endpoint='http://service-corp.odps.aliyun-inc.com/a
pi')
>>> from odps.df import DataFrame
>>> def mapper(row):
>>>     for word in row[0].split():
>>>         yield word.lower(), 1
>>>
>>> def reducer(keys):
>>>     # 这里使用list而不是cnt=0，否则h内的cnt会被认为是局部变量，其中的赋值无法输出。
>>>     cnt = [0]
>>>     def h(row, done):  # done表示这个key已经迭代结束。
>>>         cnt[0] += row[1]
>>>         if done:
>>>             yield keys[0], cnt[0]
>>>     return h
>>> # zx_word_count表只有一列，为STRING类型。
>>> word_count = DataFrame(o.get_table('zx_word_count'))
>>> table = word_count.map_reduce(mapper, reducer, group=['word', ],
                       mapper_output_names=['word', 'cnt'],
                       mapper_output_types=['string', 'int'],
                       reducer_output_names=['word', 'cnt'],
                       reducer_output_types=['string', 'int'])
         word  cnt
0         are    1
1         day    1
2      doing?    1
3   everybody    1
4       first    1
5       hello    2
6         how    1
7          is    1
8          so    1
9         the    1
10       this    1
11      world    1
12        you    1
```

`group` 参数用于指定 `reduce` 按哪些字段做分组，如果不指定，会按全部字段做分组。 `reducer` 需要接收聚合的 `keys` 进行初始化，并能继续处理按这些 `keys` 聚合的每行数据。 `done` 表示与这些 `keys` 相关的所有行是否都迭代完成。

为了方便，此处写成了函数闭包的方式，您也可以写成 `Callable` 的类。

```
class reducer(object):
    def __init__(self, keys):
        self.cnt = 0
    def __call__(self, row, done):  # done表示这个key已经迭代结束。
        self.cnt += row.cnt
        if done:
            yield row.word, self.cnt
```

使用 `output` 进行注释会让代码更简单。

```
>>> from odps.df import output
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def mapper(row):
>>>     for word in row[0].split():
>>>         yield word.lower(), 1
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def reducer(keys):
>>>     # 此处使用list而不是cnt=0，否则h内的cnt会被认为是局部变量，其中的赋值无法输出。
>>>     cnt = [0]
>>>     def h(row, done):  # done表示这个key已经迭代结束。
>>>         cnt[0] += row.cnt
>>>         if done:
>>>             yield keys.word, cnt[0]
>>>     return h
>>>
>>> word_count = DataFrame(o.get_table('zx_word_count'))
>>> table = word_count.map_reduce(mapper, reducer, group='word')
         word  cnt
0         are    1
1         day    1
2      doing?    1
3   everybody    1
4       first    1
5       hello    2
6         how    1
7          is    1
8          so    1
9         the    1
10       this    1
11      world    1
12        you    1
```

在迭代的时候，可以使用 `sort` 参数实现按指定列排序，通过 `ascending` 参数指定升序降序。 `ascending` 参数可以是一个BOOL值，表示所有的 `sort` 字段是相同升序或降序，也可以是一个列表，长度必须和 `sort` 字段长度相同。

指定COMBINER

`combiner` 表示在 `map_reduce` API里表示在 `mapper` 端，就先对数据进行聚合操作，它的用法和 `reducer` 是完全一致的，但不能引用资源。 并且， `combiner` 的输出的字段名和字段类型必须和 `mapper` 完全一致。

上面的例子，您可以使用 `reducer` 作为 `combiner` 在 `mapper` 端对数据做初步的聚合，减少Shuffle出去的数据量。

```
>>> words_df.map_reduce(mapper, reducer, combiner=reducer, group='word')
```

引用资源

在MapReduce API里，您可以分别指定 `mapper` 和 `reducer` 所要引用的资源。

以下示例为对 `mapper` 里的单词做停词过滤，在 `reducer` 里对白名单的单词数量加5。

```
>>> white_list_file = o.create_resource('pyodps_white_list_words', 'file', file_obj='Python\nWorld')
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def mapper(resources):
>>>     stop_words = set(r[0].strip() for r in resources[0])
>>>     def h(row):
>>>         for word in row[0].split():
>>>             if word not in stop_words:
>>>                 yield word, 1
>>>     return h
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def reducer(resources):
>>>     d = dict()
>>>     d['white_list'] = set(word.strip() for word in resources[0])
>>>     d['cnt'] = 0
>>>     def inner(keys):
>>>         d['cnt'] = 0
>>>         def h(row, done):
>>>             d['cnt'] += row.cnt
>>>             if done:
>>>                 if row.word in d['white_list']:
>>>                     d['cnt'] += 5
>>>                 yield keys.word, d['cnt']
>>>         return h
>>>     return inner
>>>
>>> words_df.map_reduce(mapper, reducer, group='word',
>>>                     mapper_resources=[stop_words], reducer_resources=[white_list_file])
     word  cnt
0   hello    2
1    life    1
2  python    7
3   world    6
4   short    1
5     use    1
```

使用第三方Python库

使用方法类似在MAP中使用第三方Python库。

- 在全局指定使用的库。

```
>>> from odps import options
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

- 在立即执行的方法中，局部指定使用的库。

```
>>> df.map_reduce(mapper=my_mapper, reducer=my_reducer, group='key').execute(libraries=['six.whl', 'python_dateutil.whl'])
```

> **说明** 由于字节码定义的差异，Python 3下使用新语言特性（例如 `yield from` ）时，代码在使用 Python 2.7的ODPS Worker上执行时会发生错误。因此，建议您在Python 3下使用MapReduce API编写生产作业前，先确认相关代码是否能正常执行。

重排数据

当数据在集群上分布不均匀时，您可以调用 `reshuffle` 接口对数据重排。

```
>>> df1 = df.reshuffle()
```

默认会按随机数做哈希来分布，但您也可以按指定列做分布，并且可以指定重排后的排序顺序。

```
>>> df1.reshuffle('name', sort='id', ascending=False)
```

布隆过滤器

PyODPS DataFrame提供了 `bloom_filter` 接口进行布隆过滤器的计算。

给定某个Collection和它的某个列计算的 `sequence1` ，对另外一个 `sequence2` 进行布隆过滤时， `sequence1` 不存在于 `sequence2` 中的数据一定会被过滤掉， 但可能无法完全过滤。所以这种过滤方式是一种近似的过滤方法。这样的好处是对Collection进行快速过滤一些无用数据。这在一边数据量远大过另一边数据量（大部分数据并不会参与 `join` 运算）时的大规模 `join` 场景很有用。例如，在 `join` 用户的浏览数据和交易数据时，用户的浏览数据量远大于交易数据量，可以利用交易数据先对浏览数据进行布隆过滤， 然后再 `join` 可以很好地提升性能。

```
>>> df1 = DataFrame(pd.DataFrame({'a': ['name1', 'name2', 'name3', 'name1'], 'b': [1, 2, 3, 4]}))
>>> df1
       a  b
0  name1  1
1  name2  2
2  name3  3
3  name1  4
>>> df2 = DataFrame(pd.DataFrame({'a': ['name1']}))
>>> df2
       a
0  name1
>>> df1.bloom_filter('a', df2.a) # 这里第0个参数可以是个计算表达式如：df1.a + '1'。
       a  b
0  name1  1
1  name1  4
```

示例说明：

- 由于数据量很小，`df1` 中的 `a` 为 `name2` 和 `name3` 的行都被正确过滤掉了，当数据量很大的时候，可能无法过滤全部数据。
- 之前提 `join` 场景中，少量数据不能被过滤的情况并不会影响数据的正确性，但可以较大地提升 `join` 的性能。
- 您可以传入 `capacity` 和 `error_rate` 来设置数据的量以及错误率，默认值是 `3000` 和 `0.01` 。

> 说明 要注意，调大 `capacity` 或者减小 `error_rate` 会增加内存的使用，所以应当根据实际情况选择一个合理的值。

Collection对象操作请参见DataFrame执行。

透视表（PIVOT_TABLE）

PyODPS DataFrame提供了透视表的功能。示例的表数据如下。

```
>>> df
     A    B      C  D  E
0  foo  one  small  1  3
1  foo  one  large  2  4
2  foo  one  large  2  5
3  foo  two  small  3  6
4  foo  two  small  3  4
5  bar  one  large  4  5
6  bar  one  small  5  3
7  bar  two  small  6  2
8  bar  two  large  7  1
```

- 使用透视表功能时，`rows` 参数为必选参数，表示按一个或者多个字段做取平均值的操作。

```
>>> df['A', 'D', 'E'].pivot_table(rows='A')
     A  D_mean  E_mean
0  bar     5.5    2.75
1  foo     2.2    4.40
```

- `rows` 可以提供多个字段，表示按多个字段做聚合。

```
>>> df.pivot_table(rows=['A', 'B', 'C'])
     A    B      C  D_mean  E_mean
0  bar  one  large     4.0     5.0
1  bar  one  small     5.0     3.0
2  bar  two  large     7.0     1.0
3  bar  two  small     6.0     2.0
4  foo  one  large     2.0     4.5
5  foo  one  small     1.0     3.0
6  foo  two  small     3.0     5.0
```

- 指定 `values` 显示指定要计算的列。

```
>>> df.pivot_table(rows=['A', 'B'], values='D')
     A    B    D_mean
0  bar  one  4.500000
1  bar  two  6.500000
2  foo  one  1.666667
3  foo  two  3.000000
```

- 计算值列时，默认会计算平均值。通过 `aggfunc` 指定一个或者多个聚合函数。

```
>>> df.pivot_table(rows=['A', 'B'], values=['D'], aggfunc=['mean', 'count', 'sum'])
     A    B    D_mean  D_count  D_sum
0  bar  one  4.500000        2      9
1  bar  two  6.500000        2     13
2  foo  one  1.666667        3      5
3  foo  two  3.000000        2      6
```

- 将原始数据的某一列的值，作为新的Collection的列。

```
>>> df.pivot_table(rows=['A', 'B'], values='D', columns='C')
     A    B  large_D_mean  small_D_mean
0  bar  one           4.0           5.0
1  bar  two           7.0           6.0
2  foo  one           2.0           1.0
3  foo  two           NaN           3.0
```

- 使用 `fill_value` 填充空值。

```
>>> df.pivot_table(rows=['A', 'B'], values='D', columns='C', fill_value=0)
     A    B  large_D_mean  small_D_mean
0  bar  one             4             5
1  bar  two             7             6
2  foo  one             2             1
3  foo  two             0             3
```

KEY-VALUE字符串转换

DataFrame提供了将Key-Value对展开为列，以及将普通列转换为Key-Value列的功能。DataFrame创建及其对象操作请参见创建DataFrame。

- 将Key-Value对展开为列，示例数据如下。

```
>>> df
    name               kv
0  name1  k1=1,k2=3,k5=10
1  name1    k1=7.1,k7=8.2
2  name2    k2=1.2,k3=1.5
3  name2      k9=1.1,k2=1
```

通过 `extract_kv` 方法将Key-Value字段展开。

```
>>> df.extract_kv(columns=['kv'], kv_delim='=', item_delim=',')
   name   kv_k1  kv_k2  kv_k3  kv_k5  kv_k7  kv_k9
0  name1    1.0    3.0    NaN   10.0    NaN    NaN
1  name1    7.0    NaN    NaN    NaN    8.2    NaN
2  name2    NaN    1.2    1.5    NaN    NaN    NaN
3  name2    NaN    1.0    NaN    NaN    NaN    1.1
```

其中，需要展开的字段名由 `columns` 指定，Key和Value之间的分隔符，以及Key-Value对之间的分隔符分别由 `kv_delim` 和 `item_delim` 这两个参数指定，默认分别为半角冒号和半角逗号。输出的字段名为原字段名和Key值的组合，通过 `_` 相连。缺失值默认为NONE，可通过 `fill_value` 选择需要填充的值。

```
>>> df.extract_kv(columns=['kv'], kv_delim='=', fill_value=0)
   name   kv_k1  kv_k2  kv_k3  kv_k5  kv_k7  kv_k9
0  name1    1.0    3.0    0.0   10.0    0.0    0.0
1  name1    7.0    0.0    0.0    0.0    8.2    0.0
2  name2    0.0    1.2    1.5    0.0    0.0    0.0
3  name2    0.0    1.0    0.0    0.0    0.0    1.1
```

- 将多列数据转换为一个Key-Value列，示例数据如下。

```
>>> df
   name    k1   k2   k3    k5   k7   k9
0  name1  1.0  3.0  NaN  10.0  NaN  NaN
1  name1  7.0  NaN  NaN   NaN  8.2  NaN
2  name2  NaN  1.2  1.5   NaN  NaN  NaN
3  name2  NaN  1.0  NaN   NaN  NaN  1.1
```

通过 `to_kv` 方法转换为Key-Value表示的格式。

```
>>> df.to_kv(columns=['k1', 'k2', 'k3', 'k5', 'k7', 'k9'], kv_delim='=')
    name               kv
0  name1  k1=1,k2=3,k5=10
1  name1    k1=7.1,k7=8.2
2  name2    k2=1.2,k3=1.5
3  name2      k9=1.1,k2=1
```

5.6.12. 数据合并

本文向您介绍DataFrame支持的数据表的JOIN操作、UNION操作等数据合并操作。

导入表数据，创建示例数据。

```
>>> from odps.df import DataFrame
>>> movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
>>> ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
>>> movies.dtypes
odps.Schema {
  movie_id                       int64
  title                          string
  release_date                   string
  video_release_date             string
  _url                         string
}
>>> ratings.dtypes
odps.Schema {
  user_id                  int64
  movie_id                 int64
  rating                   int64
  unix_timestamp           int64
}
```

JOIN操作

DataFrame支持对两个Collection执行 `join` 的操作。

- 如果不指定 `join` 的条件，则DataFrame API会寻找名字相同的列，并作为 `join` 的条件。

```
>>> movies.join(ratings).head(3)
   movie_id             title  release_date  video_release_date                                                 url  user_id  ratin
g  unix_timestamp
0         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
49       3       888068877
1         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
621       5       881444887
2         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
291       3       874833936
```

- 显式指定 `join` 的条件。在 `join` 时，如果 `on` 条件在两表中所选取的字段名称相同，则只会保留其中一个表的此字段。

```
>>> movies.join(ratings, on='movie_id').head(3)
   movie_id             title  release_date  video_release_date                                                  url  user_id  rati
ng  unix_timestamp
0         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
49       3       888068877
1         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
621       5       881444887
2         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
291       3       874833936
```

- 在执行其他类型的 `join` 操作时，例如 `left_join` ，两表中相同名称的字段会被重命名。

```
>>> movies.left_join(ratings, on='movie_id').head(3)
   movie_id_x             title  release_date  video_release_date                                                   url  user_id  mov
ie_id_y  rating  unix_timestamp
0           3  Four Rooms (1995)   01-Jan-1995                     http://example.aliyundoc.com/M/title-exact?Four%20Rooms%..
.       49           3       3       888068877
1           3  Four Rooms (1995)   01-Jan-1995                     http://example.aliyundoc.com/M/title-exact?Four%20Rooms%..
.      621           3       5       881444887
2           3  Four Rooms (1995)   01-Jan-1995                     http://example.aliyundoc.com/M/title-exact?Four%20Rooms%..
.      291           3       3       874833936
```

示例中， `movie_id` 被重命名为 `movie_id_x` 、 `movie_id_y` ，这与 `suffixes` 参数有关，该参数的默认值为 `('_x', '_y')` 。当遇到重名的列时，重名的列会被重命名为指定的后缀名。

```
>>> ratings2 = ratings[ratings.exclude('movie_id'), ratings.movie_id.rename('movie_id2')]
>>> ratings2.dtypes
odps.Schema {
  user_id                  int64
  rating                   int64
  unix_timestamp           int64
  movie_id2                int64
}
>>> movies.join(ratings2, on=[('movie_id', 'movie_id2')]).head(3)
   movie_id             title  release_date  video_release_date                                                  url  user_id  ratin
g  unix_timestamp  movie_id2
0         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
49       3       888068877          3
1         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
621       5       881444887          3
2         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
291       3       874833936          3
```

- 也可以在 `on` 中直接写等于表达式。

```
>>> movies.join(ratings2, on=[movies.movie_id == ratings2.movie_id2]).head(3)
   movie_id             title  release_date  video_release_date                                                  url  user_id  ratin
g  unix_timestamp  movie_id2
0         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
49       3       888068877          3
1         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
621       5       881444887          3
2         3  Four Rooms (1995)   01-Jan-1995                   http://example.aliyundoc.com/M/title-exact?Four%20Rooms%...
291       3       874833936          3
```

- 在 `self-join` 时，可以调用 `view` 方法分别取字段。

```
>>> movies2 = movies.view()
>>> movies.join(movies2, movies.movie_id == movies2.movie_id)[movies, movies2.movie_id.rename('movie_id2')].head(3)
   movie_id           title_x release_date_x video_release_date_x  \
0         2   GoldenEye (1995)    01-Jan-1995                True
1         3  Four Rooms (1995)    01-Jan-1995                True
2         4  Get Shorty (1995)    01-Jan-1995                True
                                     url_x  movie_id2
0  http://example.aliyundoc.com/M/title-exact?GoldenEye%20(...          2
1  http://example.aliyundoc.comtitle-exact?Four%20Rooms%...          3
2  http://example.aliyundoc.com/M/title-exact?Get%20Shorty%...          4
```

除了支持 `join`，DataFrame还支持 `left_join`、`right_join` 和 `outer_join`。在执行上述外连接操作时，默认会为重名列加上 `_x` 和 `_y` 后缀，可通过在 `suffixes` 参数中传入一个二元 `tuple` 进行自定义后缀。

- 如果需要在外连接中避免对位次相等的列取重复列，可以指定 `merge_columns` 选项。该选项会自动选择两列中的非空值作为新列的值。

```
>>> movies.left_join(ratings, on='movie_id', merge_columns=True)
```

使用 `mapjoin` 时，只需将 `mapjoin` 设为True，执行时会对右表做 `mapjoin` 操作。您也可以对分别来自ODPS和Pandas的Collection进行 `join`，或者对分别来自ODPS和数据库的Collection进行 `join`，该计算会在ODPS上执行。

UNION操作

如果两张表的字段和类型都一致（字段的顺序可以不同），则可以使用 `union` 或者 `concat` 将它们合并成一张表。

```
>>> mov1 = movies[movies.movie_id < 3]['movie_id', 'title']
>>> mov2 = movies[(movies.movie_id > 3) & (movies.movie_id < 6)]['title', 'movie_id']
>>> mov1.union(mov2)
   movie_id             title
0         1   Toy Story (1995)
1         2   GoldenEye (1995)
2         4  Get Shorty (1995)
3         5     Copycat (1995)
```

您也可以对分别来自ODPS和Pandas的Collection进行 `union` 操作，或者对分别来自ODPS和数据库的Collection进行 `union` 操作，相关的计算会在ODPS上执行。

5.6.13. 窗口函数

本文为您介绍DataFrame API支持使用窗口函数。

```
grouped = iris.groupby('name')
grouped.mutate(grouped.sepallength.cumsum(), grouped.sort('sepallength').row_number()).head(10)
          name  sepallength_sum  row_number
0  Iris-setosa            250.3           1
1  Iris-setosa            250.3           2
2  Iris-setosa            250.3           3
3  Iris-setosa            250.3           4
4  Iris-setosa            250.3           5
5  Iris-setosa            250.3           6
6  Iris-setosa            250.3           7
7  Iris-setosa            250.3           8
8  Iris-setosa            250.3           9
9  Iris-setosa            250.3          10
```

- 窗口函数可以用在列选择中。

```
iris['name', 'sepallength', iris.groupby('name').sort('sepallength').sepallength.cumcount()].head(5)
          name  sepallength  sepallength_count
0  Iris-setosa          4.3                  1
1  Iris-setosa          4.4                  2
2  Iris-setosa          4.4                  3
3  Iris-setosa          4.4                  4
4  Iris-setosa          4.5                  5
```

- 窗口函数按标量聚合时，与分组聚合的处理方式一致。

```
from odps.df import Scalar
iris.groupby(Scalar(1)).sort('sepallength').sepallength.cumcount()
```

DataFrame API支持的窗口函数如下。

窗口函数	说明
cumsum	计算累积和。
cummean	计算累积均值。
cummedian	计算累积中位数。
cumstd	计算累积标准差。
cummax	计算累积最大值。
cummin	计算累积最小值。
cumcount	计算累积和。
lag	按偏移量取当前行之前第几行的值。如果当前行号为rn，则取行号为rn-offset的值。
lead	按偏移量取当前行之后第几行的值。如果当前行号为rn则取行号为rn+offset的值。
rank	计算排名。
dense_rank	计算连续排名。
percent_rank	计算一组数据中某行的相对排名。
row_number	计算行号，从1开始。
qcut	将分组数据按顺序切分成n片，并返回当前切片值。如果切片不均匀，默认增加第一个切片的分布。
nth_value	返回分组中的第n个值。
cume_dist	计算分组中值小于等于当前值的行数占分组总行数的比例。

各个窗口函数支持的参数说明如下。

- `rank`、`dense_rank`、`percent_rank`和`row_number`支持下列参数。

参数名	说明
sort	排序关键字，默认为空。

参数名	说明
ascending	排序时，是否依照升序排序，默认值为True。

- `lag` 和 `lead` 除了支持 `rank` 所支持的参数，还支持下列参数。

参数名	说明
offset	取数据的行距离当前行的行数。
default	当offset指定的行不存在时的返回值。

- `cumsum` 、`cummax` 、`cummin` 、`cummean` 、`cummedian` 、`cumcount` 和 `cumstd` 除了支持 `rank` 所支持的参数外，还支持下列参数。

参数名	说明
unique	是否排除重复值，默认False。
preceding	窗口范围起点。
following	窗口范围终点。

5.6.14. 绘图

本文为您介绍PyODPS DataFrame提供的绘图方法。

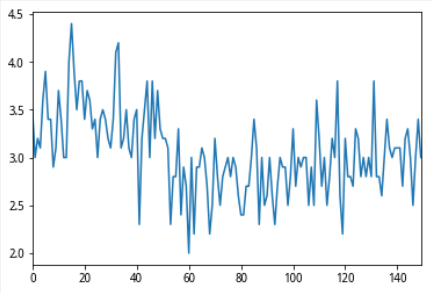
如果您需要使用绘图功能，请先安装Pandas和Matplotlib。

您可以在Jupyter中运行以下示例代码，并使用 `pip install matplotlib` 命令安装Matplotlib。

绘图

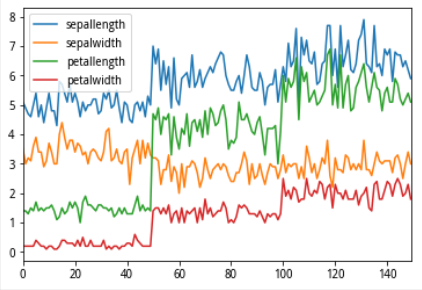
- 单线图

```
>>> from odps.df import DataFrame
>>> iris = DataFrame(o.get_table('pyodps_iris'))
>>> %matplotlib inline
>>> iris.sepalwidth.plot()
<matplotlib.axes._subplots.AxesSubplot at 0x10c2b3510>
```



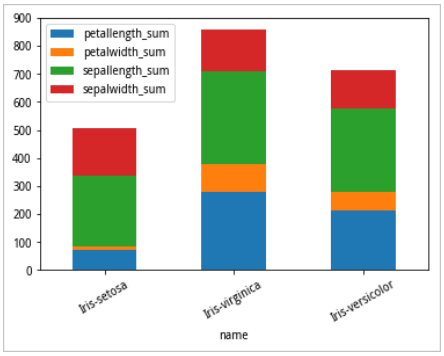
- 多线图

```
>>> iris.plot()
<matplotlib.axes._subplots.AxesSubplot at 0x10db7e690>
```



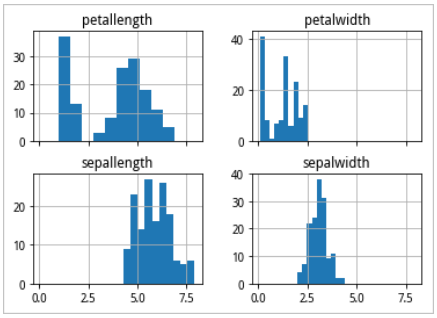
- 竖向柱状图

```
>>> iris.groupby('name').sum().plot(kind='bar', x='name', stacked=True, rot=30)
<matplotlib.axes._subplots.AxesSubplot at 0x10c5f2090>
```



- 直方图

```
>>> iris.hist(sharex=True)
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x10e013f90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10e2d1c10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x10e353f10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10e3c4410>]], dtype=object)
```



参数 `kind` 表示绘图的类型，支持的绘图类型如下表所示。详细的参数说明请参见pandas.DataFrame.plot。

kind	说明
line	线图。
bar	竖向柱状图。
barh	横向柱状图。
hist	直方图。
box	Box图。
kde	核密度估计。
density	和Kde相同。
area	Area图。
pie	饼图。
scatter	散点图。
hexbin	Hexbin图。

除上表所示参数外， `plot` 函数还增加了以下几种参数，方便进行绘图。

参数	说明
xlabel	X轴名。
ylabel	Y轴名。
xlabelsize	X轴名大小。

参数	说明
ylabelsize	Y轴名大小。
labelsize	轴名大小。
title	标题。
titlesize	标题大小。
annotate	是否标记值。

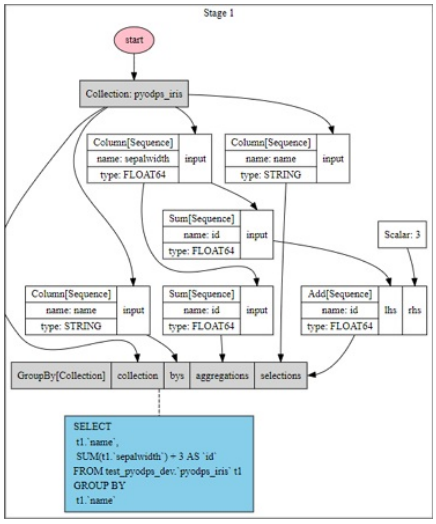
5.6.15. 调试指南

由于PyODPS DataFrame本身会对整个操作执行优化，为了更直观地反应整个过程， 您可以使用可视化的方式显示整个表达式的计算过程。

可视化DataFrame

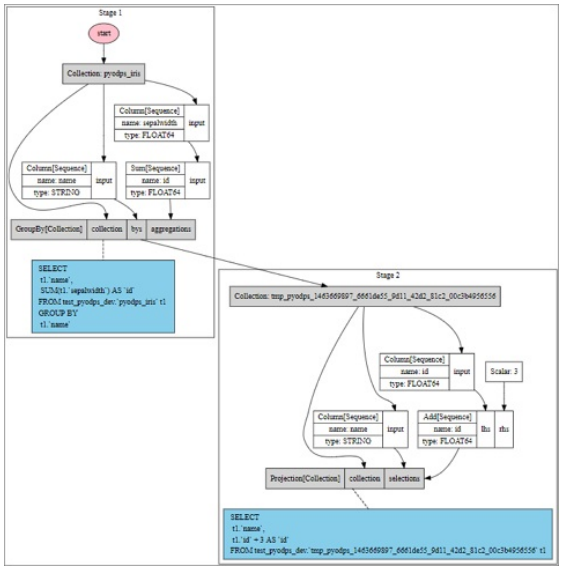
可视化需要依赖graphviz软件和*graphviz*Python包。

```
>>> df = iris.groupby('name').agg(id=iris.sepalwidth.sum())
>>> df = df[df.name, df.id + 3]
>>> df.visualize()
```



由上图可见，这个计算过程中，PyODPS DataFrame将 `groupby` 和列筛选做了操作合并。

```
>>> df = iris.groupby('name').agg(id=iris.sepalwidth.sum()).cache()
>>> df2 = df[df.name, df.id + 3]
>>> df2.visualize()
```



此时，由于执行了Cache操作，整个执行计划将会分成两步执行。

ODPS SQL后端查看编译结果

您可以直接调用 `compile` 方法查看ODPS SQL后端编译到SQL的结果。

```
>>> df = iris.groupby('name').agg(sepalwidth=iris.sepalwidth.max())
>>> df.compile()
Stage 1:
SQL compiled:
SELECT
  t1.`name`,
  MAX(t1.`sepalwidth`) AS `sepalwidth`
FROM test_pyodps_dev.`pyodps_iris` t1
GROUP BY
  t1.`name`
```

使用Pandas计算后端执行本地调试

对于来自ODPS表的DataFrame，部分操作不会编译到ODPS SQL执行，而会使用Tunnel下载表数据。这个下载的过程很快，且无需等待ODPS SQL任务的调度。利用这个特性，您可以快速下载小部分ODPS数据到本地，使用Pandas计算后端进行代码编写和调试。这些操作包括：

- 非分区表：对其进行选取全部或有限条数据、列筛选的操作（不包括列的各种计算），以及计算其数量。
- 不选取分区或筛选前几个分区字段的分区表：对其选取全部或有限条数据、列筛选的操作，以及计算其数量。

例如 `iris` 这个DataFrame来自非分区表，以下操作会使用Tunnel进行下载。

```
>>> iris.count()
>>> iris['name', 'sepalwidth'][:10]
```

例如有个DataFrame来自分区表（有三个分区字段，即 `ds` 、 `hh` 、 `mm` ），以下操作会使用Tunnel下载。

```
>>> df[:10]
>>> df[df.ds == '20160808']['f0', 'f1']
>>> df[(df.ds == '20160808') & (df.hh == 3)][:10]
>>> df[(df.ds == '20160808') & (df.hh == 3) & (df.mm == 15)]
```

因此您可以使用 `to_pandas` 方法将部分数据下载到本地进行调试。

```
>>> DEBUG = True
>>> if DEBUG:
>>> df = iris[:100].to_pandas(wrap=True)
>>> else:
>>> df = iris
```

最后把 `DEBUG` 设置为False，便可以在ODPS上执行完整的计算。

> **说明** 由于沙箱的限制，本地调试通过的程序不一定能在ODPS上也跑通。

5.7. 交互体验增强

5.7.1. 命令行增强

PyODPS提供了命令行下的增强工具。

您配置好账号后，下次登录时无需再次输入账号信息。 配置和调用的步骤如下。

1. 导入PyODPS的增强工具。

```
>>> from odps.inter import setup, enter, teardown
```

2. 配置账号。

```
>>> setup('**your-access_id**', '**your-access-key**', '**your-project**', endpoint='**your-endpoint**')
```

> **说明** 在不指定 `room` 这个参数时，会被配置到 `default` 的 `room` 里。

3. 之后，在任何命令行打开的地方，都可以直接调用。

```
>>> room = enter()
>>> o = room.odps
>>> o.get_table('table_name')
odps.Table
  name: odps_test_sqltask_finance.`table_name`
  schema:
    c_int_a                : bigint
    c_int_b                : bigint
    c_double_a             : double
    c_double_b             : double
    c_string_a             : string
    c_string_b             : string
    c_bool_a               : boolean
    c_bool_b               : boolean
    c_datetime_a           : datetime
    c_datetime_b           : datetime
```

> **说明** 在重新setup `room` 后，ODPS入口对象并不会自动替换，需要再次调用 `enter()` 获得新的 `room` 对象。

在配置好账号及调用出对象后，您可以对 `room` 里的对象进行存储、取出和删除等操作，此外您也可以删除整个 `room` 。

- 常用的ODPS表或者资源都可以存放在 `room` 里。

```
>>> room.store('存储表示例', o.get_table('table_name'), desc='简单的表存储示例')
```

调用 `display` 方法，可将已经存储的对象以表格的形式打印出来。

```
>>> room.display()
default name         desc
存储表示例      简单的存储表示例
iris            安德森鸢尾花卉数据集
```

- 通过 `room['存储表示例']` 或 `room.iris` 命令，可以取出存储的对象。

```
>>> room['存储表示例']
odps.Table
  name: odps_test_sqltask_finance.`table_name`
  schema:
    c_int_a                : bigint
    c_int_b                : bigint
    c_double_a             : double
    c_double_b             : double
    c_string_a             : string
    c_string_b             : string
    c_bool_a               : boolean
    c_bool_b               : boolean
    c_datetime_a           : datetime
    c_datetime_b           : datetime
```

- 调用 `drop` 方法可以对 `room` 内的对象执行删除操作。

```
>>> room.drop('存储表示例')
>>> room.display()
default name          desc
iris          安德森鸢尾花卉数据集
```

- 调用 `teardown()` 方法可以删除某个 `room` 。如果该方法没有传入的参数，则会删除默认的 `room` 。

```
teardown()
```

5.7.2. IPython增强

PyODPS提供了IPython的插件，来更方便得操作ODPS。

IPython增强

针对命令行增强，PyODPS也有相应的命令。

- 执行以下代码加载插件。

```
%load_ext odps
%enter
```

执行结果如下。

```
<odps.inter.Room at 0x11341df10>
```

- 此时全局会包含 `o` 和 `odps` 变量，即ODPS入口。您可以通过 `o.get_table` 或 `odps.get_table` 命令调用表。

```
o.get_table('table_name')
odps.get_table('table_name')
```

执行结果如下。

```
odps.Table
  name: odps_test_sqltask_finance.`table_name`
  schema:
    c_int_a                 : bigint
    c_int_b                 : bigint
    c_double_a              : double
    c_double_b              : double
    c_string_a              : string
    c_string_b              : string
    c_bool_a                : boolean
    c_bool_b                : boolean
    c_datetime_a            : datetime
    c_datetime_b            : datetime
```

- 调用以下命令将已储存的对象以表格形式打印出来。

```
%stores
```

default name	desc
iris	安德森鸢尾花卉数据集

对象名补全

PyODPS拓展了IPython原有的代码补全功能，支持在书写类似 `o.get_xxx` 的语句时，自动补全对象名。以下示例中，<tab>不是实际输入的字符，而是当所有输入完成后，将光标移动到相应位置，并按Tab键。

- 直接使用Tab键补全对象。

```
o.get_table(<tab>
```

- 如果已知需要补全对象的前缀，也可以在使用此功能时输入前缀。

```
o.get_table('tabl<tab>
```

输入上述语句并按Tab键后IPython会自动补全前缀为 `tabl` 的表。

- 对象名补全也支持补全不同Project下的对象名，支持的用法如下。

```
o.get_table(project='project_name', name='tabl<tab>
o.get_table('tabl<tab>', project='project_name')
```

- 如果匹配的对象有多个，IPython会给出一个列表。其最大长度由 `options.completion_size` 给出，默认值为10。

SQL命令

PyODPS还提供了SQL插件，用于执行ODPS SQL。

- 您可以使用单个百分号输入是单行SQL命令。

```
In [37]: %sql select * from pyodps_iris limit 5
|==========================================|   1 /  1  (100.00%)         3s
Out[37]:
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
```

- 多行SQL可以使用 `%%sql` 命令。

```
In [38]: %%sql
   ....: select * from pyodps_iris
   ....: where sepallength < 5
   ....: limit 5
   ....:
|==========================================|   1 /  1  (100.00%)        15s
Out[38]:
   sepallength  sepalwidth  petallength  petalwidth         name
0          4.9         3.0          1.4         0.2  Iris-setosa
1          4.7         3.2          1.3         0.2  Iris-setosa
2          4.6         3.1          1.5         0.2  Iris-setosa
3          4.6         3.4          1.4         0.3  Iris-setosa
4          4.4         2.9          1.4         0.2  Iris-setosa
```

- 如果您需要执行参数化SQL查询，则需以 `:参数` 的方式替换需要的参数。

```
In [1]: %load_ext odps
In [2]: mytable = 'table_name'
In [3]: %sql select * from :mytable
|==========================================|   1 /  1  (100.00%)         2s
Out[3]:
   c_int_a  c_int_b  c_double_a  c_double_b  c_string_a  c_string_b c_bool_a  \
0        0        0       -1203           0           0       -1203     True
  c_bool_b         c_datetime_a         c_datetime_b
0    False  2012-03-30 23:59:58  2012-03-30 23:59:59
```

- 设置SQL运行参数时，可以通过 `%set` 设置到全局，或者在SQL的cell里用SET进行局部设置。以下语句会进行局部设置，不会影响全局的配置。

```
In [17]: %%sql
         set odps.sql.mapper.split.size = 16;
         select * from pyodps_iris;
```

以下设置会进行全局设置，后续运行的SQL都会使用这个设置。

```
In [18]: %set odps.sql.mapper.split.size = 16
```

持久化Pandas DataFrame到ODPS表

PyODPS还提供把Pandas DataFrame上传到ODPS表的命令。您只需使用 `%persist` 命令即可，此命令的第1个参数 `df` 是前面的变量名，第2个参数 `pyodps_pandas_df` 是ODPS表名。

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.arange(9).reshape(3, 3), columns=list('abc'))
%persist df pyodps_pandas_df
```

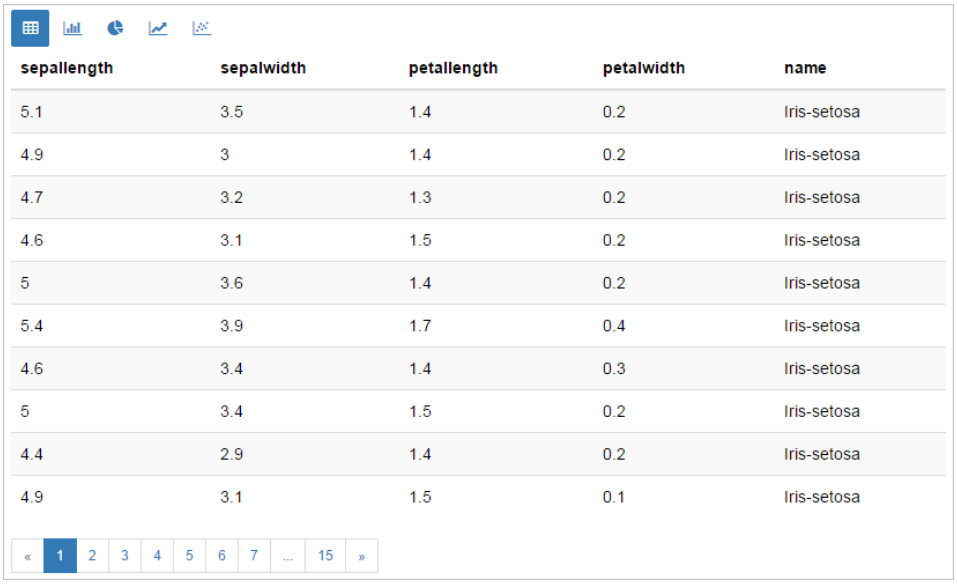
5.7.3. Jupyter Notebook增强

PyODPS针对Jupyter Notebook下的探索性数据分析功能进行了增强， 包括结果探索功能以及进度展示功能。

结果探索

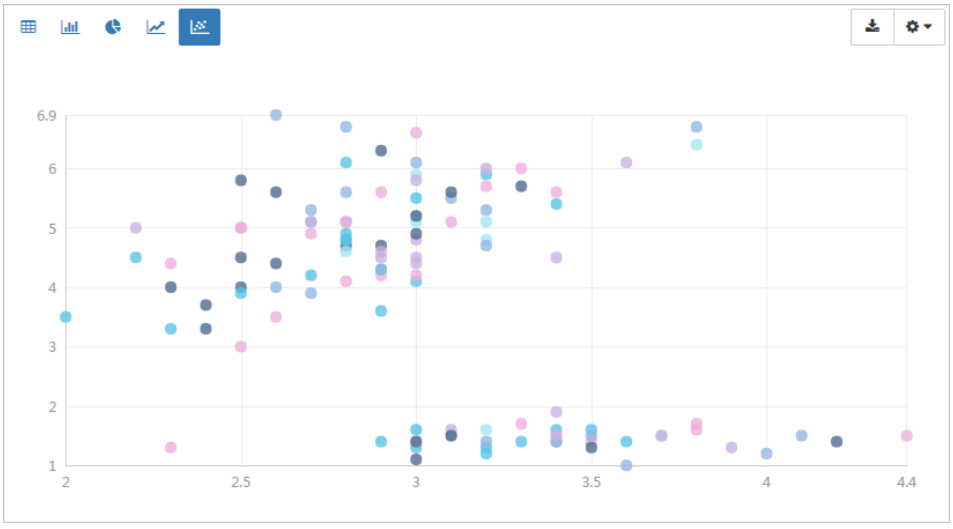
PyODPS在Jupyter Notebook中为SQL Cell和DataFrame提供了数据探索功能。对于已拉到本地的数据，可使用交互式的数据探索工具浏览数据，交互式地绘制图形。

1. 当执行结果为DataFrame时，PyODPS会读取执行结果，并以分页表格的形式展示出来。您可以单击页号或前进、后退按钮，在数据中导航。

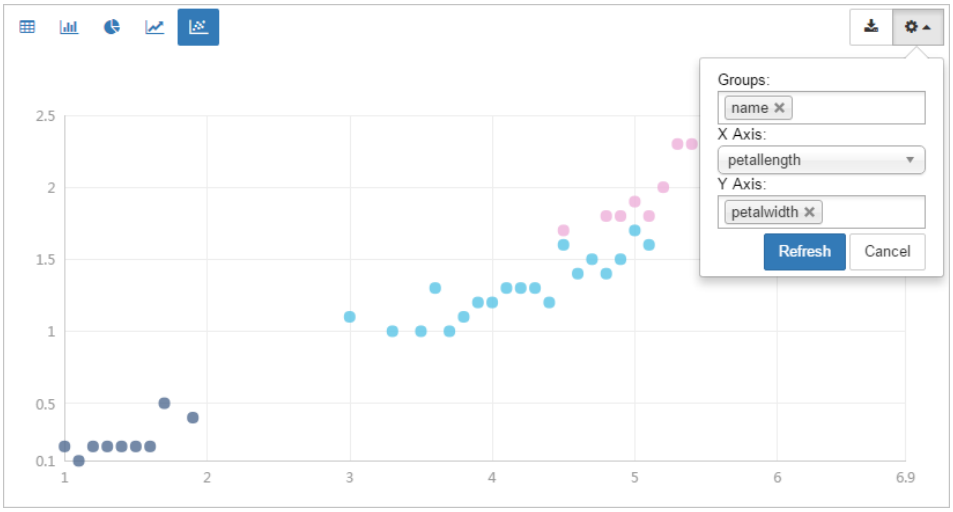


sepallength	sepalwidth	petallength	petalwidth	name
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa

2. 结果区的顶端为模式选择区。除数据表外，也可以选择柱状图、饼图、折线图和散点图。使用默认的字段选择（即前三个字段）时， 绘制的散点图如下。

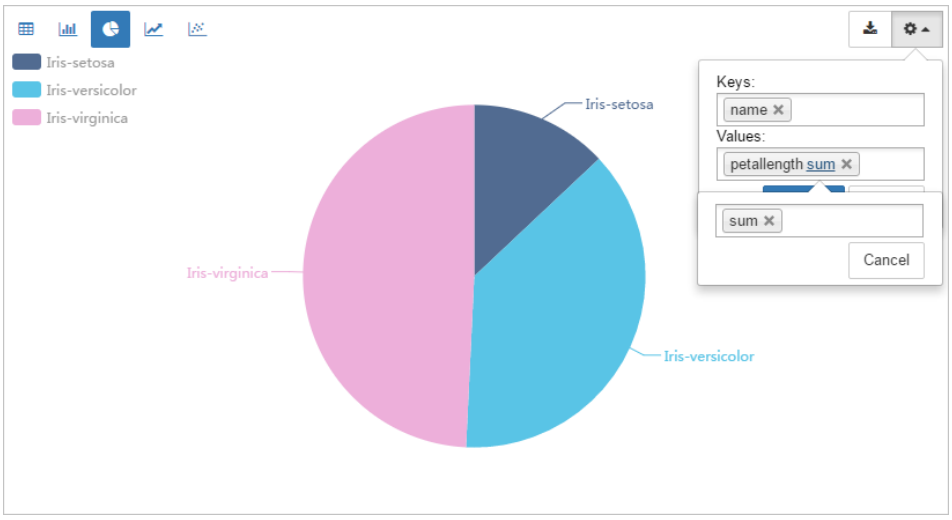


3. 在绘图模式下，单击右上角的配置按钮可以修改图表设置。将name设置为分组列，X轴选择为petallength，Y轴选择为petalwidth。由下图可见，在petallength-petalwidth维度下，数据对name有较好的区分度。



对于柱状图和饼图，值字段支持选择聚合函数。PyODPS对柱状图的默认聚合函数为 `sum` ，对饼图则为 `count` 。如需修改聚合函数， 可以在值字段名称后的聚合函数名上单击，选择所需的聚合函数。

对于折线图，需要避免X轴包含空值，否则可能导致图像不符合预期。

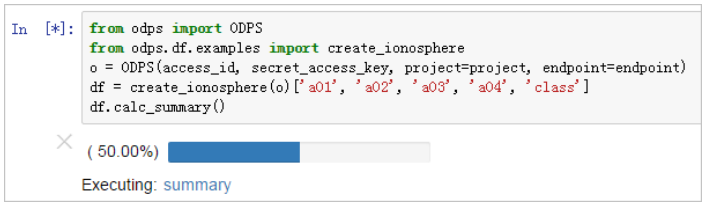


4. 完成绘图后，可单击**下载**保存绘制的图表。

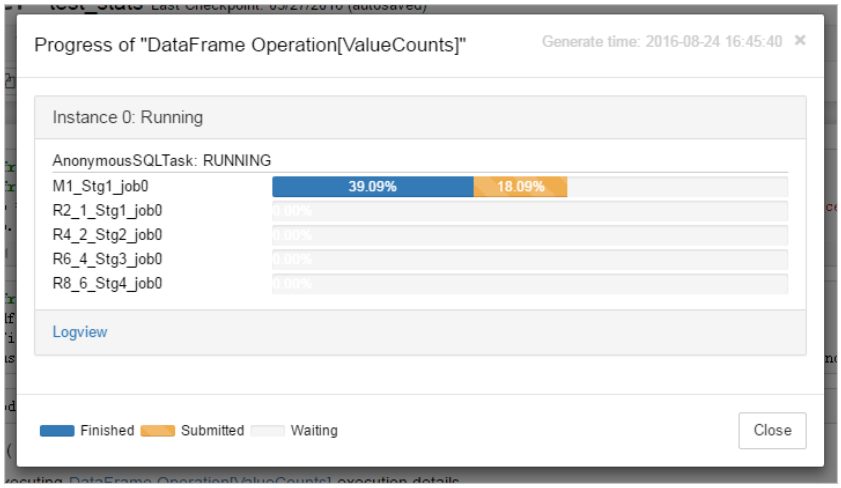
> **说明** 使用此功能前，您需要安装Pandas和ipywidgets。

进度展示

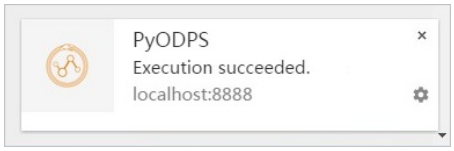
大型作业通常需要执行较长的时间，因此PyODPS提供了进度展示功能。当在Jupyter Notebook中执行DataFrame、机器学习作业或通过执行 `%sql` 编写的SQL语句时，Jupyter Notebook会显示当前正在执行的作业列表及总体进度。



当点击某个作业名称上的链接时，会弹出一个对话框，显示该作业中每个Task的具体执行进度。



当作业运行成功后，将弹出作业是否成功的提示信息。



5.8. API概述

本文为您提供了PyODPS API文档链接，其中包含各函数的参数说明及示例。

- ODPS详解（Definitions）
- PyODPS DataFrame指南（DataFrame Reference）

5.9. 示例程序

5.9.1. PyODPS使用第三方包

本文为您介绍如何在PyODPS中使用第三方包。

前提条件

- 已开通MaxCompute产品。如何开通请参见开通MaxCompute。
- 已开通DataWorks产品。如何开通请参见开通DataWorks。

操作步骤

1. 下载下表中资源包。

资源包名	下载路径	资源名称
python-dateutil	python-dateutil-2.6.0.zip	python-dateutil.zip
pytz	pytz-2017.2.zip	pytz.zip
six	six-1.11.0.tar.gz	six.tar.gz
pandas	pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.whl	pandas.zip
scipy	scipy-0.19.0-cp27-cp27m-manylinux1_x86_64.whl	scipy.zip
scikit-learn	scikit_learn-0.18.1-cp27-cp27m-manylinux1_x86_64.whl	sklearn.zip

> **说明**
> - 您需要手动将下载后的pandas、scipy等资源包后缀.whl修改为.zip，并进行重命名与上述表格**资源名称**保持一致。
> - 上述资源后续需要上传为Archive资源。

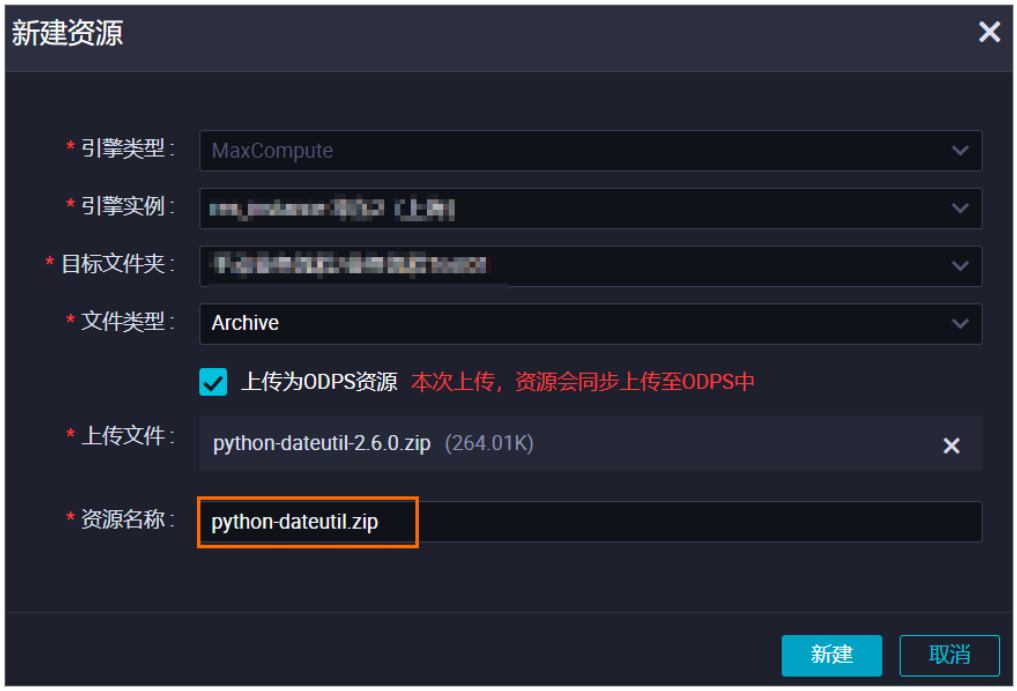
2. 登录DataWorks控制台。
3. 新建业务流程。
   i. 在左侧导航栏选择**工作空间列表**。
   ii. 在**工作空间列表**页面选择对应的项目，单击**操作**列**进入数据开发**。
   iii. 在**DataStudio**页面，右键单击**业务流程**，选择**新建业务流程**。
   iv. 在**新建业务流程**对话框，输入**业务名称**，单击**新建**。
4. 在DataWorks中创建和提交资源。
   i. 在**DataStudio**页面，鼠标悬停至图标，选择**MaxCompute > 资源 > Archive**。

      您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > 资源 > Archive**。

ii. 在**新建资源**对话框，单击**点击上传**，在本地选择*python-dateutil-2.6.0.zip*文件。



iii. 将**资源名称**重命名为*python-dateutil.zip*，单击**新建**。



> **说明** 资源名称命名规则请参见步骤一下载资源包。

iv. 单击图标，在**提交新版本**对话框填写**变更描述**后单击**确认**。



v. 按照上述步骤完成资源*pytz.zip*、*six.tar.gz*、*pandas.zip*、*sklearn.zip*和*scipy.zip*的创建与提交。

5. 新建PyODPS节点。
   i. 在**DataStudio**页面，右键单击新建的业务流程，选择**MaxCompute > 数据开发 > 新建 > PyODPS 2**。
   ii. 在**新建节点**对话框，输入**节点名称**，单击**提交**。
   iii. 在新建的PyODPS节点中，输入代码后单击图标。

   代码示例如下。

```
def test(x):
    from sklearn import datasets, svm
    from scipy import misc
    import numpy as np
    iris = datasets.load_iris()
    assert iris.data.shape == (150, 4)
    assert np.array_equal(np.unique(iris.target),  [0, 1, 2])
    clf = svm.LinearSVC()
    clf.fit(iris.data, iris.target)
    pred = clf.predict([[5.0, 3.6, 1.3, 0.25]])
    assert pred[0] == 0
    assert misc.face().shape is not None
    return x
from odps import options
hints = {
    'odps.isolation.session.enable': True
}
libraries = ['python-dateutil.zip', 'pytz.zip', 'six.tar.gz', 'pandas.zip', 'scipy.zip', 'sklearn.zip']
iris = o.get_table('pyodps_iris').to_df()
print iris[:1].sepallength.map(test).execute(hints=hints, libraries=libraries)
```

6. 在**运行日志**中查看运行结果。

   运行结果如下。

```
Sql compiled:
CREATE TABLE tmp_pyodps_a3172c30_a0d7_4c88_bc39_434168263897 LIFECYCLE 1 AS
SELECT pyodps_udf_1576485276_94d9d978_af66_4e27_a874_e787022dfb3d(t1.`sepallength`) AS `sepallength`
FROM WB_BestPractice_dev.`pyodps_iris` t1
LIMIT 1
Instance ID: 20191216083438175gcv6n4pr2
  Log view: http://logview.odps.aliyun.com/logview/?h=xxxxxx
   sepallength
0          5.1
```

> **说明** 最佳实践案例请参见PyODPS节点实现结巴中文分词。

5.9.2. PyODPS条件查询

本文为您介绍如何使用PyODPS进行条件查询。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。

- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

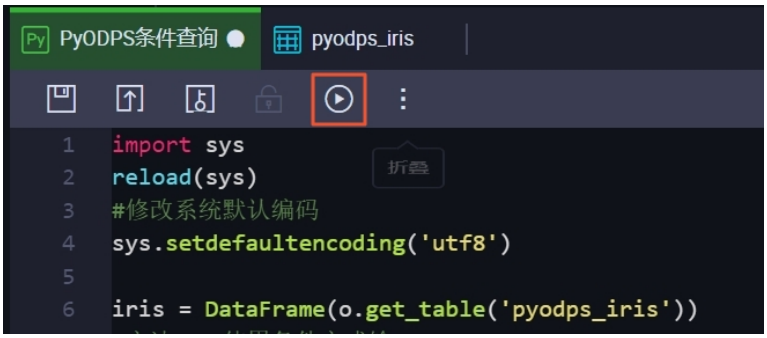
1. 创建表并导入数据。
   i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。
   ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。
   建表语句如下。

```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 单击工作空间后的**进入数据开发**。
5. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
6. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
7. 在PyODPS节点输入示例代码。

```
import sys
reload(sys)
#修改系统默认编码。
sys.setdefaultencoding('utf8')
iris = DataFrame(o.get_table('pyodps_iris'))
#方法一：使用条件方式输出。
with o.execute_sql('select * from pyodps_iris WHERE sepallength > 5 ').open_reader() as reader4:
    print reader4.raw
    for record in reader4:
        print record["sepallength"],record["sepalwidth"],record["petallength"],record["petalwidth"],record["name"]
#方法二：使用ODPS的DataFrame的过滤条件得出数据结果。
print iris[iris.sepallength > 5].head(5)
#方法三：使用query的方式输出。
print iris.query("(sepallength < 5) and (petallength > 1.5)").head(5)
```

8. 单击。



9. 在**运行日志**中查看结果。

```
Executing user script with PyODPS 0.8.0
"sepallength","sepalwidth","petallength","petalwidth","name"
5.1,3.5,1.4,0.2,"Iris-setosa"
5.4,3.9,1.7,0.4,"Iris-setosa"
5.4,3.7,1.5,0.2,"Iris-setosa"
5.8,4.0,1.2,0.2,"Iris-setosa"
5.7,4.4,1.5,0.4,"Iris-setosa"
5.4,3.9,1.3,0.4,"Iris-setosa"
5.1,3.5,1.4,0.3,"Iris-setosa"
5.7,3.8,1.7,0.3,"Iris-setosa"
5.1,3.8,1.5,0.3,"Iris-setosa"
5.4,3.4,1.7,0.2,"Iris-setosa"
5.1,3.7,1.5,0.4,"Iris-setosa"
5.1,3.3,1.7,0.5,"Iris-setosa"
5.2,3.5,1.5,0.2,"Iris-setosa"
```

```
5.2,3.4,1.4,0.2,"Iris-setosa"
5.4,3.4,1.5,0.4,"Iris-setosa"
5.2,4.1,1.5,0.1,"Iris-setosa"
5.5,4.2,1.4,0.2,"Iris-setosa"
5.5,3.5,1.3,0.2,"Iris-setosa"
5.1,3.4,1.5,0.2,"Iris-setosa"
5.1,3.8,1.9,0.4,"Iris-setosa"
5.1,3.8,1.6,0.2,"Iris-setosa"
5.3,3.7,1.5,0.2,"Iris-setosa"
7.0,3.2,4.7,1.4,"Iris-versicolor"
6.4,3.2,4.5,1.5,"Iris-versicolor"
6.9,3.1,4.9,1.5,"Iris-versicolor"
5.5,2.3,4.0,1.3,"Iris-versicolor"
6.5,2.8,4.6,1.5,"Iris-versicolor"
5.7,2.8,4.5,1.3,"Iris-versicolor"
6.3,3.3,4.7,1.6,"Iris-versicolor"
6.6,2.9,4.6,1.3,"Iris-versicolor"
5.2,2.7,3.9,1.4,"Iris-versicolor"
5.9,3.0,4.2,1.5,"Iris-versicolor"
6.0,2.2,4.0,1.0,"Iris-versicolor"
6.1,2.9,4.7,1.4,"Iris-versicolor"
5.6,2.9,3.6,1.3,"Iris-versicolor"
6.7,3.1,4.4,1.4,"Iris-versicolor"
5.6,3.0,4.5,1.5,"Iris-versicolor"
5.8,2.7,4.1,1.0,"Iris-versicolor"
6.2,2.2,4.5,1.5,"Iris-versicolor"
5.6,2.5,3.9,1.1,"Iris-versicolor"
5.9,3.2,4.8,1.8,"Iris-versicolor"
6.1,2.8,4.0,1.3,"Iris-versicolor"
6.3,2.5,4.9,1.5,"Iris-versicolor"
6.1,2.8,4.7,1.2,"Iris-versicolor"
6.4,2.9,4.3,1.3,"Iris-versicolor"
6.6,3.0,4.4,1.4,"Iris-versicolor"
6.8,2.8,4.8,1.4,"Iris-versicolor"
6.7,3.0,5.0,1.7,"Iris-versicolor"
6.0,2.9,4.5,1.5,"Iris-versicolor"
5.7,2.6,3.5,1.0,"Iris-versicolor"
5.5,2.4,3.8,1.1,"Iris-versicolor"
5.5,2.4,3.7,1.0,"Iris-versicolor"
5.8,2.7,3.9,1.2,"Iris-versicolor"
6.0,2.7,5.1,1.6,"Iris-versicolor"
5.4,3.0,4.5,1.5,"Iris-versicolor"
6.0,3.4,4.5,1.6,"Iris-versicolor"
6.7,3.1,4.7,1.5,"Iris-versicolor"
6.3,2.3,4.4,1.3,"Iris-versicolor"
5.6,3.0,4.1,1.3,"Iris-versicolor"
5.5,2.5,4.0,1.3,"Iris-versicolor"
5.5,2.6,4.4,1.2,"Iris-versicolor"
6.1,3.0,4.6,1.4,"Iris-versicolor"
5.8,2.6,4.0,1.2,"Iris-versicolor"
5.6,2.7,4.2,1.3,"Iris-versicolor"
5.7,3.0,4.2,1.2,"Iris-versicolor"
5.7,2.9,4.2,1.3,"Iris-versicolor"
6.2,2.9,4.3,1.3,"Iris-versicolor"
5.1,2.5,3.0,1.1,"Iris-versicolor"
5.7,2.8,4.1,1.3,"Iris-versicolor"
6.3,3.3,6.0,2.5,"Iris-virginica"
5.8,2.7,5.1,1.9,"Iris-virginica"
7.1,3.0,5.9,2.1,"Iris-virginica"
6.3,2.9,5.6,1.8,"Iris-virginica"
6.5,3.0,5.8,2.2,"Iris-virginica"
7.6,3.0,6.6,2.1,"Iris-virginica"
7.3,2.9,6.3,1.8,"Iris-virginica"
6.7,2.5,5.8,1.8,"Iris-virginica"
7.2,3.6,6.1,2.5,"Iris-virginica"
6.5,3.2,5.1,2.0,"Iris-virginica"
6.4,2.7,5.3,1.9,"Iris-virginica"
6.8,3.0,5.5,2.1,"Iris-virginica"
5.7,2.5,5.0,2.0,"Iris-virginica"
5.8,2.8,5.1,2.4,"Iris-virginica"
6.4,3.2,5.3,2.3,"Iris-virginica"
6.5,3.0,5.5,1.8,"Iris-virginica"
7.7,3.8,6.7,2.2,"Iris-virginica"
7.7,2.6,6.9,2.3,"Iris-virginica"
6.0,2.2,5.0,1.5,"Iris-virginica"
```

```
6.9,3.2,5.7,2.3,"Iris-virginica"
5.6,2.8,4.9,2.0,"Iris-virginica"
7.7,2.8,6.7,2.0,"Iris-virginica"
6.3,2.7,4.9,1.8,"Iris-virginica"
6.7,3.3,5.7,2.1,"Iris-virginica"
7.2,3.2,6.0,1.8,"Iris-virginica"
6.2,2.8,4.8,1.8,"Iris-virginica"
6.1,3.0,4.9,1.8,"Iris-virginica"
6.4,2.8,5.6,2.1,"Iris-virginica"
7.2,3.0,5.8,1.6,"Iris-virginica"
7.4,2.8,6.1,1.9,"Iris-virginica"
7.9,3.8,6.4,2.0,"Iris-virginica"
6.4,2.8,5.6,2.2,"Iris-virginica"
6.3,2.8,5.1,1.5,"Iris-virginica"
6.1,2.6,5.6,1.4,"Iris-virginica"
7.7,3.0,6.1,2.3,"Iris-virginica"
6.3,3.4,5.6,2.4,"Iris-virginica"
6.4,3.1,5.5,1.8,"Iris-virginica"
6.0,3.0,4.8,1.8,"Iris-virginica"
6.9,3.1,5.4,2.1,"Iris-virginica"
6.7,3.1,5.6,2.4,"Iris-virginica"
6.9,3.1,5.1,2.3,"Iris-virginica"
5.8,2.7,5.1,1.9,"Iris-virginica"
6.8,3.2,5.9,2.3,"Iris-virginica"
6.7,3.3,5.7,2.5,"Iris-virginica"
6.7,3.0,5.2,2.3,"Iris-virginica"
6.3,2.5,5.0,1.9,"Iris-virginica"
6.5,3.0,5.2,2.0,"Iris-virginica"
6.2,3.4,5.4,2.3,"Iris-virginica"
5.9,3.0,5.1,1.8,"Iris-virginica"
5.1 3.5 1.4 0.2 Iris-setosa
5.4 3.9 1.7 0.4 Iris-setosa
5.4 3.7 1.5 0.2 Iris-setosa
5.8 4.0 1.2 0.2 Iris-setosa
5.7 4.4 1.5 0.4 Iris-setosa
5.4 3.9 1.3 0.4 Iris-setosa
5.1 3.5 1.4 0.3 Iris-setosa
5.7 3.8 1.7 0.3 Iris-setosa
5.1 3.8 1.5 0.3 Iris-setosa
5.4 3.4 1.7 0.2 Iris-setosa
5.1 3.7 1.5 0.4 Iris-setosa
5.1 3.3 1.7 0.5 Iris-setosa
5.2 3.5 1.5 0.2 Iris-setosa
5.2 3.4 1.4 0.2 Iris-setosa
5.4 3.4 1.5 0.4 Iris-setosa
5.2 4.1 1.5 0.1 Iris-setosa
5.5 4.2 1.4 0.2 Iris-setosa
5.5 3.5 1.3 0.2 Iris-setosa
5.1 3.4 1.5 0.2 Iris-setosa
5.1 3.8 1.9 0.4 Iris-setosa
5.1 3.8 1.6 0.2 Iris-setosa
5.3 3.7 1.5 0.2 Iris-setosa
7.0 3.2 4.7 1.4 Iris-versicolor
6.4 3.2 4.5 1.5 Iris-versicolor
6.9 3.1 4.9 1.5 Iris-versicolor
5.5 2.3 4.0 1.3 Iris-versicolor
6.5 2.8 4.6 1.5 Iris-versicolor
5.7 2.8 4.5 1.3 Iris-versicolor
6.3 3.3 4.7 1.6 Iris-versicolor
6.6 2.9 4.6 1.3 Iris-versicolor
5.2 2.7 3.9 1.4 Iris-versicolor
5.9 3.0 4.2 1.5 Iris-versicolor
6.0 2.2 4.0 1.0 Iris-versicolor
6.1 2.9 4.7 1.4 Iris-versicolor
5.6 2.9 3.6 1.3 Iris-versicolor
6.7 3.1 4.4 1.4 Iris-versicolor
5.6 3.0 4.5 1.5 Iris-versicolor
5.8 2.7 4.1 1.0 Iris-versicolor
6.2 2.2 4.5 1.5 Iris-versicolor
5.6 2.5 3.9 1.1 Iris-versicolor
5.9 3.2 4.8 1.8 Iris-versicolor
6.1 2.8 4.0 1.3 Iris-versicolor
6.3 2.5 4.9 1.5 Iris-versicolor
6.1 2.8 4.7 1.2 Iris-versicolor
6.4 2.9 4.3 1.3 Iris-versicolor
```

```
6.6 3.0 4.4 1.4 Iris-versicolor
6.8 2.8 4.8 1.4 Iris-versicolor
6.7 3.0 5.0 1.7 Iris-versicolor
6.0 2.9 4.5 1.5 Iris-versicolor
5.7 2.6 3.5 1.0 Iris-versicolor
5.5 2.4 3.8 1.1 Iris-versicolor
5.5 2.4 3.7 1.0 Iris-versicolor
5.8 2.7 3.9 1.2 Iris-versicolor
6.0 2.7 5.1 1.6 Iris-versicolor
5.4 3.0 4.5 1.5 Iris-versicolor
6.0 3.4 4.5 1.6 Iris-versicolor
6.7 3.1 4.7 1.5 Iris-versicolor
6.3 2.3 4.4 1.3 Iris-versicolor
5.6 3.0 4.1 1.3 Iris-versicolor
5.5 2.5 4.0 1.3 Iris-versicolor
5.5 2.6 4.4 1.2 Iris-versicolor
6.1 3.0 4.6 1.4 Iris-versicolor
5.8 2.6 4.0 1.2 Iris-versicolor
5.6 2.7 4.2 1.3 Iris-versicolor
5.7 3.0 4.2 1.2 Iris-versicolor
5.7 2.9 4.2 1.3 Iris-versicolor
6.2 2.9 4.3 1.3 Iris-versicolor
5.1 2.5 3.0 1.1 Iris-versicolor
5.7 2.8 4.1 1.3 Iris-versicolor
6.3 3.3 6.0 2.5 Iris-virginica
5.8 2.7 5.1 1.9 Iris-virginica
7.1 3.0 5.9 2.1 Iris-virginica
6.3 2.9 5.6 1.8 Iris-virginica
6.5 3.0 5.8 2.2 Iris-virginica
7.6 3.0 6.6 2.1 Iris-virginica
7.3 2.9 6.3 1.8 Iris-virginica
6.7 2.5 5.8 1.8 Iris-virginica
7.2 3.6 6.1 2.5 Iris-virginica
6.5 3.2 5.1 2.0 Iris-virginica
6.4 2.7 5.3 1.9 Iris-virginica
6.8 3.0 5.5 2.1 Iris-virginica
5.7 2.5 5.0 2.0 Iris-virginica
5.8 2.8 5.1 2.4 Iris-virginica
6.4 3.2 5.3 2.3 Iris-virginica
6.5 3.0 5.5 1.8 Iris-virginica
7.7 3.8 6.7 2.2 Iris-virginica
7.7 2.6 6.9 2.3 Iris-virginica
6.0 2.2 5.0 1.5 Iris-virginica
6.9 3.2 5.7 2.3 Iris-virginica
5.6 2.8 4.9 2.0 Iris-virginica
7.7 2.8 6.7 2.0 Iris-virginica
6.3 2.7 4.9 1.8 Iris-virginica
6.7 3.3 5.7 2.1 Iris-virginica
7.2 3.2 6.0 1.8 Iris-virginica
6.2 2.8 4.8 1.8 Iris-virginica
6.1 3.0 4.9 1.8 Iris-virginica
6.4 2.8 5.6 2.1 Iris-virginica
7.2 3.0 5.8 1.6 Iris-virginica
7.4 2.8 6.1 1.9 Iris-virginica
7.9 3.8 6.4 2.0 Iris-virginica
6.4 2.8 5.6 2.2 Iris-virginica
6.3 2.8 5.1 1.5 Iris-virginica
6.1 2.6 5.6 1.4 Iris-virginica
7.7 3.0 6.1 2.3 Iris-virginica
6.3 3.4 5.6 2.4 Iris-virginica
6.4 3.1 5.5 1.8 Iris-virginica
6.0 3.0 4.8 1.8 Iris-virginica
6.9 3.1 5.4 2.1 Iris-virginica
6.7 3.1 5.6 2.4 Iris-virginica
6.9 3.1 5.1 2.3 Iris-virginica
5.8 2.7 5.1 1.9 Iris-virginica
6.8 3.2 5.9 2.3 Iris-virginica
6.7 3.3 5.7 2.5 Iris-virginica
6.7 3.0 5.2 2.3 Iris-virginica
6.3 2.5 5.0 1.9 Iris-virginica
6.5 3.0 5.2 2.0 Iris-virginica
6.2 3.4 5.4 2.3 Iris-virginica
5.9 3.0 5.1 1.8 Iris-virginica
Sql compiled:
CREATE TABLE tmp_pyodps_xxxx LIFECYCLE 1 AS
```

```
SELECT *
FROM xxx.`pyodps_iris` t1
WHERE t1.`sepallength` > 5
Instance ID: 2019xxxx
  Log view: http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/api&p=xxxx&i=2019xxxx&token=xxx
x
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          5.4         3.9          1.7         0.4  Iris-setosa
2          5.4         3.7          1.5         0.2  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.7         4.4          1.5         0.4  Iris-setosa
Sql compiled:
CREATE TABLE tmp_pyodps_xxxx LIFECYCLE 1 AS
SELECT *
FROM xxxx.`pyodps_iris` t1
WHERE (t1.`sepallength` < 5) AND (t1.`petallength` > 1.5)
Instance ID: 2019xxxx
  Log view: http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/api&p=xxxx&i=2019xxxx&token=xxx
x
   sepallength  sepalwidth  petallength  petalwidth            name
0          4.8         3.4          1.6         0.2     Iris-setosa
1          4.8         3.4          1.9         0.2     Iris-setosa
2          4.7         3.2          1.6         0.2     Iris-setosa
3          4.8         3.1          1.6         0.2     Iris-setosa
4          4.9         2.4          3.3         1.0  Iris-versicolor
```

5.9.3. PyODPS参数传递

本文为您介绍如何在DataWorks中进行PyODPS参数的传递。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 准备测试数据。
   i. 创建表并上传数据。操作方法请参见建表并上传数据。

      表结构以及源数据信息如下。

      - 分区表*user_detail*建表语句如下。

        ```
        create table if not exists user_detail
        (
        userid    BIGINT comment '用户id',
        job       STRING comment '工作类型',
        education STRING comment '教育程度'
        ) comment '用户信息表'
        partitioned by (dt STRING comment '日期',region STRING comment '地区');
        ```

      - 源数据表*user_detail_ods*建表语句如下。

        ```
        create table if not exists user_detail_ods
        (
          userid    BIGINT comment '用户id',
          job       STRING comment '工作类型',
          education STRING comment '教育程度',
          dt STRING comment '日期',
          region STRING comment '地区'
        );
        ```

      - 测试数据保存为*user_detail.txt*文件。将此文件上传至表*user_detail_ods*中。

        ```
        0001,互联网,本科,20190715,beijing
        0002,教育,大专,20190716,beijing
        0003,金融,硕士,20190715,shandong
        0004,互联网,硕士,20190715,beijing
        ```

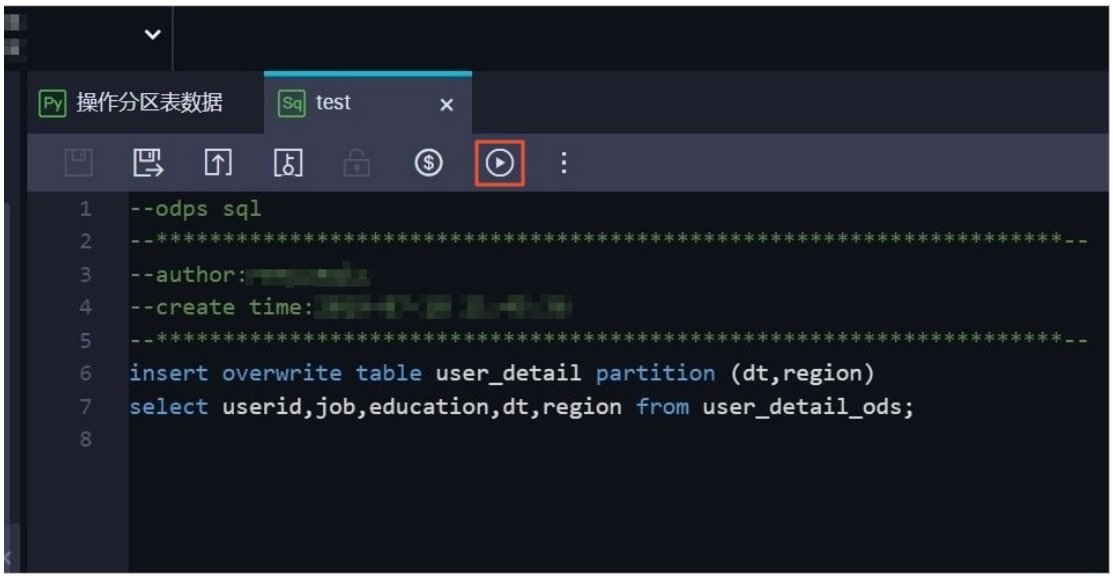
   ii. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。

   iii. 输入**节点名称**，并单击**提交**。

iv. 在ODPS SQL节点中输入如下代码。

```
insert overwrite table user_detail partition (dt,region)
select userid,job,education,dt,region from user_detail_ods;
```

v. 单击**运行**，将数据插入到分区表*user_detail*中。



2. 使用PyODPS传递参数。

i. 登录DataWorks控制台。

ii. 在左侧导航栏上单击**工作空间列表**。

iii. 单击工作空间后的**进入数据开发**。

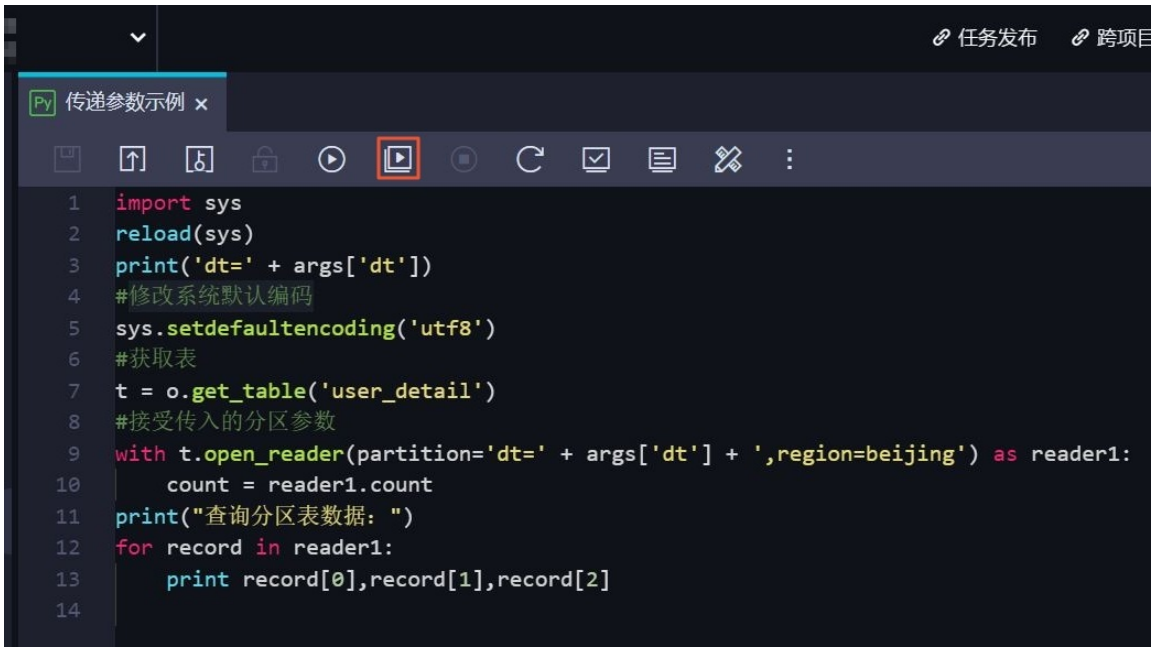
iv. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。

v. 输入**节点名称**，单击**提交**。

vi. 在PyODPS 2节点中输入如下代码实现参数传递。

```
import sys
reload(sys)
print('dt=' + args['dt'])
#修改系统默认编码。
sys.setdefaultencoding('utf8')
#获取表。
t = o.get_table('user_detail')
#接受传入的分区参数。
with t.open_reader(partition='dt=' + args['dt'] + ',region=beijing') as reader1:
    count = reader1.count
print("查询分区表数据：")
for record in reader1:
    print record[0],record[1],record[2]
```

vii. 单击**高级运行（带参数运行）**。

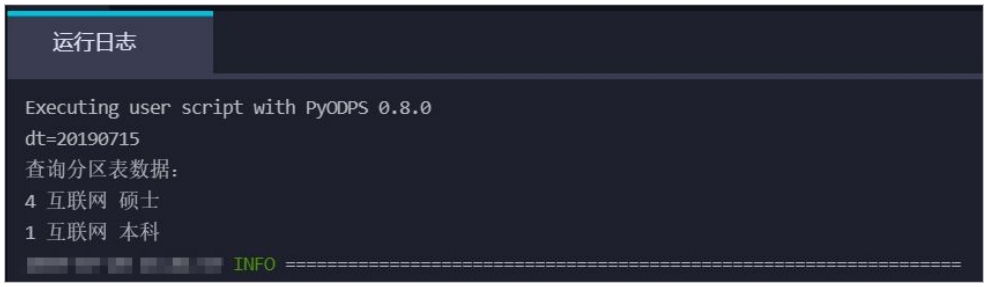


viii. 在**参数**对话框填写配置参数，单击**确定**。

配置参数说明如下：

- **调度资源组**：选择**默认资源组**。
- **dt**：设置为dt=20190715。

ix. 在**运行日志**中查看运行结果。



5.9.4. PyODPS读取分区表数据

本文为您介绍如何通过PyODPS读取分区表数据。

前提条件

您需要完成以下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 准备测试数据。

i. 创建表并上传数据。操作方法请参见建表并上传数据。

表结构以及源数据信息如下。

- 分区表*user_detail*建表语句如下。

```
create table if not exists user_detail
(
userid    BIGINT comment '用户id',
job       STRING comment '工作类型',
education STRING comment '教育程度'
) comment '用户信息表'
partitioned by (dt STRING comment '日期',region STRING comment '地区');
```

- 源数据表*user_detail_ods*建表语句如下。

```
create table if not exists user_detail_ods
(
  userid    BIGINT comment '用户id',
  job       STRING comment '工作类型',
  education STRING comment '教育程度',
  dt STRING comment '日期',
  region STRING comment '地区'
);
```

- 测试数据保存为*user_detail.txt*文件。将此文件上传至表*user_detail_ods*中。

```
0001,互联网,本科,20190715,beijing
0002,教育,大专,20190716,beijing
0003,金融,硕士,20190715,shandong
0004,互联网,硕士,20190715,beijing
```

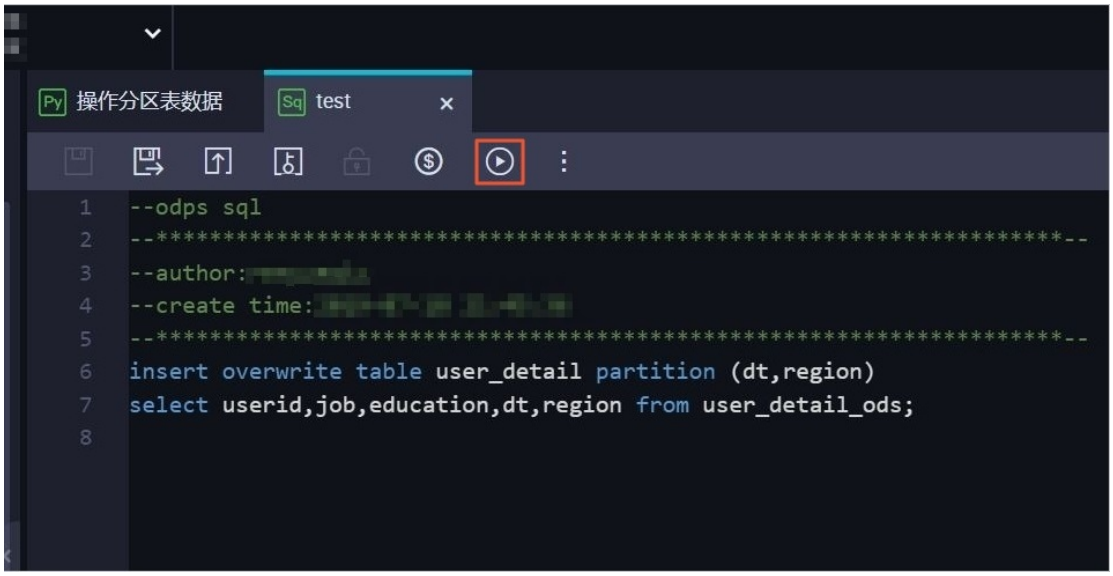
ii. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。

iii. 输入**节点名称**，并单击**提交**。

iv. 在ODPS SQL节点中输入如下代码。

```
insert overwrite table user_detail partition (dt,region)
select userid,job,education,dt,region from user_detail_ods;
```

v. 单击**运行**，将数据插入到分区表*user_detail*中。



2. 使用PyODPS读取分区表数据。

i. 登录DataWorks控制台。

ii. 在左侧导航栏上单击**工作空间列表**。

iii. 单击工作空间后的**进入数据开发**。

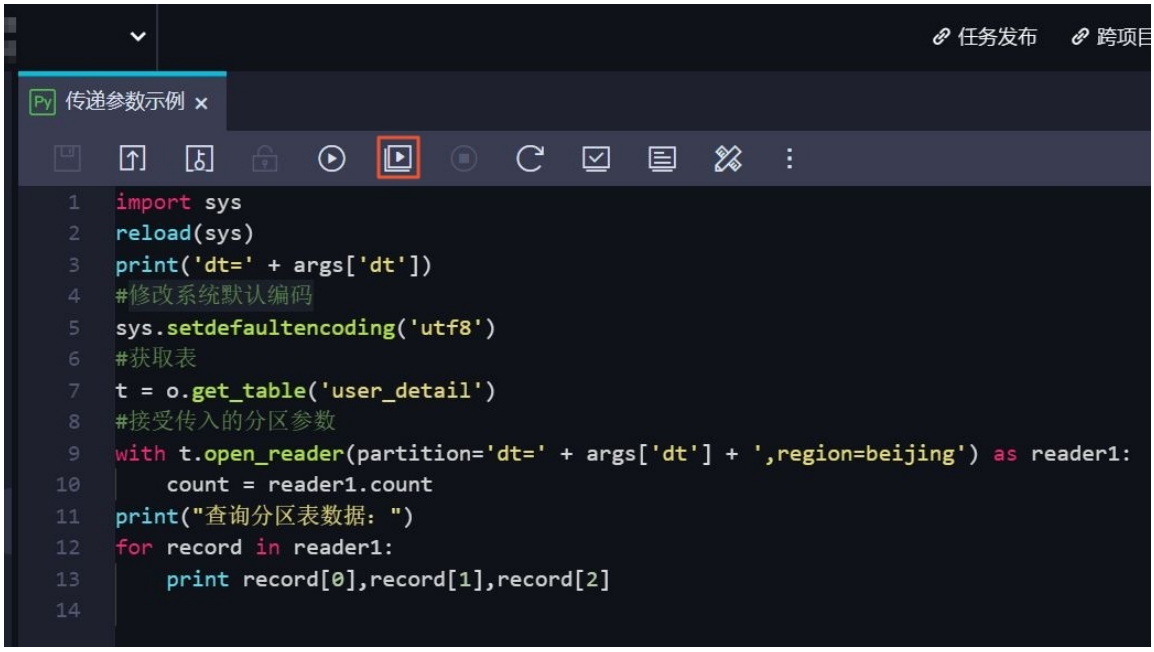
iv. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。

v. 输入**节点名称**，单击**提交**。

vi. 在PyODPS 2节点中输入如下代码读取分区表数据。

```
import sys
from odps import ODPS
reload(sys)
print('dt=' + args['dt'])
#修改系统默认编码。
sys.setdefaultencoding('utf8')
#个人信息凭证。
t = o.get_table('user_detail')
#获取分区表。
print t.exist_partition('dt=20190715,region=beijing')
#查看所有分区。
for partition in t.partitions:
    print partition.name
#您可以通过以下三种方式查询分区表数据。
#第一种方式如下。
with t.open_reader(partition='dt=20190715,region=beijing') as reader1:
    count = reader1.count
print("第一种方式查询分区表数据：")
for record in reader1:
    print record[0],record[1],record[2]
#第二种方式如下。
print("第二种方式查询分区表数据：")
reader2 = t.open_reader(partition='dt=20190715,region=beijing')
for record in reader2:
    print record["userid"],record["job"],record["education"]
#第三种方式如下。
print("第三种方式查询分表数据：")
for record in o.read_table('user_detail', partition='dt=20190715,region=beijing'):
    print record["userid"],record["job"],record["education"]
```

vii. 单击**高级运行（带参数运行）**。



viii. 在**参数**对话框填写配置参数，单击**确定**。

配置参数说明如下：

- **调度资源组**：选择**默认资源组**。
- **dt**：设置为dt=20190715。

ix. 在**运行日志**中查看运行结果。



5.9.5. PyODPS查看一级分区

本文为您介绍如何在PyODPS中查看一级分区。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 准备测试数据。
   i. 创建表并上传数据。操作方法请参见建表并上传数据。

   表结构以及源数据信息如下。

   - 分区表*user_detail*建表语句如下。

   ```
   create table if not exists user_detail
   (
   userid    BIGINT comment '用户id',
   job       STRING comment '工作类型',
   education STRING comment '教育程度'
   ) comment '用户信息表'
   partitioned by (dt STRING comment '日期',region STRING comment '地区');
   ```

   - 源数据表*user_detail_ods*建表语句如下。

   ```
   create table if not exists user_detail_ods
   (
     userid    BIGINT comment '用户id',
     job       STRING comment '工作类型',
     education STRING comment '教育程度',
     dt STRING comment '日期',
     region STRING comment '地区'
   );
   ```

   - 测试数据保存为*user_detail.txt*文件。将此文件上传至表*user_detail_ods*中。

   ```
   0001,互联网,本科,20190715,beijing
   0002,教育,大专,20190716,beijing
   0003,金融,硕士,20190715,shandong
   0004,互联网,硕士,20190715,beijing
   ```

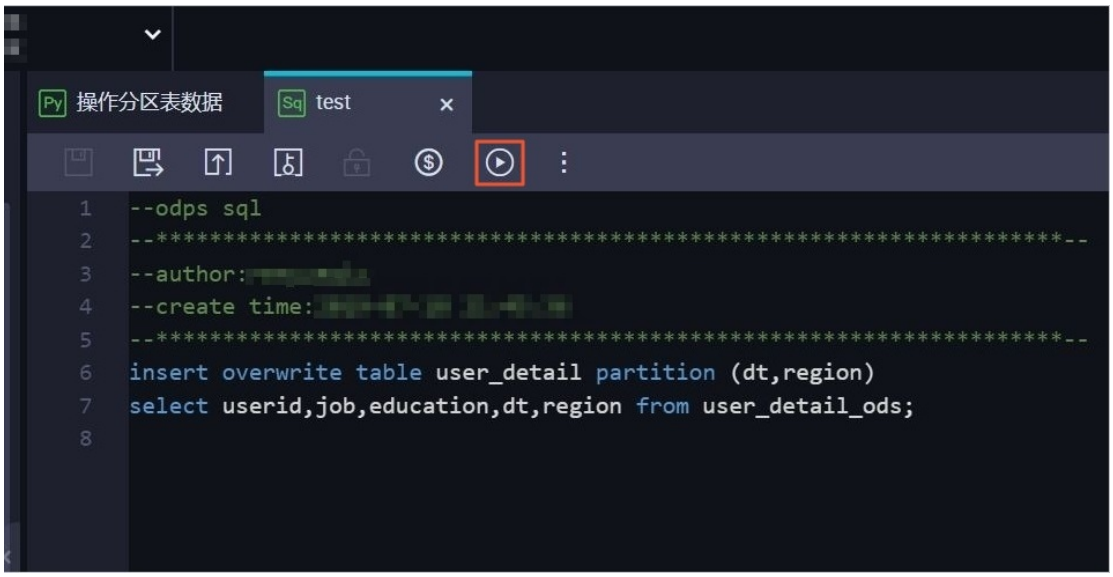
   ii. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。

   iii. 输入**节点名称**，并单击**提交**。

   iv. 在ODPS SQL节点中输入如下代码。

   ```
   insert overwrite table user_detail partition (dt,region)
   select userid,job,education,dt,region from user_detail_ods;
   ```

v. 单击**运行**，将数据插入到分区表*user_detail*中。



2. 使用PyODPS查看一级分区。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏上单击**工作空间列表**。
   iii. 单击工作空间后的**进入数据开发**。
   iv. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
   v. 输入**节点名称**，单击**提交**。
   vi. 在PyODPS 2节点中输入如下代码查看一级分区数据。

```
import sys
reload(sys)
#修改系统默认编码。
sys.setdefaultencoding('utf8')
#异步方式读取一级分区。
instance = o.run_sql('select * from user_detail WHERE dt=\'20190715\'')
instance.wait_for_success()
for record in instance.open_reader():
    print record["userid"],record["job"],record["education"]
#同步方式读取一级分区。
with o.execute_sql('select * from user_detail WHERE dt=\'20190715\'').open_reader() as reader4:
    print reader4.raw
    for record in reader4:
        print record["userid"],record["job"],record["education"]
#使用ODPS的DataFrame获取一级分区。
pt_df = DataFrame(o.get_table('user_detail').get_partition('dt=20190715'))
print pt_df.head(10)
```

vii. 单击**运行**按钮。

查看一级分区

```
import sys
reload(sys)
#修改系统默认编码
sys.setdefaultencoding('utf8')

#异步方式读取一级分区
instance = o.run_sql('select * from user_detail WHERE dt=\'20190715\'')
instance.wait_for_success()
for record in instance.open_reader():
    print record["userid"],record["job"],record["education"]

#同步方式读取一级分区
with o.execute_sql('select * from user_detail WHERE dt=\'20190715\'').open_reader() as reader4:
    print reader4.raw
    for record in reader4:
        print record["userid"],record["job"],record["education"]

#使用ODPS的DataFrame获取一级分区
pt_df = DataFrame(o.get_table('user_detail').get_partition('dt=20190715'))
print pt_df.head(10)
```

viii. 在**运行日志**中查看运行结果。

运行日志

```
4 互联网 硕士
1 互联网 本科
3 金融 硕士

"userid","job","education","dt","region"
4,"互联网","硕士","20190715","beijing"
1,"互联网","本科","20190715","beijing"
3,"金融","硕士","20190715","shandong"
4 互联网 硕士
1 互联网 本科
3 金融 硕士
Try to fetch data from tunnel

   userid  job education
0       4  互联网        硕士
1       1  互联网        本科
2       3   金融        硕士
```

5.9.6. PyODPS的Sequence及执行操作

本文为您介绍如何进行PyODPS的Sequence及执行操作。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 创建表并导入数据。
   i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。

ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。

建表语句如下。

```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 单击工作空间后的**进入数据开发**。
5. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
6. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
7. 进入**PyODPS**节点编辑框，输入示例代码。示例代码如下。

```
from odps import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
#获取列。
print iris.sepallength.head(5)
print iris['sepallength'].head(5)
#查看列的类型。
print iris.sepallength.dtype
#修改列的类型。
iris.sepallength.astype('int')
#计算。
print iris.groupby('name').sepallength.max().head(5)
print iris.sepallength.max()
#重命名列。
print iris.sepalwidth.rename('speal_width').head(5)
#简单的列变化。
print (iris.sepallength + iris.sepalwidth).rename('sum_sepal').head(5)
```

8. 单击**运行**。
9. 在**运行日志**中查看结果。

结果如下。

```
Executing user script with PyODPS 0.8.0
Try to fetch data from tunnel
   sepallength
0          4.9
1          4.7
2          4.6
3          5.0
4          5.4
Try to fetch data from tunnel
   sepallength
0          4.9
1          4.7
2          4.6
3          5.0
4          5.4
FLOAT64
Sql compiled:
CREATE TABLE tmp_pyodps_ed78e3ba_f13c_4a49_812d_2790d57c25dd LIFECYCLE 1 AS
SELECT MAX(t1.`sepallength`) AS `sepallength_max`
FROM data_service_fr.`pyodps_iris` t1
GROUP BY t1.`name`
   sepallength_max
0              5.8
1              7.0
2              7.9
Collection: ref_0
  odps.Table
    name: data_service_fr.`pyodps_iris`
    schema:
      sepallength           : double      # 片长度(cm)
      sepalwidth            : double      # 片宽度(cm)
      petallength           : double      # 瓣长度(cm)
      petalwidth            : double      # 瓣宽度(cm)
      name                  : string      # 种类
max = Max[float64]
  sepallength = Column[sequence(float64)] 'sepallength' from collection ref_0
Try to fetch data from tunnel
   speal_width
0          3.0
1          3.2
2          3.1
3          3.6
4          3.9
Sql compiled:
CREATE TABLE tmp_pyodps_28120275_8d0f_4683_8318_302fa21459ac LIFECYCLE 1 AS
SELECT t1.`sepallength` + t1.`sepalwidth` AS `sum_sepal`
FROM data_service_fr.`pyodps_iris` t1
   sum_sepal
0        7.9
1        7.9
2        7.7
3        8.6
4        9.3
2019-08-13 10:48:13 INFO ================================================================
2019-08-13 10:48:13 INFO Exit code of the Shell command 0
2019-08-13 10:48:13 INFO --- Invocation of Shell command completed ---
2019-08-13 10:48:13 INFO Shell run successfully!
```

10. 按照如上方法，新建并运行Pyodps节点PyExecute。

    PyExecute节点示例代码如下。

```
from odps import options
from odps import DataFrame
#查看运行时的instance的logview。
options.verbose = True
iris = DataFrame(o.get_table('pyodps_iris'))
iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
my_logs = []
def my_loggers(x):
    my_logs.append(x)
options.verbose_log = my_loggers
iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
print(my_logs)
#缓存中间Collection结果。
cached = iris[iris.sepalwidth < 3.5].cache()
print cached.head(3)
#异步和并行执行。
from odps.df import Delay
delay = Delay() #创建Delay对象。
df = iris[iris.sepalwidth < 5].cache()  #有一个共同的依赖。
future1 = df.sepalwidth.sum().execute(delay=delay) #立即返回future对象，此时并没有执行。
future2 = df.sepalwidth.mean().execute(delay=delay)
future3 = df.sepalwidth.max().execute(delay=delay)
delay.execute(n_parallel=3)
print future1.result()
print future2.result()
print future3.result()
```

运行结果如下。

```
Executing user script with PyODPS 0.8.0
Sql compiled:
CREATE TABLE tmp_pyodps_4a204590_0510_4e9c_823b_5b837a437840 LIFECYCLE 1 AS
SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`
FROM data_service_fr.`pyodps_iris` t1
WHERE t1.`sepallength` < 5
LIMIT 5
Instance ID: 20190813025233386g04djssa
  Log view: http://logview.odps.aliyun.com/logview/XXX
['Sql compiled:', 'CREATE TABLE tmp_pyodps_03b92c55_8442_4e61_8978_656495487b8a LIFECYCLE 1 AS \nSELECT t1.`sepalwidth`, t1.`
petallength`, t1.`petalwidth`, t1.`name` \nFROM data_service_fr.`pyodps_iris` t1 \nWHERE t1.`sepallength` < 5 \nLIMIT 5', 'In
stance ID: 20190813025236282gcsna5pr2', u'
Log view: http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&XXX
   sepallength  sepalwidth  petallength  petalwidth         name
0          4.9         3.0          1.4         0.2  Iris-setosa
1          4.7         3.2          1.3         0.2  Iris-setosa
2          4.6         3.1          1.5         0.2  Iris-setosa
454.6
3.05100671141
4.4
2019-08-13 10:52:48 INFO ======================================================================
2019-08-13 10:52:48 INFO Exit code of the Shell command 0
2019-08-13 10:52:48 INFO --- Invocation of Shell command completed ---
2019-08-13 10:52:48 INFO Shell run successfully!
2019-08-13 10:52:48 INFO Current task status: FINISH
```

5.9.7. PyODPS的聚合操作

本文为您介绍如何进行PyODPS的聚合操作。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 创建表并导入数据。
   i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。

ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。

建表语句如下。

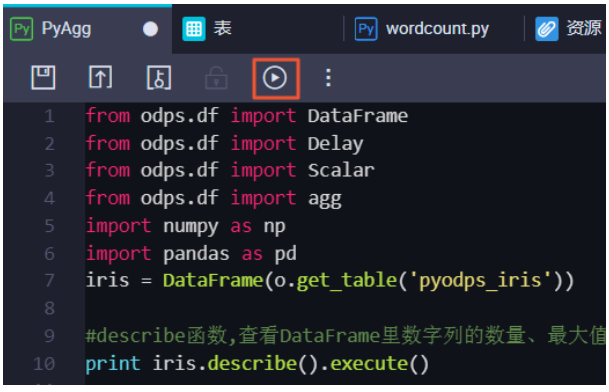
```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
5. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
6. 在PyODPS节点中输入聚合操作代码。

示例代码如下。

```
from odps.df import DataFrame
from odps.df import Delay
from odps.df import Scalar
from odps.df import agg
import numpy as np
import pandas as pd
iris = DataFrame(o.get_table('pyodps_iris'))
#describe函数，查看DataFrame里数字列的数量、最大值、最小值、平均值以及标准差。
print iris.describe().execute()
#单列聚合操作。
print iris.sepallength.max().execute()
#消除重复列的聚合。
print iris.name.unique().cat(sep=',').execute()
#统计总行数。
print iris.count().execute()
#分组聚合。
print iris.groupby('name').agg(iris.sepallength.max(),smin=iris.sepallength.min()).head()
print iris.groupby('name').agg(count=iris.name.count()).sort('count', ascending=False).head(5)
print iris.groupby('name').petallength.sum().head()
print iris.groupby('name').agg(iris.petallength.notnull().sum()).head()
print iris.groupby(Scalar(1)).petallength.sum().head()
#自定义聚合，对应单列。
class Agg(object):
    def buffer(self):
        return [0.0, 0]
    def __call__(self, buffer, val):
        buffer[0] += val
        buffer[1] += 1
    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]
    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
print iris.sepalwidth.agg(Agg).execute()
#自定义聚合，对应双列。
class Agg2(object):
    def buffer(self):
        return [0.0, 0.0]
    def __call__(self, buffer, val1, val2):
        buffer[0] += val1
        buffer[1] += val2
    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]
    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] - buffer[1]
to_agg = agg([iris.sepalwidth, iris.sepallength], Agg2, rtype='float')  # 对两列调用自定义聚合。
print iris.groupby('name').agg(val=to_agg).execute()
#HyperLogLog计数。
df5 = DataFrame(pd.DataFrame({'a': np.random.randint(100000, size=100000)}))
print df5.a.hll_count().execute()
```

7. 单击**运行**。



8. 在**运行日志**中查看结果。

运行结果如下。

```
Executing user script with PyODPS 0.8.0
Sql compiled:
CREATE TABLE tmp_pyodps_28b83337_87f9_4b11_a084_f1896758d53b LIFECYCLE 1 AS
SELECT pyodps_udf_1567394092_9467b7ba_3121_4105_81dc_431e8e68be39(t3.`sepallength_count`, t3.`sepallength_mean`, t3.`sepallen
gth_std`, t3.`sepallength_min`, t3.`sepallength_quantile_25`, t3.`sepallength_quantile_50`, t3.`sepallength_quantile_75`, t3.
`sepallength_max`, t3.`sepalwidth_count`, t3.`sepalwidth_mean`, t3.`sepalwidth_std`, t3.`sepalwidth_min`, t3.`sepalwidth_quan
tile_25`, t3.`sepalwidth_quantile_50`, t3.`sepalwidth_quantile_75`, t3.`sepalwidth_max`, t3.`petallength_count`, t3.`petallen
gth_mean`, t3.`petallength_std`, t3.`petallength_min`, t3.`petallength_quantile_25`, t3.`petallength_quantile_50`, t3.`petall
ength_quantile_75`, t3.`petallength_max`, t3.`petalwidth_count`, t3.`petalwidth_mean`, t3.`petalwidth_std`, t3.`petalwidth_mi
n`, t3.`petalwidth_quantile_25`, t3.`petalwidth_quantile_50`, t3.`petalwidth_quantile_75`, t3.`petalwidth_max`) AS (`type`, `
sepallength`, `sepalwidth`, `petallength`, `petalwidth`)
FROM (
  SELECT COUNT(t2.`sepallength`) AS `sepallength_count`, AVG(t2.`sepallength`) AS `sepallength_mean`, STDDEV_SAMP(t2.`sepalle
ngth`) AS `sepallength_std`, MIN(t2.`sepallength`) AS `sepallength_min`, PERCENTILE(t2.`sepallength`, 0.25) AS `sepallength_q
uantile_25`, PERCENTILE(t2.`sepallength`, 0.5) AS `sepallength_quantile_50`, PERCENTILE(t2.`sepallength`, 0.75) AS `sepalleng
th_quantile_75`, MAX(t2.`sepallength`) AS `sepallength_max`, COUNT(t2.`sepalwidth`) AS `sepalwidth_count`, AVG(t2.`sepalwidth
`) AS `sepalwidth_mean`, STDDEV_SAMP(t2.`sepalwidth`) AS `sepalwidth_std`, MIN(t2.`sepalwidth`) AS `sepalwidth_min`, PERCENTI
LE(t2.`sepalwidth`, 0.25) AS `sepalwidth_quantile_25`, PERCENTILE(t2.`sepalwidth`, 0.5) AS `sepalwidth_quantile_50`, PERCENTI
LE(t2.`sepalwidth`, 0.75) AS `sepalwidth_quantile_75`, MAX(t2.`sepalwidth`) AS `sepalwidth_max`, COUNT(t2.`petallength`) AS `
petallength_count`, AVG(t2.`petallength`) AS`petallength_mean`, STDDEV_SAMP(t2.`petallength`) AS `petallength_std`, MIN(t2.`p
etallength`) AS `petallength_min`, PERCENTILE(t2.`petallength`, 0.25) AS `petallength_quantile_25`, PERCENTILE(t2.`petallengt
h`, 0.5) AS `petallength_quantile_50`, PERCENTILE(t2.`petallength`, 0.75) AS `petallength_quantile_75`, MAX(t2.`petallength`)
AS `petallength_max`, COUNT(t2.`petalwidth`) AS `petalwidth_count`, AVG(t2.`petalwidth`) AS `petalwidth_mean`, STDDEV_SAMP(t2
.`petalwidth`) AS `petalwidth_std`, MIN(t2.`petalwidth`) AS `petalwidth_min`, PERCENTILE(t2.`petalwidth`, 0.25) AS `petalwidt
h_quantile_25`, PERCENTILE(t2.`petalwidth`, 0.5) AS `petalwidth_quantile_50`, PERCENTILE(t2.`petalwidth`, 0.75) AS `petalwidt
h_quantile_75`, MAX(t2.`petalwidth`) AS `petalwidth_max`
  FROM (
    SELECT t1.`sepallength`, t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`
    FROM DQC_0221_dev.`pyodps_iris` t1
  ) t2
) t3
          type  sepallength  sepalwidth  petallength  petalwidth
0        count   149.000000  149.000000   149.000000  149.000000
1         mean     5.848322    3.051007     3.774497    1.205369
2          std     0.828594    0.433499     1.759651    0.761292
3          min     4.300000    2.000000     1.000000    0.100000
4  quantile_25     5.000000    2.000000     1.000000    0.000000
5  quantile_50     5.000000    3.000000     4.000000    1.000000
6  quantile_75     6.000000    3.000000     5.000000    1.000000
7          max     7.900000    4.400000     6.900000    2.500000
Sql compiled:
SELECT MAX(t2.`sepallength`) AS `sepallength_max`
FROM (
  SELECT t1.`sepallength`
  FROM DQC_0221_dev.`pyodps_iris` t1
) t2
7.9
Sql compiled:
SELECT WM_CONCAT(DISTINCT ',', t2.`name`) AS `name_cat`
FROM (
  SELECT t1.`name`
  FROM DQC_0221_dev.`pyodps_iris` t1
) t2
Iris-setosa,Iris-versicolor,Iris-virginica
149
Sql compiled:
CREATE TABLE tmp_pyodps_4b00d671_82b1_4cd8_8af4_f11dbdac4570 LIFECYCLE 1 AS
SELECT t1.`name`, MAX(t1.`sepallength`) AS `sepallength_max`, MIN(t1.`sepallength`) AS `smin`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
             name  sepallength_max  smin
0     Iris-setosa              5.8   4.3
1  Iris-versicolor              7.0   4.9
2   Iris-virginica              7.9   4.9
Sql compiled:
CREATE TABLE tmp_pyodps_276c901f_7cf0_48ee_aaae_0d3092822b34 LIFECYCLE 1 AS
SELECT t1.`name`, COUNT(t1.`name`) AS `count`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
ORDER BY count DESC
LIMIT 10000
             name  count
0  Iris-versicolor     50
```

```
1   Iris-virginica     50
2      Iris-setosa     49
Sql compiled:
CREATE TABLE tmp_pyodps_440d12ec_8496_4b87_b33f_b9624941e5bd LIFECYCLE 1 AS
SELECT SUM(t1.`petallength`) AS `petallength_sum`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
Instance ID: 20190902031604255go6c472m
   petallength_sum
0            71.8
1           213.0
2           277.6
Sql compiled:
CREATE TABLE tmp_pyodps_61e62f5f_7b4f_4971_8c39_dcaccba37764 LIFECYCLE 1 AS
SELECT t1.`name`, SUM(IF(t1.`petallength` IS NOT NULL, 1, 0)) AS `petallength_sum`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
Instance ID: 20190902031607441ghzyr392
              name  petallength_sum
0      Iris-setosa               49
1  Iris-versicolor               50
2   Iris-virginica               50
Sql compiled:
CREATE TABLE tmp_pyodps_daeffd50_88fe_4b4d_9db9_57810cfe71c3 LIFECYCLE 1 AS
SELECT SUM(t1.`petallength`) AS `petallength_sum`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY 1
   petallength_sum
0           562.4
Sql compiled:
SELECT pyodps_udf_1567394173_ef78183a_a4f5_4372_91dd_18d6ecc6f651(t2.`sepalwidth`) AS `sepalwidth_aggregation`
FROM (
  SELECT t1.`sepalwidth`
  FROM DQC_0221_dev.`pyodps_iris` t1
) t2
Sql compiled:
CREATE TABLE tmp_pyodps_e312f110_64e7_465e_adae_fd053f125a90 LIFECYCLE 1 AS
SELECT t1.`name`, pyodps_udf_1567394220_49231d7a_faeb_4f2f_9de5_36c4aefaa29b(t1.`sepalwidth`, t1.`sepallength`) AS `val`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
              name    val
0      Iris-setosa  -77.8
1  Iris-versicolor -158.3
2   Iris-virginica -180.7
63021
```

5.9.8. PyODPS的列运算

本文为您介绍如何进行PyODPS的列运算。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 创建表并导入数据。
   i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。
   ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。
   建表语句如下。

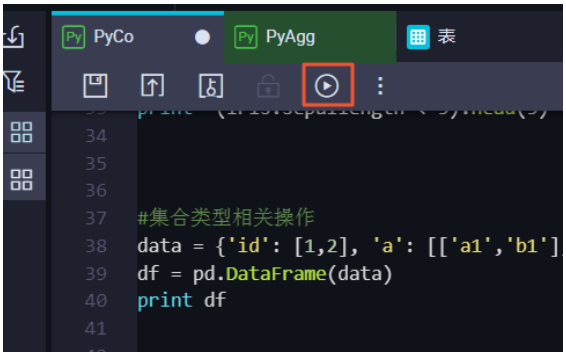
```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 单击工作空间后的**进入数据开发**。
5. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
6. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
7. 在PyODPS节点输入示例代码。

   示例代码如下。

```
from odps import DataFrame
import numpy as np
import pandas as pd
iris = DataFrame(o.get_table('pyodps_iris'))
#判断是否为null。
print iris.sepallength.isnull().head(5)
#逻辑判断。
print (iris.sepallength > 5).ifelse('gt5','lte5').rename('cmp5').head(5)
#多条件判断。
print iris.sepallength.switch(4.9,'eq4.9',5.0,'eq5.0',default='noeq').rename('equalness').head(5)
from odps.df import switch
print switch(iris.sepallength == 4.9,'eq4.9',iris.sepallength == 5.0,'eq5.0',default='noeq').rename('equalness').head(5)
#修该数据集某一列的一部分值。
iris[iris.sepallength > 5,'cmp5'] = 'gt5'
iris[iris.sepallength <=5,'cmp5'] = 'lte5'
print iris.head(5)
#数学计算。
print (iris.sepallength * 10).log().head(5)
fields = [iris.sepallength,(iris.sepallength /2).rename('sepallength/2'),(iris.sepallength ** 2).rename('sepallength的平方')]
print iris[fields].head(5)
print  (iris.sepallength < 5).head(5)
#集合类型相关操作。
data = {'id': [1,2], 'a': [['a1','b1'],['c1']], 'b': [{'a2': 0, 'b2': 1, 'c2': 2},{'d2': 3, 'e2': 4}]}
df = pd.DataFrame(data)
print df
df1 = DataFrame(df, unknown_as_string=True, as_type={'a': 'list<string>','b' : 'dict<string,int64>'})
print df1.dtypes
print df1.head()
print df1[df1.id,df1.a[0],df1.b.len()].head()
print df1.a.explode().head()
print df1.a.explode(pos=True).head()
print df1.b.explode().head()
print df1.b.explode(['key','value']).head()
#explode与其他并列多行输出。
print df1[df1.id,df1.a.explode()].head()
#isin,notin,cut。
#isin用于判断Sequence里的元素是否在某个集合元素里；notin反之。
print iris.sepallength.isin([4.9,5.1]).rename('sepallength').head()
#cut提供离散化的操作，可以将Sequence的数据拆成几个区段。
print iris.sepallength.cut(range(6),labels=['0-1','1-2','2-3','3-4','4-5']).rename('sepallength_cut').head(5)
#include_under和include_over可以分别包括向下和向上的区间。
labels = ['0-1', '1-2', '2-3', '3-4', '4-5', '5-']
iris.sepallength.cut(range(6), labels=labels, include_over=True).rename('sepallength_cut').head(5)
```

8. 单击**运行**。



9. 在**运行日志**中查看结果。

   运行结果如下。

```
Executing user script with PyODPS 0.8.0
```

```
Sql compiled:
CREATE TABLE tmp_pyodps_32e39a9a_8ae6_4815_b465_4b9dfa3bf1b9 LIFECYCLE 1 AS
SELECT t1.`sepallength` IS NULL AS `sepallength`
FROM DQC_0221_dev.`pyodps_iris` t1
  sepallength
0       False
1       False
2       False
3       False
4       False
Sql compiled:
CREATE TABLE tmp_pyodps_d684aded_8ad8_4119_8558_1fddd774dd3f LIFECYCLE 1 AS
SELECT IF(t1.`sepallength` > 5, 'gt5', 'lte5') AS `cmp5`
FROM DQC_0221_dev.`pyodps_iris` t1
   cmp5
0  lte5
1  lte5
2  lte5
3  lte5
4   gt5
Sql compiled:
CREATE TABLE tmp_pyodps_cfc5bdb9_6d3e_4711_8acc_913b29b96c9a LIFECYCLE 1 AS
SELECT CASE t1.`sepallength` WHEN 4.9 THEN 'eq4.9' WHEN 5.0 THEN 'eq5.0' ELSE 'noeq' END AS `equalness`
FROM DQC_0221_dev.`pyodps_iris` t1
  equalness
0     eq4.9
1      noeq
2      noeq
3     eq5.0
4      noeq
Sql compiled:
CREATE TABLE tmp_pyodps_d1fd70b4_2ad8_4dc9_838e_f566f7f5841e LIFECYCLE 1 AS
SELECT CASE WHEN t1.`sepallength` == 4.9 THEN 'eq4.9' WHEN t1.`sepallength` == 5.0 THEN 'eq5.0' ELSE 'noeq' END AS `equalness
`
FROM DQC_0221_dev.`pyodps_iris` t1
  equalness
0     eq4.9
1      noeq
2      noeq
3     eq5.0
4      noeq
Sql compiled:
CREATE TABLE tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553 LIFECYCLE 1 AS
SELECT t1.`sepallength`, t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`, IF(t1.`sepallength` <= 5, 'lte5', IF(
t1.`sepallength` > 5, 'gt5', CAST(NULL AS STRING))) AS `cmp5`
FROM DQC_0221_dev.`pyodps_iris` t1
   sepallength  sepalwidth  petallength  petalwidth         name  cmp5
0          4.9         3.0          1.4         0.2  Iris-setosa  lte5
1          4.7         3.2          1.3         0.2  Iris-setosa  lte5
2          4.6         3.1          1.5         0.2  Iris-setosa  lte5
3          5.0         3.6          1.4         0.2  Iris-setosa  lte5
4          5.4         3.9          1.7         0.4  Iris-setosa   gt5
Sql compiled:
CREATE TABLE tmp_pyodps_4bfb0aa0_7ec5_4f35_9480_d36baf2b8079 LIFECYCLE 1 AS
SELECT LN(t1.`sepallength` * 10) AS `sepallength`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
   sepallength
0     3.891820
1     3.850148
2     3.828641
3     3.912023
4     3.988984
Sql compiled:
CREATE TABLE tmp_pyodps_deb6fa45_fbb1_41d2_8174_76d35eebea9c LIFECYCLE 1 AS
SELECT t1.`sepallength`, t1.`sepallength` / 2 AS `sepallength/2`, POW(t1.`sepallength`, 2) AS `sepallength的平方`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
   sepallength  sepallength/2  sepallength的平方
0          4.9           2.45           24.01
1          4.7           2.35           22.09
2          4.6           2.30           21.16
3          5.0           2.50           25.00
4          5.4           2.70           29.16
Sql compiled:
CREATE TABLE tmp_pyodps_fa326546_c5e8_466f_915d_e783b7dcacdf LIFECYCLE 1 AS
SELECT t1.`sepallength` < 5 AS `sepallength`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
```

```
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
  sepallength
0        True
1        True
2        True
3       False
4       False
          a                                b  id
0  [a1, b1]  {u'c2': 2, u'a2': 0, u'b2': 1}   1
1      [c1]            {u'd2': 3, u'e2': 4}   2
odps.Schema {
  a   list<string>
  b   dict<string,int64>
  id  int64
}
          a                                b  id
0  [a1, b1]  {u'c2': 2, u'a2': 0, u'b2': 1}   1
1      [c1]            {u'd2': 3, u'e2': 4}   2
   id   a  b
0   1  a1  3
1   2  c1  2
    a
0  a1
1  b1
2  c1
   a_pos   a
0      0  a1
1      1  b1
2      0  c1
  b_key  b_value
0    c2        2
1    a2        0
2    b2        1
3    d2        3
4    e2        4
  key  value
0  c2      2
1  a2      0
2  b2      1
3  d2      3
4  e2      4
   id   a
0   1  a1
1   1  b1
2   2  c1
Sql compiled:
CREATE TABLE tmp_pyodps_fb28d9b9_fa45_4712_a564_dd751641286e LIFECYCLE 1 AS
SELECT t1.`sepallength` IN (4.9, 5.1) AS `sepallength`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
   sepallength
0         True
1        False
2        False
3        False
4        False
5        False
6        False
7        False
8         True
9        False
10       False
11       False
12       False
13       False
14       False
15       False
16        True
17       False
18        True
19       False
20        True
21       False
22        True
23       False
24       False
25       False
```

```
26       False
27       False
28       False
29       False
30       False
31       False
32       False
33        True
34       False
35       False
36        True
37       False
38        True
39       False
40       False
41       False
42       False
43        True
44       False
45        True
46       False
47       False
48       False
49       False
50       False
51       False
52       False
53       False
54       False
55       False
56        True
57       False
58       False
59       False
Sql compiled:
CREATE TABLE tmp_pyodps_1f85e8e1_a9f0_473f_ac0e_d526fdd80dc4 LIFECYCLE 1 AS
SELECT CASE WHEN (0 < t1.`sepallength`) AND (t1.`sepallength` <= 1) THEN '0-1' WHEN (1 < t1.`sepallength`) AND (t1.`sepalleng
th` <= 2) THEN '1-2' WHEN (2 < t1.`sepallength`) AND (t1.`sepallength` <= 3) THEN '2-3' WHEN (3 < t1.`sepallength`) AND (t1.`
sepallength` <= 4) THEN '3-4' WHEN (4 < t1.`sepallength`) AND (t1.`sepallength` <= 5) THEN '4-5' END AS `sepallength_cut`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
  sepallength_cut
0             4-5
1             4-5
2             4-5
3             4-5
4            None
Sql compiled:
CREATE TABLE tmp_pyodps_080a677e_904b_4edc_a961_160f8c664d7f LIFECYCLE 1 AS
SELECT CASE WHEN (0 < t1.`sepallength`) AND (t1.`sepallength` <= 1) THEN '0-1' WHEN (1 < t1.`sepallength`) AND (t1.`sepalleng
th` <= 2) THEN '1-2' WHEN (2 < t1.`sepallength`) AND (t1.`sepallength` <= 3) THEN '2-3' WHEN (3 < t1.`sepallength`) AND (t1.`
sepallength` <= 4) THEN '3-4' WHEN (4 < t1.`sepallength`) AND (t1.`sepallength` <= 5) THEN '4-5' WHEN 5 < t1.`sepallength` TH
EN '5-' END AS `sepallength_cut`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
```

5.9.9. PyDOPS的排序

本文为您介绍如何进行PyODPS的排序。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 创建表并导入数据。
    i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。

ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。

建表语句如下。

```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 单击工作空间后的**进入数据开发**。
5. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
6. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
7. 在PyODPS节点输入如下代码实现数据排序。

```
from  odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
#排序
print iris.sort('sepalwidth').head(5)
#降序排列两种方式
#设置参数ascending=False;进行降序排列
print iris.sort('sepalwidth',ascending=False).head(5)
#设置-实现降序排列
print iris.sort(-iris.sepalwidth).head(5)
#多字段排序
print iris.sort(['sepalwidth','petallength']).head(5)
#多字段排序时，如果是升序降序不同，ascending参数可以用于传入一个列表，长度必须等同于排序的字段，它们的值都是BOOLEAN类型
print iris.sort(['sepalwidth','petallength'],ascending=[True,False]).head(5)
print iris.sort(['sepalwidth',-iris.petallength]).head(5)
```

8. 单击**运行**。
9. 在**运行日志**中查看运行结果。



完整的运行结果如下。

```
Sql compiled:
CREATE TABLE tmp_pyodps_d1b06785_dc18_4288_ad34_de860de1be08 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth
LIMIT 10000
Instance ID: 20191010061554817gwml01im
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          5.0         1.5   Iris-virginica
```

```
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          5.5         2.3          4.0         1.3  Iris-versicolor
Sql compiled:
CREATE TABLE tmp_pyodps_3cb90bb2_fb95_43fb_ae84_f2b5a27d72dc LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth DESC
LIMIT 10000
Instance ID: 20191010061601287gs086792
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.7         4.4          1.5         0.4  Iris-setosa
1          5.5         4.2          1.4         0.2  Iris-setosa
2          5.2         4.1          1.5         0.1  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.4         3.9          1.3         0.4  Iris-setosa
Sql compiled:
CREATE TABLE tmp_pyodps_97b080bb_e014_48e8_a310_4b45fcd6a2ed LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth DESC
LIMIT 10000
Instance ID: 20191010061606927g6emz192
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.7         4.4          1.5         0.4  Iris-setosa
1          5.5         4.2          1.4         0.2  Iris-setosa
2          5.2         4.1          1.5         0.1  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.4         3.9          1.3         0.4  Iris-setosa
Sql compiled:
CREATE TABLE tmp_pyodps_6fe37b6e_6705_4052_b733_211eb9bd16ac LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth, petallength
LIMIT 10000
Instance ID: 20191010061611714gn586792
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          4.0         1.0  Iris-versicolor
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          5.0         1.5   Iris-virginica
4          4.5         2.3          1.3         0.3      Iris-setosa
Sql compiled:
CREATE TABLE tmp_pyodps_a52c805c_94a1_4a75_a6af_4fc9ed06ae68 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth, petallength DESC
LIMIT 10000
Instance ID: 20191010061616553gw3m9592
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          5.0         1.5   Iris-virginica
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          6.3         2.3          4.4         1.3  Iris-versicolor
Sql compiled:
CREATE TABLE tmp_pyodps_aac5538e_9b40_4078_b3c6_852b99c663c1 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth, petallength DESC
LIMIT 10000
Instance ID: 20191010061621329gvmkc292
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          5.0         1.5   Iris-virginica
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          6.3         2.3          4.4         1.3  Iris-versicolor
```

5.9.10. PyODPS的去重

本文为您介绍如何进行PyODPS的去重。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 创建表并导入数据。
   i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。
   ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。
   建表语句如下。

```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 单击工作空间后的**进入数据开发**。
5. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
6. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
7. 在PyODPS节点输入代码实现数据去重。
   示例代码如下。

```
from  odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
print iris[['name']].distinct()
print iris.distinct('name')
print iris.distinct('name','sepallength').head(3)
#您可以调用unique对Sequence进行去重操作，但是调用unique的Sequence不能用在列选择中
print iris.name.unique()
```

8. 单击**运行**。
9. 在**运行日志**中查看运行结果。



完整的运行结果如下。

```
Sql compiled:
CREATE TABLE tmp_pyodps_ed85ebd5_d678_44dd_9ece_bff1822376f6 LIFECYCLE 1 AS
SELECT DISTINCT t1.`name`
FROM WB_BestPractice_dev.`pyodps_iris` t1
Instance ID: 20191010063911142g2cp5692
             name
0      Iris-setosa
1  Iris-versicolor
2   Iris-virginica
Sql compiled:
CREATE TABLE tmp_pyodps_8ce6128f_9c6f_45af_b9de_c73ce9d5ba51 LIFECYCLE 1 AS
SELECT DISTINCT t1.`name`
FROM WB_BestPractice_dev.`pyodps_iris` t1
Instance ID: 20191010063915987gmuws592
             name
0      Iris-setosa
1  Iris-versicolor
2   Iris-virginica
Sql compiled:
CREATE TABLE tmp_pyodps_a3dc338e_0fea_4d5f_847c_79fb19ec1c72 LIFECYCLE 1 AS
SELECT DISTINCT t1.`name`, t1.`sepallength`
FROM WB_BestPractice_dev.`pyodps_iris` t1
Instance ID: 20191010063922l0gj056292
          name  sepallength
0  Iris-setosa          4.3
1  Iris-setosa          4.4
2  Iris-setosa          4.5
Sql compiled:
CREATE TABLE tmp_pyodps_bc0917bb_f10c_426b_9b75_47e94478382a LIFECYCLE 1 AS
SELECT t2.`name`
FROM (
  SELECT DISTINCT t1.`name`
  FROM WB_BestPractice_dev.`pyodps_iris` t1
) t2
Instance ID: 20191010063927189g9fsz192
             name
0      Iris-setosa
1  Iris-versicolor
2   Iris-virginica
```

5.9.11. PyODPS的采样

本文为您介绍如何进行PyODPS的采样。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 创建表并导入数据。
   i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。
   ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。
   建表语句如下。

```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 单击工作空间后的**进入数据开发**。
5. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。

6. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
7. 在PyODPS节点输入采样代码。

   示例代码如下。

```
#采样
from  odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
#按份数采样
print iris.sample(parts=10).head(5) #分为10，分默认取第0份
print iris.sample(parts=10,i=0).head(5) #手动指定取第0份
print iris.sample(parts=10,i=[2,5]).head(5)  #分成10份，取第2和第5份
print iris.sample(parts=10,columns=['name','sepalwidth']).head(5) #根据name和sepalwidth的值做采样
#按比例、条数采样
print iris.sample(n=100).head()  #选取100条数据
print iris.sample(frac=0.3).head() #采样30%的数据
#按权重列采样
print iris.sample(n=100,weights='sepallength').head()
print iris.sample(n=100,weights='sepalwidth',replace=True).head()
#分层采样
print iris.sample(strata='name',n={'Iris-setosa' : 10,'Iris-versicolor' : 10}).head()
print iris.sample(strata='name',frac={'Iris-setosa': 0.5,'Iris-versicolor': 0.4}).head()
```

8. 单击**运行**。



9. 在**运行日志**中查看运行结果。

   完整的运行结果如下。

```
Collection: ref_0
  odps.Table
    name: WB_BestPractice_dev.`pyodps_iris`
    schema:
      sepallength           : double      # 片长度(cm)
      sepalwidth            : double      # 片宽度(cm)
      petallength           : double      # 瓣长度(cm)
      petalwidth            : double      # 瓣宽度(cm)
      name                  : string      # 种类
Sample[collection]
  _input:
  _parts:
    Scalar[int8]
      10
  _i:
  _replace:
    Scalar[boolean]
      False
Collection: ref_0
  odps.Table
    name: WB_BestPractice_dev.`pyodps_iris`
    schema:
      sepallength           : double      # 片长度(cm)
      sepalwidth            : double      # 片宽度(cm)
      petallength           : double      # 瓣长度(cm)
```

```
      petalwidth            : double      # 瓣宽度(cm)
      name                  : string      # 种类
Sample[collection]
  _input:
  _parts:
    Scalar[int8]
      10
  _i:
  _replace:
    Scalar[boolean]
      False
Collection: ref_0
  odps.Table
    name: WB_BestPractice_dev.`pyodps_iris`
    schema:
      sepallength           : double      # 片长度(cm)
      sepalwidth            : double      # 片宽度(cm)
      petallength           : double      # 瓣长度(cm)
      petalwidth            : double      # 瓣宽度(cm)
      name                  : string      # 种类
Sample[collection]
  _input:
  _parts:
    Scalar[int8]
      10
  _i:
  _replace:
    Scalar[boolean]
      False
Collection: ref_0
  odps.Table
    name: WB_BestPractice_dev.`pyodps_iris`
    schema:
      sepallength           : double      # 片长度(cm)
      sepalwidth            : double      # 片宽度(cm)
      petallength           : double      # 瓣长度(cm)
      petalwidth            : double      # 瓣宽度(cm)
      name                  : string      # 种类
Sample[collection]
  _input:
  _parts:
    Scalar[int8]
      10
  _i:
  _sampled_fields:
    name = Column[sequence(string)] 'name' from collection ref_0
    sepalwidth = Column[sequence(float64)] 'sepalwidth' from collection ref_0
  _replace:
    Scalar[boolean]
      False
Executing RandomSample...
Command: PAI -name RandomSample -project algo_public -Dreplace="false" -Dlifecycle="1" -DoutputTableName="tmp_pyodps_15706900
14_69f3d75d_9537_4c9c_87ea_a5f6ad8d2e07" -DsampleSize="100" -DinputTableName="WB_BestPractice_dev.pyodps_iris";
Instance ID: 20191010064654985g6co9592
Sub Instance: create_output (20191010064700688g5cbn62m_71ee0561_bcc4_4147_b849_f74688353fb6)
Sub Instance: without_replacement (20191010064703694g9cbn62m_93a8a15b_ffd1_4afe_8928_19f28455a15c)
Try to fetch data from tunnel
    sepallength  sepalwidth  petallength  petalwidth           name
0           5.1         3.5          1.4         0.2    Iris-setosa
1           4.9         3.0          1.4         0.2    Iris-setosa
2           4.7         3.2          1.3         0.2    Iris-setosa
3           4.6         3.1          1.5         0.2    Iris-setosa
4           5.0         3.6          1.4         0.2    Iris-setosa
5           4.6         3.4          1.4         0.3    Iris-setosa
6           4.4         2.9          1.4         0.2    Iris-setosa
7           4.9         3.1          1.5         0.1    Iris-setosa
8           4.8         3.4          1.6         0.2    Iris-setosa
9           4.8         3.0          1.4         0.1    Iris-setosa
10          4.3         3.0          1.1         0.1    Iris-setosa
11          5.1         3.5          1.4         0.3    Iris-setosa
12          5.7         3.8          1.7         0.3    Iris-setosa
13          5.1         3.8          1.5         0.3    Iris-setosa
14          5.1         3.7          1.5         0.4    Iris-setosa
15          4.6         3.6          1.0         0.2    Iris-setosa
16          5.1         3.3          1.7         0.5    Iris-setosa
```

```
17          4.8         3.4          1.9         0.2     Iris-setosa
18          5.0         3.4          1.6         0.4     Iris-setosa
19          5.2         3.5          1.5         0.2     Iris-setosa
20          5.2         3.4          1.4         0.2     Iris-setosa
21          4.8         3.1          1.6         0.2     Iris-setosa
22          5.2         4.1          1.5         0.1     Iris-setosa
23          5.5         4.2          1.4         0.2     Iris-setosa
24          4.9         3.1          1.5         0.1     Iris-setosa
25          5.0         3.2          1.2         0.2     Iris-setosa
26          4.4         3.0          1.3         0.2     Iris-setosa
27          5.1         3.4          1.5         0.2     Iris-setosa
28          5.0         3.5          1.3         0.3     Iris-setosa
29          4.5         2.3          1.3         0.3     Iris-setosa
..          ...         ...          ...         ...              ...
70          7.1         3.0          5.9         2.1  Iris-virginica
71          7.6         3.0          6.6         2.1  Iris-virginica
72          7.3         2.9          6.3         1.8  Iris-virginica
73          7.2         3.6          6.1         2.5  Iris-virginica
74          6.5         3.2          5.1         2.0  Iris-virginica
75          6.8         3.0          5.5         2.1  Iris-virginica
76          5.8         2.8          5.1         2.4  Iris-virginica
77          7.7         3.8          6.7         2.2  Iris-virginica
78          7.7         2.6          6.9         2.3  Iris-virginica
79          7.7         2.8          6.7         2.0  Iris-virginica
80          6.3         2.7          4.9         1.8  Iris-virginica
81          6.7         3.3          5.7         2.1  Iris-virginica
82          6.2         2.8          4.8         1.8  Iris-virginica
83          6.1         3.0          4.9         1.8  Iris-virginica
84          6.4         2.8          5.6         2.1  Iris-virginica
85          7.2         3.0          5.8         1.6  Iris-virginica
86          7.4         2.8          6.1         1.9  Iris-virginica
87          7.9         3.8          6.4         2.0  Iris-virginica
88          6.3         2.8          5.1         1.5  Iris-virginica
89          6.3         3.4          5.6         2.4  Iris-virginica
90          6.4         3.1          5.5         1.8  Iris-virginica
91          6.0         3.0          4.8         1.8  Iris-virginica
92          6.9         3.1          5.4         2.1  Iris-virginica
93          6.9         3.1          5.1         2.3  Iris-virginica
94          5.8         2.7          5.1         1.9  Iris-virginica
95          6.8         3.2          5.9         2.3  Iris-virginica
96          6.7         3.3          5.7         2.5  Iris-virginica
97          6.3         2.5          5.0         1.9  Iris-virginica
98          6.2         3.4          5.4         2.3  Iris-virginica
99          5.9         3.0          5.1         1.8  Iris-virginica
[100 rows x 5 columns]
Executing RandomSample...
Command: PAI -name RandomSample -project algo_public -Dreplace="false" -DsampleRatio="0.3" -DoutputTableName="tmp_pyodps_1570
690039_e1867332_72ea_4656_928d_3bd6e31d87c7" -Dlifecycle="1" -DinputTableName="WB_BestPractice_dev.pyodps_iris";
Instance ID: 20191010064720117gmpms38
Sub Instance: create_output (20191010064725740grcbn62m_b338a671_6047_4360_8792_41d2e748e41f)
Sub Instance: without_replacement (20191010064728747gtcbn62m_6c9914da_d5c3_4336_b076_163edb1bf48a)
Try to fetch data from tunnel
    sepallength  sepalwidth  petallength  petalwidth            name
0           5.1         3.5          1.4         0.2     Iris-setosa
1           4.7         3.2          1.3         0.2     Iris-setosa
2           4.6         3.1          1.5         0.2     Iris-setosa
3           4.8         3.4          1.6         0.2     Iris-setosa
4           5.7         4.4          1.5         0.4     Iris-setosa
5           5.1         3.5          1.4         0.3     Iris-setosa
6           5.0         3.4          1.6         0.4     Iris-setosa
7           5.2         3.4          1.4         0.2     Iris-setosa
8           4.9         3.1          1.5         0.1     Iris-setosa
9           5.5         3.5          1.3         0.2     Iris-setosa
10          4.4         3.2          1.3         0.2     Iris-setosa
11          5.0         3.3          1.4         0.2     Iris-setosa
12          5.7         2.8          4.5         1.3 Iris-versicolor
13          5.2         2.7          3.9         1.4 Iris-versicolor
14          5.0         2.0          3.5         1.0 Iris-versicolor
15          5.6         2.9          3.6         1.3 Iris-versicolor
16          5.8         2.7          4.1         1.0 Iris-versicolor
17          6.1         2.8          4.0         1.3 Iris-versicolor
18          6.6         3.0          4.4         1.4 Iris-versicolor
19          6.8         2.8          4.8         1.4 Iris-versicolor
20          6.0         2.9          4.5         1.5 Iris-versicolor
21          5.4         3.0          4.5         1.5 Iris-versicolor
```

```
22          5.7         2.9          4.2         1.3  Iris-versicolor
23          5.7         2.8          4.1         1.3  Iris-versicolor
24          6.3         2.9          5.6         1.8   Iris-virginica
25          4.9         2.5          4.5         1.7   Iris-virginica
26          6.7         2.5          5.8         1.8   Iris-virginica
27          6.4         2.7          5.3         1.9   Iris-virginica
28          6.8         3.0          5.5         2.1   Iris-virginica
29          5.7         2.5          5.0         2.0   Iris-virginica
30          5.8         2.8          5.1         2.4   Iris-virginica
31          6.5         3.0          5.5         1.8   Iris-virginica
32          6.0         2.2          5.0         1.5   Iris-virginica
33          6.3         2.7          4.9         1.8   Iris-virginica
34          7.2         3.2          6.0         1.8   Iris-virginica
35          6.2         2.8          4.8         1.8   Iris-virginica
36          6.1         3.0          4.9         1.8   Iris-virginica
37          6.4         2.8          5.6         2.1   Iris-virginica
38          7.2         3.0          5.8         1.6   Iris-virginica
39          6.3         2.8          5.1         1.5   Iris-virginica
40          6.4         3.1          5.5         1.8   Iris-virginica
41          6.0         3.0          4.8         1.8   Iris-virginica
42          6.8         3.2          5.9         2.3   Iris-virginica
43          6.3         2.5          5.0         1.9   Iris-virginica
44          6.2         3.4          5.4         2.3   Iris-virginica
Executing WeightedSample...
Command: PAI -name WeightedSample -project algo_public -DinputTableName="WB_BestPractice_dev.pyodps_iris" -DsampleSize="100"
-DprobCol="sepallength" -Dreplace="false" -DoutputTableName="tmp_pyodps_1570690063_6a62857e_8f85_4ea7_99ef_08aa259546d4" -Dli
fecycle="1";
Instance ID: 20191010064743533gnpms38
Sub Instance: create_output (20191010064748787gkdbn62m_8d47bfb7_e470_4cce_8b69_28811f190083)
Sub Instance: without_replacement (20191010064751793gmdbn62m_230a1d26_5c2e_440e_a31d_fe9e63c6f906)
Try to fetch data from tunnel
    sepallength  sepalwidth  petallength  petalwidth            name
0           4.9         3.0          1.4         0.2     Iris-setosa
1           4.7         3.2          1.3         0.2     Iris-setosa
2           5.0         3.6          1.4         0.2     Iris-setosa
3           5.4         3.9          1.7         0.4     Iris-setosa
4           5.0         3.4          1.5         0.2     Iris-setosa
5           4.4         2.9          1.4         0.2     Iris-setosa
6           4.8         3.4          1.6         0.2     Iris-setosa
7           4.8         3.0          1.4         0.1     Iris-setosa
8           5.4         3.9          1.3         0.4     Iris-setosa
9           5.1         3.5          1.4         0.3     Iris-setosa
10          5.7         3.8          1.7         0.3     Iris-setosa
11          4.6         3.6          1.0         0.2     Iris-setosa
12          5.0         3.4          1.6         0.4     Iris-setosa
13          5.2         3.5          1.5         0.2     Iris-setosa
14          5.2         3.4          1.4         0.2     Iris-setosa
15          4.7         3.2          1.6         0.2     Iris-setosa
16          4.8         3.1          1.6         0.2     Iris-setosa
17          5.5         4.2          1.4         0.2     Iris-setosa
18          4.9         3.1          1.5         0.1     Iris-setosa
19          5.0         3.2          1.2         0.2     Iris-setosa
20          5.5         3.5          1.3         0.2     Iris-setosa
21          4.9         3.1          1.5         0.1     Iris-setosa
22          5.1         3.4          1.5         0.2     Iris-setosa
23          4.5         2.3          1.3         0.3     Iris-setosa
24          4.8         3.0          1.4         0.3     Iris-setosa
25          5.1         3.8          1.6         0.2     Iris-setosa
26          4.6         3.2          1.4         0.2     Iris-setosa
27          5.3         3.7          1.5         0.2     Iris-setosa
28          5.0         3.3          1.4         0.2     Iris-setosa
29          7.0         3.2          4.7         1.4  Iris-versicolor
..          ...         ...          ...         ...             ...
70          7.2         3.6          6.1         2.5   Iris-virginica
71          6.4         2.7          5.3         1.9   Iris-virginica
72          5.7         2.5          5.0         2.0   Iris-virginica
73          5.8         2.8          5.1         2.4   Iris-virginica
74          6.4         3.2          5.3         2.3   Iris-virginica
75          7.7         3.8          6.7         2.2   Iris-virginica
76          6.9         3.2          5.7         2.3   Iris-virginica
77          5.6         2.8          4.9         2.0   Iris-virginica
78          7.7         2.8          6.7         2.0   Iris-virginica
79          6.3         2.7          4.9         1.8   Iris-virginica
80          6.7         3.3          5.7         2.1   Iris-virginica
81          7.2         3.2          6.0         1.8   Iris-virginica
```

```
82          6.2         2.8          4.8         1.8   Iris-virginica
83          6.1         3.0          4.9         1.8   Iris-virginica
84          7.2         3.0          5.8         1.6   Iris-virginica
85          7.9         3.8          6.4         2.0   Iris-virginica
86          6.4         2.8          5.6         2.2   Iris-virginica
87          6.3         2.8          5.1         1.5   Iris-virginica
88          6.1         2.6          5.6         1.4   Iris-virginica
89          6.3         3.4          5.6         2.4   Iris-virginica
90          6.4         3.1          5.5         1.8   Iris-virginica
91          6.0         3.0          4.8         1.8   Iris-virginica
92          6.9         3.1          5.1         2.3   Iris-virginica
93          5.8         2.7          5.1         1.9   Iris-virginica
94          6.8         3.2          5.9         2.3   Iris-virginica
95          6.7         3.3          5.7         2.5   Iris-virginica
96          6.7         3.0          5.2         2.3   Iris-virginica
97          6.5         3.0          5.2         2.0   Iris-virginica
98          6.2         3.4          5.4         2.3   Iris-virginica
99          5.9         3.0          5.1         1.8   Iris-virginica
[100 rows x 5 columns]
Executing WeightedSample...
Command: PAI -name WeightedSample -project algo_public -DinputTableName="WB_BestPractice_dev.pyodps_iris" -DsampleSize="100"
-DprobCol="sepalwidth" -Dreplace="true" -DoutputTableName="tmp_pyodps_1570690082_f55e899c_3cb4_4eeb_ade4_b8cb79e018dc" -Dlife
cycle="1";
Instance ID: 20191010064480392g9ers38
Sub Instance: create_output (20191010064808827g9ebn62m_fb70c859_913a_4830_9248_8c8eaf134f1d)
Sub Instance: with_replacement (20191010064811833gdebn62m_07544cc2_1d7d_4fa5_972d_196eb6b9f537)
    sepallength  sepalwidth  petallength  petalwidth            name
0           5.1         3.5          1.4         0.2     Iris-setosa
1           4.6         3.4          1.4         0.3     Iris-setosa
2           5.0         3.4          1.5         0.2     Iris-setosa
3           5.0         3.4          1.5         0.2     Iris-setosa
4           4.4         2.9          1.4         0.2     Iris-setosa
5           4.8         3.4          1.6         0.2     Iris-setosa
6           5.8         4.0          1.2         0.2     Iris-setosa
7           5.8         4.0          1.2         0.2     Iris-setosa
8           5.1         3.5          1.4         0.3     Iris-setosa
9           5.1         3.5          1.4         0.3     Iris-setosa
10          5.1         3.5          1.4         0.3     Iris-setosa
11          5.1         3.7          1.5         0.4     Iris-setosa
12          4.6         3.6          1.0         0.2     Iris-setosa
13          4.8         3.4          1.9         0.2     Iris-setosa
14          5.0         3.0          1.6         0.2     Iris-setosa
15          5.0         3.4          1.6         0.4     Iris-setosa
16          5.2         3.4          1.4         0.2     Iris-setosa
17          4.8         3.1          1.6         0.2     Iris-setosa
18          4.8         3.1          1.6         0.2     Iris-setosa
19          5.4         3.4          1.5         0.4     Iris-setosa
20          5.4         3.4          1.5         0.4     Iris-setosa
21          5.4         3.4          1.5         0.4     Iris-setosa
22          5.2         4.1          1.5         0.1     Iris-setosa
23          5.2         4.1          1.5         0.1     Iris-setosa
24          5.5         4.2          1.4         0.2     Iris-setosa
25          5.0         3.2          1.2         0.2     Iris-setosa
26          4.9         3.1          1.5         0.1     Iris-setosa
27          4.4         3.0          1.3         0.2     Iris-setosa
28          5.1         3.4          1.5         0.2     Iris-setosa
29          5.0         3.5          1.3         0.3     Iris-setosa
..          ...         ...          ...         ...             ...
70          5.6         2.7          4.2         1.3  Iris-versicolor
71          5.7         2.9          4.2         1.3  Iris-versicolor
72          6.3         3.3          6.0         2.5   Iris-virginica
73          7.1         3.0          5.9         2.1   Iris-virginica
74          6.3         2.9          5.6         1.8   Iris-virginica
75          7.3         2.9          6.3         1.8   Iris-virginica
76          7.2         3.6          6.1         2.5   Iris-virginica
77          6.4         2.7          5.3         1.9   Iris-virginica
78          6.8         3.0          5.5         2.1   Iris-virginica
79          6.8         3.0          5.5         2.1   Iris-virginica
80          5.8         2.8          5.1         2.4   Iris-virginica
81          6.2         2.8          4.8         1.8   Iris-virginica
82          6.2         2.8          4.8         1.8   Iris-virginica
83          6.1         3.0          4.9         1.8   Iris-virginica
84          6.4         2.8          5.6         2.1   Iris-virginica
85          7.2         3.0          5.8         1.6   Iris-virginica
86          7.4         2.8          6.1         1.9   Iris-virginica
```

```
87          7.4         2.8          6.1         1.9   Iris-virginica
88          7.4         2.8          6.1         1.9   Iris-virginica
89          6.4         3.1          5.5         1.8   Iris-virginica
90          6.0         3.0          4.8         1.8   Iris-virginica
91          6.9         3.1          5.4         2.1   Iris-virginica
92          6.7         3.1          5.6         2.4   Iris-virginica
93          6.7         3.1          5.6         2.4   Iris-virginica
94          6.7         3.1          5.6         2.4   Iris-virginica
95          6.9         3.1          5.1         2.3   Iris-virginica
96          6.9         3.1          5.1         2.3   Iris-virginica
97          6.7         3.0          5.2         2.3   Iris-virginica
98          6.5         3.0          5.2         2.0   Iris-virginica
99          5.9         3.0          5.1         1.8   Iris-virginica
[100 rows x 5 columns]
Executing StratifiedSample...
Command: PAI -name StratifiedSample -project algo_public -Dlifecycle="1" -DoutputTableName="tmp_pyodps_1570690104_6cc52795_2a
86_4634_a905_740b3a426d3f" -DsampleSize="Iris-setosa:10,Iris-versicolor:10" -DstrataColName="name" -DinputTableName="WB_BestP
ractice_dev.pyodps_iris";
Instance ID: 20191010064824633gaco9592
Sub Instance: create_output (20191010064829870g4fbn62m_dfa297c5_23b5_43f5_bb17_83ba8d263630)
Sub Instance: stratified_sampling (20191010064831874g7fbn62m_fc9eddb7_42f1_49fe_8206_891ef451fb76)
Try to fetch data from tunnel
    sepallength  sepalwidth  petallength  petalwidth            name
0           5.4         3.9          1.7         0.4     Iris-setosa
1           4.3         3.0          1.1         0.1     Iris-setosa
2           5.4         3.9          1.3         0.4     Iris-setosa
3           5.1         3.3          1.7         0.5     Iris-setosa
4           4.7         3.2          1.6         0.2     Iris-setosa
5           4.5         2.3          1.3         0.3     Iris-setosa
6           5.0         3.5          1.6         0.6     Iris-setosa
7           5.1         3.8          1.9         0.4     Iris-setosa
8           4.8         3.0          1.4         0.3     Iris-setosa
9           5.0         3.3          1.4         0.2     Iris-setosa
10          7.0         3.2          4.7         1.4  Iris-versicolor
11          5.5         2.3          4.0         1.3  Iris-versicolor
12          6.5         2.8          4.6         1.5  Iris-versicolor
13          5.6         3.0          4.5         1.5  Iris-versicolor
14          5.7         2.6          3.5         1.0  Iris-versicolor
15          5.5         2.4          3.7         1.0  Iris-versicolor
16          5.0         2.3          3.3         1.0  Iris-versicolor
17          5.6         2.7          4.2         1.3  Iris-versicolor
18          5.7         3.0          4.2         1.2  Iris-versicolor
19          5.1         2.5          3.0         1.1  Iris-versicolor
Executing StratifiedSample...
Command: PAI -name StratifiedSample -project algo_public -DsampleRatio="Iris-setosa:0.5,Iris-versicolor:0.4" -DoutputTableNam
e="tmp_pyodps_1570690128_a68477cd_19e5_4fe0_bb39_4712f76dd967" -Dlifecycle="1" -DstrataColName="name" -DinputTableName="WB_Be
stPractice_dev.pyodps_iris";
Instance ID: 20191010064848733gbers38
Sub Instance: create_output (20191010064853918gwfbn62m_4eb22c22_7051_4372_8d13_05c5a417aa87)
Sub Instance: stratified_sampling (20191010064855924g0gbn62m_b4242ac7_bd5a_47a8_a1f2_3367a6a101a7)
Try to fetch data from tunnel
    sepallength  sepalwidth  petallength  petalwidth            name
0           4.9         3.0          1.4         0.2     Iris-setosa
1           4.7         3.2          1.3         0.2     Iris-setosa
2           5.0         3.6          1.4         0.2     Iris-setosa
3           5.4         3.9          1.7         0.4     Iris-setosa
4           5.0         3.4          1.5         0.2     Iris-setosa
5           5.4         3.7          1.5         0.2     Iris-setosa
6           4.8         3.4          1.6         0.2     Iris-setosa
7           4.8         3.0          1.4         0.1     Iris-setosa
8           5.8         4.0          1.2         0.2     Iris-setosa
9           5.4         3.4          1.7         0.2     Iris-setosa
10          5.1         3.7          1.5         0.4     Iris-setosa
11          4.8         3.4          1.9         0.2     Iris-setosa
12          5.0         3.0          1.6         0.2     Iris-setosa
13          5.0         3.4          1.6         0.4     Iris-setosa
14          5.2         3.5          1.5         0.2     Iris-setosa
15          5.2         3.4          1.4         0.2     Iris-setosa
16          4.7         3.2          1.6         0.2     Iris-setosa
17          5.2         4.1          1.5         0.1     Iris-setosa
18          5.0         3.2          1.2         0.2     Iris-setosa
19          5.1         3.4          1.5         0.2     Iris-setosa
20          4.5         2.3          1.3         0.3     Iris-setosa
21          5.0         3.5          1.6         0.6     Iris-setosa
22          5.1         3.8          1.9         0.4     Iris-setosa
```

```
23          5.1         3.8          1.6         0.2      Iris-setosa
24          5.3         3.7          1.5         0.2      Iris-setosa
25          7.0         3.2          4.7         1.4  Iris-versicolor
26          6.4         3.2          4.5         1.5  Iris-versicolor
27          6.9         3.1          4.9         1.5  Iris-versicolor
28          6.5         2.8          4.6         1.5  Iris-versicolor
29          5.7         2.8          4.5         1.3  Iris-versicolor
30          6.6         2.9          4.6         1.3  Iris-versicolor
31          5.6         2.9          3.6         1.3  Iris-versicolor
32          5.6         3.0          4.5         1.5  Iris-versicolor
33          5.6         2.5          3.9         1.1  Iris-versicolor
34          6.1         2.8          4.7         1.2  Iris-versicolor
35          6.8         2.8          4.8         1.4  Iris-versicolor
36          5.5         2.4          3.8         1.1  Iris-versicolor
37          5.5         2.4          3.7         1.0  Iris-versicolor
38          6.0         2.7          5.1         1.6  Iris-versicolor
39          5.6         3.0          4.1         1.3  Iris-versicolor
40          5.5         2.6          4.4         1.2  Iris-versicolor
41          6.1         3.0          4.6         1.4  Iris-versicolor
42          5.7         3.0          4.2         1.2  Iris-versicolor
43          5.7         2.9          4.2         1.3  Iris-versicolor
44          6.2         2.9          4.3         1.3  Iris-versicolor
```

5.9.12. PyODPS的数据缩放

本文介绍如何进行PyODPS的数据缩放。

前提条件

请提前完成如下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

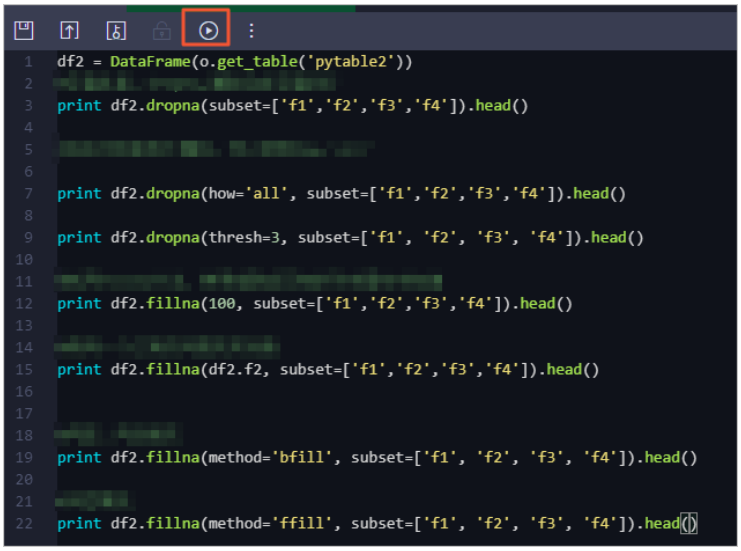
1. 创建表并导入数据。
   i. 下载鸢尾花数据集*iris.data*，重命名为*iris.csv*。
   ii. 创建表pyodps_iris并上传数据集*iris.csv*。操作方法请参见建表并上传数据。
   建表语句如下。

```
CREATE TABLE if not exists pyodps_iris
(
sepallength  DOUBLE comment '片长度(cm)',
sepalwidth   DOUBLE comment '片宽度(cm)',
petallength  DOUBLE comment '瓣长度(cm)',
petalwidth   DOUBLE comment '瓣宽度(cm)',
name         STRING comment '种类'
);
```

2. 登录DataWorks控制台。
3. 在左侧导航栏上单击**工作空间列表**。
4. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
5. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
6. 在PyODPS 2节点中输入数据缩放代码。
   示例代码如下。

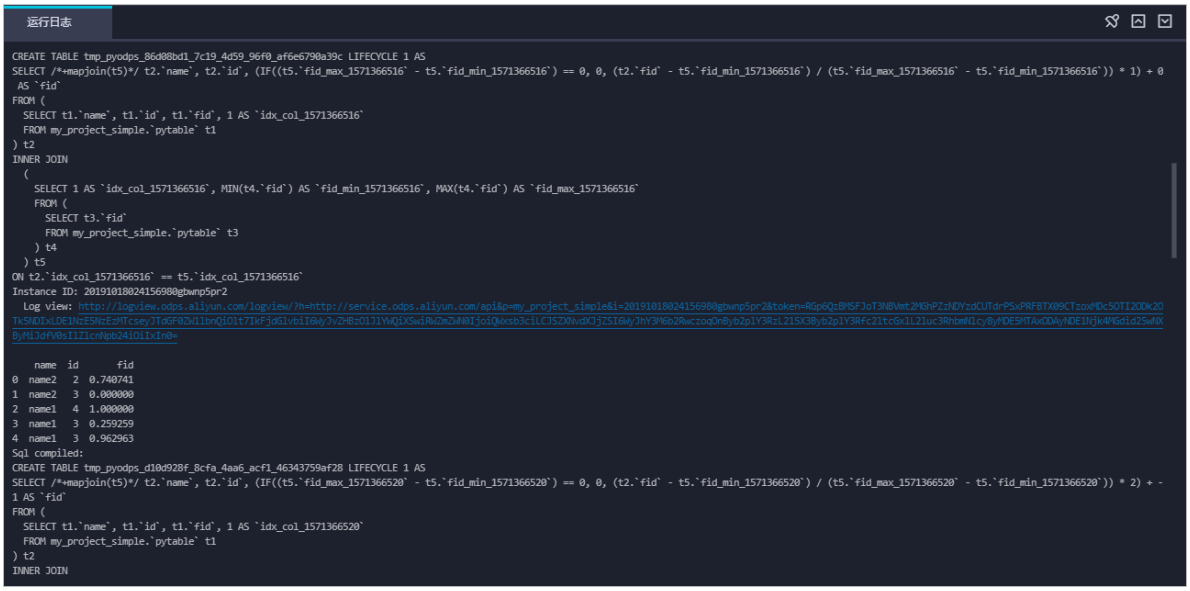
```
#数据缩放
df = DataFrame(o.get_table('pytable'))
#使用min_max_scale方法进行归一化
print df.min_max_scale(columns=['fid']).head()
#min_max_scale还支持使用feature_range参数指定输出值的范围。例如，需要使输出值在 (-1, 1) 范围内
print df.min_max_scale(columns=['fid'],feature_range=(-1,1)).head()
#如果需要保留原始值，可以使用preserve参数。此时，缩放后的数据将会以新增列的形式追加到数据中，列名默认为原列名追加_scaled后缀，该后缀可使用suffix参数更改
print df.min_max_scale(columns=['fid'],preserve=True).head()
#使用group分组
print df.min_max_scale(columns=['fid'],group=['name']).head()
#min_max_scale也支持使用group参数指定一个或多个分组列，在分组列中分别取最值进行缩放
print df.std_scale(columns=['fid']).head()
```

7. 单击**运行**。



```
df2 = DataFrame(o.get_table('pytable2'))

print df2.dropna(subset=['f1','f2','f3','f4']).head()


print df2.dropna(how='all', subset=['f1','f2','f3','f4']).head()

print df2.dropna(thresh=3, subset=['f1', 'f2', 'f3', 'f4']).head()


print df2.fillna(100, subset=['f1','f2','f3','f4']).head()


print df2.fillna(df2.f2, subset=['f1','f2','f3','f4']).head()


print df2.fillna(method='bfill', subset=['f1', 'f2', 'f3', 'f4']).head()


print df2.fillna(method='ffill', subset=['f1', 'f2', 'f3', 'f4']).head()
```

8. 在**运行日志**中查看运行结果。



```
运行日志

CREATE TABLE tmp_pyodps_86d08bd1_7c19_4d59_96f0_af6e6790a39c LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, (IF((t5.`fid_max_1571366516` - t5.`fid_min_1571366516`) == 0, 0, (t2.`fid` - t5.`fid_min_1571366516`) / (t5.`fid_max_1571366516` - t5.`fid_min_1571366516`)) * 1) + 0
 AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1571366516`
  FROM my_project_simple.`pytable` t1
) t2
INNER JOIN
  (
    SELECT 1 AS `idx_col_1571366516`, MIN(t4.`fid`) AS `fid_min_1571366516`, MAX(t4.`fid`) AS `fid_max_1571366516`
    FROM (
      SELECT t3.`fid`
      FROM my_project_simple.`pytable` t3
    ) t4
  ) t5
ON t2.`idx_col_1571366516` == t5.`idx_col_1571366516`
Instance ID: 20191018024156980gbwnp5pr2
  Log view: http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=my_project_simple&i=20191018024156980gbwnp5pr2&token=RGp6QzBMSFJoT3NBVmt2MGhPZzNDYzdCUTdrPSxPRFBTX09CTzoxMDc5OTI2ODk2O
Tk5NDIxLDE1NzE5NzEzMTcseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL215X3Byb2plY3Rfc21tcGxlL2luc3RhbmNlcy8yMDE5MTAxODAyNDE1Njk4MGdid25wNX
ByMiJdfV0sI1Z1cnNpb24iOiIxIn0=

    name  id       fid
0  name2   2  0.740741
1  name2   3  0.000000
2  name1   4  1.000000
3  name1   3  0.259259
4  name1   3  0.962963
Sql compiled:
CREATE TABLE tmp_pyodps_d10d928f_8cfa_4aa6_acf1_46343759af28 LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, (IF((t5.`fid_max_1571366520` - t5.`fid_min_1571366520`) == 0, 0, (t2.`fid` - t5.`fid_min_1571366520`) / (t5.`fid_max_1571366520` - t5.`fid_min_1571366520`)) * 2) + -
1 AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1571366520`
  FROM my_project_simple.`pytable` t1
) t2
INNER JOIN
```

完整的运行结果如下。

```
Sql compiled:
CREATE TABLE tmp_pyodps_59cb5533_6dc3_4b64_9312_b2da29d70cee LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, (IF((t5.`fid_max_1570691133` - t5.`fid_min_1570691133`) == 0, 0, (t2.`fid` - t5.`
fid_min_1570691133`) / (t5.`fid_max_1570691133` - t5.`fid_min_1570691133`)) * 1) + 0 AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691133`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
  (
    SELECT 1 AS `idx_col_1570691133`, MIN(t4.`fid`) AS `fid_min_1570691133`, MAX(t4.`fid`) AS `fid_max_1570691133`
    FROM (
      SELECT t3.`fid`
      FROM WB_BestPractice_dev.`pytable` t3
    ) t4
  ) t5
ON t2.`idx_col_1570691133` == t5.`idx_col_1570691133`
Instance ID: 2019101007053495gl8by72m
    name  id       fid
0  name1   4  1.000000
1  name2   2  0.526316
2  name2   3  0.000000
3  name1   4  0.710526
```

```
4  name1   3  0.184211
5  name1   3  0.684211
Sql compiled:
CREATE TABLE tmp_pyodps_88a24967_3bdc_41ce_85d7_4bbd23e4ce01 LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, (IF((t5.`fid_max_1570691139` - t5.`fid_min_1570691139`) == 0, 0, (t2.`fid` - t5.`
fid_min_1570691139`) / (t5.`fid_max_1570691139` - t5.`fid_min_1570691139`)) * 2) + -1 AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691139`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
  (
    SELECT 1 AS `idx_col_1570691139`, MIN(t4.`fid`) AS `fid_min_1570691139`, MAX(t4.`fid`) AS `fid_max_1570691139`
    FROM (
      SELECT t3.`fid`
      FROM WB_BestPractice_dev.`pytable` t3
    ) t4
  ) t5
ON t2.`idx_col_1570691139` == t5.`idx_col_1570691139`
Instance ID: 20191010070539772gjo56292
    name  id       fid
0  name1   4  1.000000
1  name2   2  0.052632
2  name2   3 -1.000000
3  name1   4  0.421053
4  name1   3 -0.631579
5  name1   3  0.368421
Sql compiled:
CREATE TABLE tmp_pyodps_439117fc_9ef7_4086_899d_a5bf77d653e5 LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, t2.`fid`, (IF((t5.`fid_max_1570691146` - t5.`fid_min_1570691146`) == 0, 0, (t2.`f
id` - t5.`fid_min_1570691146`) / (t5.`fid_max_1570691146` - t5.`fid_min_1570691146`)) * 1) + 0 AS `fid_scaled`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691146`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
  (
    SELECT 1 AS `idx_col_1570691146`, MIN(t4.`fid`) AS `fid_min_1570691146`, MAX(t4.`fid`) AS `fid_max_1570691146`
    FROM (
      SELECT t3.`fid`
      FROM WB_BestPractice_dev.`pytable` t3
    ) t4
  ) t5
ON t2.`idx_col_1570691146` == t5.`idx_col_1570691146`
Instance ID: 20191010070546769g0c14f72
    name  id  fid  fid_scaled
0  name1   4  5.3    1.000000
1  name2   2  3.5    0.526316
2  name2   3  1.5    0.000000
3  name1   4  4.2    0.710526
4  name1   3  2.2    0.184211
5  name1   3  4.1    0.684211
Sql compiled:
CREATE TABLE tmp_pyodps_d3839b4b_1087_4d52_91f5_52763b72f272 LIFECYCLE 1 AS
SELECT /*+mapjoin(t3)*/ t1.`name`, t1.`id`, (IF((t3.`fid_max_1570691151` - t3.`fid_min_1570691151`) == 0, 0, (t1.`fid` - t3.`
fid_min_1570691151`) / (t3.`fid_max_1570691151` - t3.`fid_min_1570691151`)) * 1) + 0 AS `fid`
FROM WB_BestPractice_dev.`pytable` t1
INNER JOIN
  (
    SELECT t2.`name`, MAX(t2.`fid`) AS `fid_max_1570691151`, MIN(t2.`fid`) AS `fid_min_1570691151`
    FROM WB_BestPractice_dev.`pytable` t2
    GROUP BY t2.`name`
  ) t3
ON t1.`name` == t3.`name`
Instance ID: 20191010070551756gtf14f72
    name  id       fid
0  name1   4  1.000000
1  name2   2  1.000000
2  name2   3  0.000000
3  name1   4  0.645161
4  name1   3  0.000000
5  name1   3  0.612903
Sql compiled:
CREATE TABLE tmp_pyodps_1ea6e5b4_129f_4d1e_a6a7_410e08d77ae6 LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, IF(t5.`fid_std_1570691157` == 0, 0, (t2.`fid` - t5.`fid_mean_1570691157`) / t5.`f
```

```
id_std_1570691157`) AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691157`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
  (
    SELECT 1 AS `idx_col_1570691157`, AVG(t4.`fid`) AS `fid_mean_1570691157`, STDDEV(t4.`fid`) AS `fid_std_1570691157`
    FROM (
      SELECT t3.`fid`
      FROM WB_BestPractice_dev.`pytable` t3
    ) t4
  ) t5
ON t2.`idx_col_1570691157` == t5.`idx_col_1570691157`
Instance ID: 20191010070557788gdl14f72
    name  id       fid
0  name1   4  1.436467
1  name2   2  0.026118
2  name2   3 -1.540938
3  name1   4  0.574587
4  name1   3 -0.992468
5  name1   3  0.496234
```

5.9.13. PyODPS的空值处理

本文为您介绍如何使用DataWorks的PyODPS节点处理PyODPS的空值。

前提条件

您需要完成以下操作：

- 已开通MaxCompute。
- 已开通DataWorks。
- 在DataWorks上完成业务流程创建，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 准备测试数据。
   i. 登录DataWorks控制台。
   ii. 创建表并上传数据。操作方法请参见建表并上传数据。
   表pytable2的建表语句如下。

```
CREATE TABLE `pytable2` (
  `id` string,
  `name` string,
  `f1` double,
  `f2` double,
  `f3` double,
  `f4` double
) ;
```

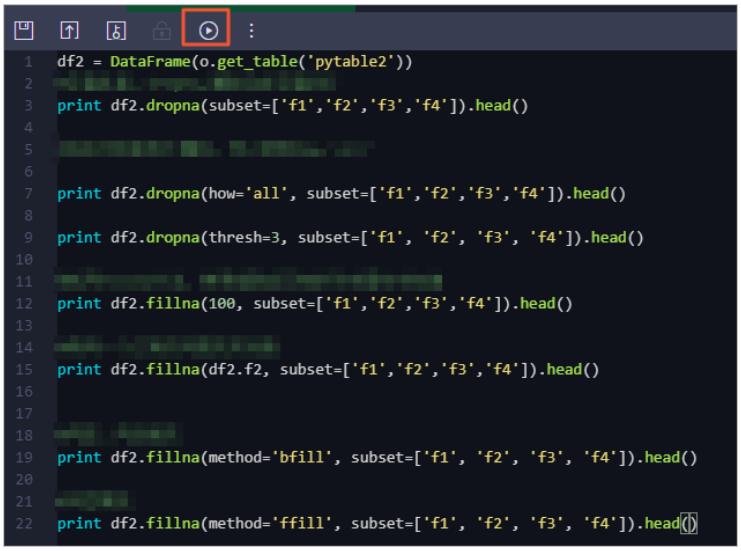
   示例数据文件pytable2.txt的内容如下。

```
0, name1, 1.0, NaN, 3.0, 4.0
1, name1, 2.0, NaN, NaN, 1.0
2, name1, 3.0, 4.0, 1.0, NaN
3, name1, NaN, 1.0, 2.0, 3.0
4, name1, 1.0, NaN, 3.0, 4.0
5, name1, 1.0, 2.0, 3.0, 4.0
6, name1, NaN, NaN, NaN, NaN
```

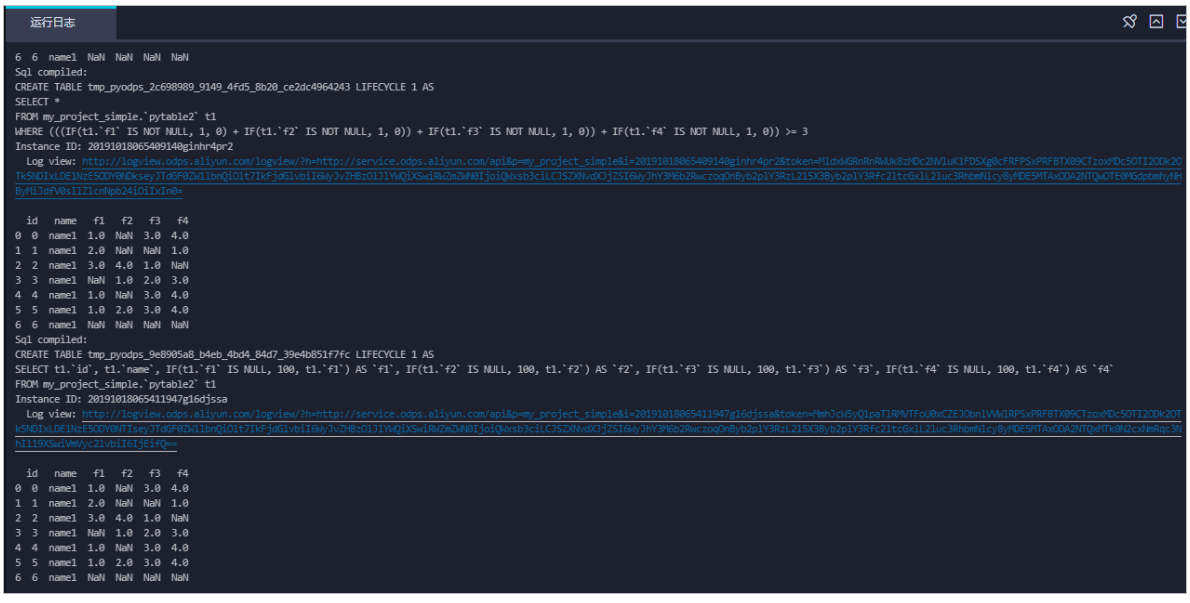
2. 在左侧导航栏上单击**工作空间列表**。
3. 单击工作空间后的**进入数据开发**。
4. 在数据开发页面，右键单击已经创建的业务流程，选择**新建 > MaxCompute > PyODPS 2**。
5. 在**新建节点**对话框，输入**节点名称**，并单击**提交**。
6. 在PyODPS 2节点中输入空值处理代码。
   示例代码如下。

```
df2 = DataFrame(o.get_table('pytable2'))
#dropna方法，删除包含空值的行。
print df2.dropna(subset=['f1','f2','f3','f4']).head()
#包含非空值则不删除，可以使用how='all'。
print df2.dropna(how='all', subset=['f1','f2','f3','f4']).head()
print df2.dropna(thresh=3, subset=['f1', 'f2', 'f3', 'f4']).head()
#fillna方法，使用常数或已有的列填充未知值。
print df2.fillna(100, subset=['f1','f2','f3','f4']).head()
#使用一个已有的列填充未知值。
print df2.fillna(df2.f2, subset=['f1','f2','f3','f4']).head()
#向前填充。
print df2.fillna(method='bfill', subset=['f1', 'f2', 'f3', 'f4']).head()
#向后填充。
print df2.fillna(method='ffill', subset=['f1', 'f2', 'f3', 'f4']).head()
```

7. 单击**运行**。



8. 在**运行日志**中查看运行结果。



完整的运行结果如下。

```
Sql compiled:
CREATE TABLE tmp_pyodps_d0c7d8c2_be38_4d48_b0eb_e89bae5bde01 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pytable2` t1
WHERE (((IF(t1.`f1` IS NOT NULL, 1, 0) + IF(t1.`f2` IS NOT NULL, 1, 0)) + IF(t1.`f3` IS NOT NULL, 1, 0)) + IF(t1.`f4` IS NOT
NULL, 1, 0)) >= 4
Instance ID: 20191010071154980g2hic292
  id   name   f1   f2   f3   f4
```

```
  id   name   f1   f2   f3   f4
0  0  name1  1.0  NaN  3.0  4.0
1  1  name1  2.0  NaN  NaN  1.0
2  2  name1  3.0  4.0  1.0  NaN
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0  NaN  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
Sql compiled:
CREATE TABLE tmp_pyodps_49b46768_f589_48f6_be8a_b7139f31f6f2 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pytable2` t1
WHERE (((IF(t1.`f1` IS NOT NULL, 1, 0) + IF(t1.`f2` IS NOT NULL, 1, 0)) + IF(t1.`f3` IS NOT NULL, 1, 0)) + IF(t1.`f4` IS NOT
NULL, 1, 0)) >= 1
Instance ID: 20191010071159759g0dk9592
  id   name   f1   f2   f3   f4
0  0  name1  1.0  NaN  3.0  4.0
1  1  name1  2.0  NaN  NaN  1.0
2  2  name1  3.0  4.0  1.0  NaN
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0  NaN  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
Sql compiled:
CREATE TABLE tmp_pyodps_7f941800_1539_415b_9257_283ebeb893a6 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pytable2` t1
WHERE (((IF(t1.`f1` IS NOT NULL, 1, 0) + IF(t1.`f2` IS NOT NULL, 1, 0)) + IF(t1.`f3` IS NOT NULL, 1, 0)) + IF(t1.`f4` IS NOT
NULL, 1, 0)) >= 3
Instance ID: 20191010071204544giyswx7
0  0  name1  1.0  NaN  3.0  4.0
1  1  name1  2.0  NaN  NaN  1.0
2  2  name1  3.0  4.0  1.0  NaN
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0  NaN  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
Sql compiled:
CREATE TABLE tmp_pyodps_16d6ea6d_5195_4e4c_8346_644a395852f7 LIFECYCLE 1 AS
SELECT t1.`id`, t1.`name`, IF(t1.`f1` IS NULL, 100, t1.`f1`) AS `f1`, IF(t1.`f2` IS NULL, 100, t1.`f2`) AS `f2`, IF(t1.`f3` I
S NULL, 100, t1.`f3`) AS `f3`, IF(t1.`f4` IS NULL, 100, t1.`f4`) AS `f4`
FROM WB_BestPractice_dev.`pytable2` t1
Instance ID: 20191010071209190gyl56292
  id   name   f1   f2   f3   f4
0  0  name1  1.0  NaN  3.0  4.0
1  1  name1  2.0  NaN  NaN  1.0
2  2  name1  3.0  4.0  1.0  NaN
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0  NaN  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
Sql compiled:
CREATE TABLE tmp_pyodps_40755ebd_2d2a_482e_b360_3f3da0d5422c LIFECYCLE 1 AS
SELECT t1.`id`, t1.`name`, IF(t1.`f1` IS NULL, t1.`f2`, t1.`f1`) AS `f1`, IF(t1.`f2` IS NULL, t1.`f2`, t1.`f2`) AS `f2`, IF(t
1.`f3` IS NULL, t1.`f2`, t1.`f3`) AS `f3`, IF(t1.`f4` IS NULL, t1.`f2`, t1.`f4`) AS `f4`
FROM WB_BestPractice_dev.`pytable2` t1
Instance ID: 20191010071213970gbp66792
  id   name   f1   f2   f3   f4
0  0  name1  1.0  NaN  3.0  4.0
1  1  name1  2.0  NaN  NaN  1.0
2  2  name1  3.0  4.0  1.0  NaN
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0  NaN  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
Sql compiled:
CREATE TABLE tmp_pyodps_d39fcce1_d8a9_4cc2_8aff_2ed1e9c6bb1b LIFECYCLE 1 AS
SELECT pyodps_udf_1570691538_d9441c59_c666_4a5d_8154_67d8bc8c24ad(t1.`id`, t1.`name`, t1.`f1`, t1.`f2`, t1.`f3`, t1.`f4`) AS
(`id`, `name`, `f1`, `f2`, `f3`, `f4`)
FROM WB_BestPractice_dev.`pytable2` t1
Instance ID: 20191010071219627goqv9292
  id   name   f1   f2   f3   f4
0  0  name1  1.0  3.0  3.0  4.0
1  1  name1  2.0  1.0  1.0  1.0
2  2  name1  3.0  4.0  1.0  NaN
3  3  name1  1.0  1.0  2.0  3.0
```

```
4  4  name1  1.0  3.0  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
Sql compiled:
CREATE TABLE tmp_pyodps_3f190cf0_f9fb_4e06_a942_ab31c0241cd3 LIFECYCLE 1 AS
SELECT pyodps_udf_1570691566_0330848b_82d3_411c_88e1_cbbcc6adb9c1(t1.`id`, t1.`name`, t1.`f1`, t1.`f2`, t1.`f3`, t1.`f4`) AS
(`id`, `name`, `f1`, `f2`, `f3`, `f4`)
FROM WB_BestPractice_dev.`pytable2` t1
Instance ID: 20191010071247729gt776792
  id   name   f1   f2   f3   f4
0  0  name1  1.0  1.0  3.0  4.0
1  1  name1  2.0  2.0  2.0  1.0
2  2  name1  3.0  4.0  1.0  1.0
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0  1.0  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
```

5.10. PyODPS常见问题

本文为您介绍使用PyODPS时的常见问题。

问题类别	常见问题
安装PyODPS	• 安装PyODPS时，提示Warning: XXX not installed，如何解决？ • 安装PyODPS时，提示Project Not Found，如何解决？ • 安装PyODPS时，报错Syntax Error，如何解决？ • 在Mac上安装PyODPS时，报错Permission Denied，如何解决？ • 在Mac上安装PyODPS时，报错Operation Not Permitted，如何解决？
导入模块	• 执行from odps import ODPS时，报错No Module Named ODPS，如何解决？ • 执行from odps import ODPS时，报错Cannot Import Name ODPS，如何解决？ • 执行from odps import ODPS时，报错Cannot Import Module odps，如何解决？ • 在Ipython或Jupyter下使用PyODPS时，报错ImportError，如何解决？

问题类别	常见问题
使用PyODPS	• o.gettable('table_name').size中size字段的含义是什么？ • 如何设置Tunnel Endpoint？ • PyODPS如何使用包含cPython的第三方包？ • PyODPS中的DataFrame最多可以处理多少数据，对表的大小有限制吗？ • 在DataFrame中如何使用max_pt？ • 使用PyODPS向表写入数据的两种方式open_writer()和write_table()有什么区别？ • 为什么DataWorks PyODPS节点上查出的数据量要少于本地运行的结果？ • DataFrame如何获得Count实际数字？ • 使用PyODPS时，报错sourceIP is not in the white list，如何解决？ • 使用from odps import options options.sql.settings设置MaxCompute运行环境不成功，如何解决？ • 调用DataFrame的head方法时，报错IndexError: list index out of range，是什么原因？ • 上传Pandas DataFrame至MaxCompute时，报错ODPSError，如何解决？ • 通过DataFrame写表时，报错lifecycle is not specified in mandatory mode，如何解决？ • 使用PyODPS写数据时，提示Perhaps the datastream from server is crushed，如何解决？ • 使用PyODPS读数据时，报错Project is protected，如何解决？ • PyODPS脚本任务不定时出现连接失败，报错ConnectionError: timed out try catch exception，如何解决？ • 使用PyODPS运行get_sql_task_cost函数时，报错is not defined，如何解决？ • 使用PyODPS打印日志时，中文自动转化为编码显示，如何显示成原始中文？ • 设置options.tunnel.use_instance_tunnel = False，为什么字段在MaxCompute中定义为DATETIME类型，使用SELECT语句得到的数据为STRING类型？ • 如何利用Python语言特性来实现丰富的功能？ • 如何使用Pandas计算后端进行本地Debug？ • 如何避免嵌套循环执行慢的情况？ • 如何避免将数据下载到本地？ • 什么情况下可以下载PyODPS数据到本地处理？ • 通过open_reader最多只能取到1万条记录，如何获取多于1万条的记录？ • 为什么尽量使用内建算子，而不是自定义函数？ • 为什么通过DataFrame().schema.partitions获得分区表的分区值为空？ • 如何使用PyODPS DataFrame执行笛卡尔积？ • 如何使用PyODPS实现结巴中文分词？ • 如何使用PyODPS下载全量数据？ • 使用PyODPS统计表中某个字段的空值率时，是用execute_sql还是DataFrame？ • PyODPS数据类型如何设置？

安装PyODPS时，提示Warning: XXX not installed，如何解决？

产生此问题的原因为组件缺失，请根据报错信息中提示的XXX信息，明确缺失的组件名称，使用`pip`命令安装此组件。

安装PyODPS时，提示Project Not Found，如何解决？

产生此问题的原因为：

- Endpoint配置错误，需要修改为目标Project的Endpoint。更多Endpoint信息，请参见Endpoint。
- MaxCompute入口对象参数位置填写错误。请检查此项确保其填写正确。更多MaxCompute入口对象参数信息，请参见从平台到自行部署。

安装PyODPS时，报错Syntax Error，如何解决？

由于Python版本过低导致报错。不支持Python2.5及以下版本，建议使用PyODPS支持的主流版本，例如Python2.7.6+、Python3.3+以及Python2.6。

在Mac上安装PyODPS时，报错Permission Denied，如何解决？

您可以使用`sudo pip install pyodps`命令安装PyODPS。

在Mac上安装PyODPS时，报错Operation Not Permitted，如何解决？

此报错是由系统完整性保护导致。您需要重启设备，并在重启中按⌘+R键，此后在终端中运行如下命令可以解决此问题。

```
csrutil disable
reboot
```

更多信息，请参见Operation Not Permitted when on root - El Capitan (rootless disabled)。

执行from odps import ODPS时，报错No Module Named ODPS，如何解决？

此报错说明无法加载ODPS Package。无法加载的原因有如下几种：

- 原因一：安装了多个Python版本。

  解决措施：Search Path（通常是当前目录）中包含 `odps.py` 或 `init.py` 文件且名为*odps*的文件夹。解决方法如下：

  - 如果是文件夹重名，请修改文件夹名称。
  - 如果是曾经安装过一个名为*odps*的Python包，请使用 `pip uninstall odps` 进行删除。

- 原因二：同时安装了Python2和Python3版本。

  解决措施：确保设备只安装了Python2或Python3版本。

- 原因三：当前使用的Python下并未安装PyODPS。

  解决措施：安装PyODPS，安装方法请参见安装指南及使用限制。

执行from odps import ODPS时，报错Cannot Import Name ODPS，如何解决？

请检查当前工作路径下是否存在名为**odps.py**的文件。若存在，请改名后再执行导入操作。

执行from odps import ODPS时，报错Cannot Import Module odps，如何解决？

此报错通常是由于PyODPS遇到了依赖问题。请添加PyODPS技术支持钉钉群（11701793），联系钉群管理员定位解决。

在Ipython或Jupyter下使用PyODPS时，报错ImportError，如何解决？

在代码头部增加 `from odps import errors` 。

如果增加 `from odps import errors` 也报错则是缺少Ipython组件，执行 `pip install -U jupyter` 解决此问题。

o.gettable('table_name').size中size字段的含义是什么？

SIZE字段表示表的物理存储大小。

如何设置Tunnel Endpoint？

您可以通过 `options.tunnel.endpoint` 设置，请参见aliyun-odps-python-sdk。

PyODPS如何使用包含cPython的第三方包？

建议您打包成WHEEL格式后使用，请参见如何制作可以在MaxCompute上使用的crcmod。

PyODPS中的DataFrame最多可以处理多少数据，对表的大小有限制吗？

PyODPS对表的大小没有限制。本地Pandas创建DataFrame的大小受限于本地内存的大小。

在DataFrame中如何使用max_pt？

使用 `odps.df.func` 模块来调用MaxCompute内建函数。

```
from odps.df import func
df = o.get_table('your_table').to_df()
df[df.ds == func.max_pt('your_project.your_table')]  # ds是分区字段。
```

使用PyODPS向表写入数据的两种方式open_writer()和write_table()有什么区别？

每次调用 `write_table()` ，MaxCompute都会在服务端生成一个文件。这一操作需要较大的时间开销，同时过多的文件会降低后续的查询效率，还可能造成服务端内存不足。因此，建议在使用 `write_table()` 方法时，一次性写入多组数据或者传入一个Generator对象。

`open_writer()` 默认写入到Block中。

为什么DataWorks PyODPS节点上查出的数据量要少于本地运行的结果？

DataWorks上默认未开启Instance Tunnel，即 `instance.open_reader` 默认使用Result接口，最多可以获取一万条记录。

开启Instance Tunnel后，您可以通过 `reader.count` 获取记录数。如果您需要迭代获取全部数据，则需要通过设置 `options.tunnel.limit_instance_tunnel = False` 关闭Limit限制。

DataFrame如何获得Count实际数字？

1. 安装PyODPS后，在Python环境下执行如下命令创建MaxCompute表来初始化DataFrame。

   ```
   iris = DataFrame(o.get_table('pyodps_iris'))
   ```

2. 在DataFrame上执行Count获取DataFrame的总行数。

   ```
   iris.count()
   ```

3. 由于DataFrame上的操作并不会立即执行，只有当用户显式调用Execute方法或者立即执行的方法时，才会真正执行。此时为了防止Count方法延迟执行，可输入如下命令。

```
df.count().execute()
```

获取DataFrame实际数量的相关方法请参见聚合操作。详细的PyODPS方法延迟操作，请参见执行。

使用PyODPS时，报错sourceIP is not in the white list，如何解决？

PyODPS访问的MaxCompute项目存在白名单保护，请联系项目所有者将设备添加至IP白名单。更多IP白名单信息，请参见管理IP白名单。

使用from odps import options options.sql.settings设置MaxCompute运行环境不成功，如何解决？

- 问题现象

  使用PyODPS运行SQL，在申请MaxCompute实例前，通过如下代码设置MaxCompute运行环境。

  ```
  from odps import options
  options.sql.settings = {'odps.sql.mapper.split.size': 32}
  ```

  运行任务后只启动了6个Mapper，设置未生效。 在客户端执行 `set odps.stage.mapper.split.size=32` ，一分钟运行完毕。

- 产生原因

  客户端和PyODPS里设置的参数不一致。客户端的参数是 `odps.stage.mapper.split.size` ，而PyODPS里的参数是 `odps.sql.mapper.split.size` 。

- 解决措施

  修改参数为 `odps.stage.mapper.split.size` 。

调用DataFrame的head方法时，报错IndexError:listindexoutofrange，是什么原因？

由于 `list[index]` 没有元素或 `list[index]` 超出范围。

上传Pandas DataFrame至MaxCompute时，报错ODPSError，如何解决？

- 问题现象

  上传Pandas DataFrame至MaxCompute时，返回报错如下。

  ```
  ODPSError: ODPS entrance should be provided.
  ```

- 产生原因

  报错原因为没有找到全局的MaxCompute对象入口。

- 解决措施
  - 使用Room机制 `%enter` 时，会配置全局入口。
  - 对MaxCompute对象入口调用 `to_global` 方法。
  - 使用参数 `DataFrame(pd_df).persist('your_table', odps=odps)` 。

通过DataFrame写表时，报错lifecycle is not specified in mandatory mode，如何解决？

- 问题现象

  通过DataFrame写表时，返回报错如下。

  ```
  table lifecycle is not specified in mandatory mode
  ```

- 产生原因

  未给表设置生命周期。

- 解决措施

  Project要求对每张表设置生命周期，因此需要在每次执行时设置如下信息即可。

  ```
  from odps import options
  options.lifecycle = 7  # 此处设置lifecycle的值。lifecycle取值为整数，单位为天。
  ```

使用PyODPS写数据时，提示Perhaps the datastream from server is crushed，如何解决？

该报错是由脏数据导致，请您检查数据列数是否和目标表一致。

使用PyODPS读数据时，报错Project is protected，如何解决？

Project上的安全策略禁止读取表中的数据，如果想使用全部数据，可以使用以下方法：

- 联系Project Owner增加例外规则。
- 使用DataWorks或其他脱敏工具先对数据进行脱敏，导出到非保护Project，再进行读取。

如果只想查看部分数据，可使用如下方法：

- 改用 `o.execute_sql('select * from <table_name>').open_reader()` 。

- 改用 `DataFrame, o.get_table('<table_name>').to_df()` 。

PyODPS脚本任务不定时出现连接失败，报错ConnectionError: timed out try catch exception，如何解决？

产生此报错的可能原因如下：

- 建立连接超时。PyODPS默认的超时时间是5s，解决方法如下：
  - 您可以在代码头部加上如下代码，增加超时时间隔。

    ```
    workaround from odps import options
    options.connect_timeout=30
    ```

  - 捕获异常，进行重试。
- 由于沙箱限制，会造成部分机器禁止网络访问。建议您使用独享调度资源组执行任务，解决此问题。

使用PyODPS运行get_sql_task_cost函数时，报错is not defined，如何解决？

- 问题现象

  使用PyODPS运行get_sql_task_cost函数时，返回报错如下。

  ```
  NameError: name 'get_task_cost' is not defined.
  ```

- 解决措施

  函数名称有误。

- 解决措施

  使用execute_sql_cost替代get_sql_task_cost。

使用PyODPS打印日志时，中文自动转化为编码显示，如何显示成原始中文？

您可以使用类似 `print ("我叫 %s" % ('abc'))` 的输入方式解决。目前仅Python 2涉及此类问题。

设置options.tunnel.use_instance_tunnel = False，为什么字段在MaxCompute中定义为DATETIME类型，使用SELECT语句得到的数据为STRING类型？

在调用Open_Reader时，PyODPS会默认调用旧的Result接口。此时从服务端得到的数据是CSV格式的，所以DATETIME都是STRING类型。

打开Instance Tunnel，即设置 `options.tunnel.use_instance_tunnel = True` ，PyODPS会默认调用Instance Tunnel，即可解决此问题。

如何利用Python语言特性来实现丰富的功能？

- 编写Python函数。

  计算两点之间的距离有多种计算方法，例如欧氏距离、曼哈顿距离等，您可以定义一系列函数，在计算时根据具体情况调用相应的函数即可。

  ```
  def euclidean_distance(from_x, from_y, to_x, to_y):
      return ((from_x - to_x) ** 2 + (from_y - to_y) ** 2).sqrt()
  def manhattan_distance(center_x, center_y, x, y):
     return (from_x - to_x).abs() + (from_y - to_y).abs()
  ```

  调用如下。

```
In [42]: df
     from_x    from_y      to_x      to_y
0  0.393094  0.427736  0.463035  0.105007
1  0.629571  0.364047  0.972390  0.081533
2  0.460626  0.530383  0.443177  0.706774
3  0.647776  0.192169  0.244621  0.447979
4  0.846044  0.153819  0.873813  0.257627
5  0.702269  0.363977  0.440960  0.639756
6  0.596976  0.978124  0.669283  0.936233
7  0.376831  0.461660  0.707208  0.216863
8  0.632239  0.519418  0.881574  0.972641
9  0.071466  0.294414  0.012949  0.368514
In [43]: euclidean_distance(df.from_x, df.from_y, df.to_x, df.to_y).rename('distance')
   distance
0  0.330221
1  0.444229
2  0.177253
3  0.477465
4  0.107458
5  0.379916
6  0.083565
7  0.411187
8  0.517280
9  0.094420
In [44]: manhattan_distance(df.from_x, df.from_y, df.to_x, df.to_y).rename('distance')
   distance
0  0.392670
1  0.625334
2  0.193841
3  0.658966
4  0.131577
5  0.537088
6  0.114198
7  0.575175
8  0.702558
9  0.132617
```

- 利用Python语言的条件和循环语句。

  如果用户要计算的表保存在数据库，需要根据配置来对表的字段进行处理，然后对所有表进行UNION或者JOIN操作。这时如果用SQL实现是相当复杂的，但是用DataFrame处理则会非常简单。

  例如，您有30张表需要合成一张表，此时如果使用SQL，则需要对30张表执行UNION ALL操作。如果使用PyODPS，如下代码就可以完成。

```
table_names = ['table1', ..., 'tableN']
dfs = [o.get_table(tn).to_df() for tn in table_names]
reduce(lambda x, y: x.union(y), dfs)
## reduce语句等价于如下代码。
df = dfs[0]
for other_df in dfs[1:]:
    df = df.union(other_df)
```

如何使用Pandas计算后端进行本地Debug?

您可以通过如下两种方式来进行本地Debug。这两种方式除了初始化方法不同，后续代码完全一致：

- 通过Pandas DataFrame创建的PyODPS DataFrame可以使用Pandas执行本地计算。
- 使用MaxCompute表创建的DataFrame可以在MaxCompute上执行。

示例代码如下。

```
df = o.get_table('movielens_ratings').to_df()
DEBUG = True
if DEBUG:
    df = df[:100].to_pandas(wrap=True)
```

当所有后续代码都编写完成，本地的测试速度非常快。当测试结束后，您可以把Debug值改为False，这样后续就能在MaxCompute上执行全量的计算。

推荐您使用MaxCompute Studio来执行本地PyODPS程序调试。

如何避免嵌套循环执行慢的情况?

建议您通过Dict数据结构记录下循环的执行结果，最后在循环外统一导入到Dataframe对象中。如果您将Dataframe对象代码 `df=XXX` 放置在外层循环中，会导致每次循环计算都生成一个Dataframe对象，从而降低嵌套循环整体的执行速度。

如何避免将数据下载到本地？

请参见PyODPS节点实现避免将数据下载到本地。

什么情况下可以下载PyODPS数据到本地处理？

在如下两种情况下，可以下载PyODPS数据到本地：

- 数据量很小的情况下进行本地数据处理。
- 如果需要对单行数据应用一个Python函数，或者执行一行变多行的操作，这时使用PyODPS DataFrame就可以轻松完成，并且可以完全发挥MaxCompute的并行计算能力。

  例如有一份JSON串数据，需要把JSON串按Key-Value对展开成一行，代码如下所示。

```
In [12]: df
                json
0  {"a": 1, "b": 2}
1  {"c": 4, "b": 3}
In [14]: from odps.df import output
In [16]: @output(['k', 'v'], ['string', 'int'])
    ...: def h(row):
    ...:     import json
    ...:     for k, v in json.loads(row.json).items():
    ...:         yield k, v
    ...:
In [21]: df.apply(h, axis=1)
   k  v
0  a  1
1  b  2
2  c  4
3  b  3
```

通过open_reader最多只能取到1万条记录，如何获取多于1万条的记录？

使用 `create table as select ...` 将SQL的结果保存成表，再使用 `table.open_reader` 读取。

为什么尽量使用内建算子，而不是自定义函数？

计算过程中使用自定义函数比使用内建算子速度慢很多，因此建议使用内建算子。

对于百万行的数据，当一行应用了自定义函数后，执行时间从7秒延长到了27秒。如果有更大的数据集、更复杂的操作，时间的差距可能会更大。

为什么通过DataFrame().schema.partitions获得分区表的分区值为空？

这是因为DataFrame不区分分区字段和普通字段，所以获取分区表的分区字段作为普通字段处理。您可以通过如下方式过滤掉分区字段。

```
df = o.get_table().to_df()
df[df.ds == '']
```

建议您参照表来设置分区或读取分区信息。

如何使用PyODPS DataFrame执行笛卡尔积？

请参见PyODPS DataFrame处理笛卡尔积的方式。

如何使用PyODPS实现结巴中文分词？

请参见PyODPS节点实现结巴中文分词。

如何使用PyODPS下载全量数据？

PyODPS默认不限制从Instance读取的数据规模。但是对于受保护的MaxCompute项目，通过Tunnel下载数据将受限。此时，如果未设置 `options.tunnel.limit_instance_tunnel` ，则数据量限制会被自动打开，可下载的数据条数受到MaxCompute配置限制，通常该限制为10000条。如果您需要迭代获取全部数据，则需要关闭 `limit` 限制。您可以通过下列语句在全局范围内打开 `Instance Tunnel` 并关闭 `limit` 限制。

```
options.tunnel.use_instance_tunnel = True
options.tunnel.limit_instance_tunnel = False  # 关闭limit限制，读取全部数据。
with instance.open_reader() as reader:
    # 通过Instance Tunnel可读取全部数据。
```

使用PyODPS统计表中某个字段的空值率时，是用execute_sql还是DataFrame？

DataFrame聚合性能更高一些，推荐使用DataFrame来执行聚合操作。

PyODPS数据类型如何设置？

如果您使用PyODPS，可以通过下列方法打开新数据类型开关：

- 如果通过 `execute_sql` 方式打开新数据类型，可以执行 `o.execute_sql('set odps.sql.type.system.odps2=true;query_sql', hints={"odps.sql.submit.mode" : "script"})` 。
- 如果通过Dataframe打开新数据类型，例如persist、execute、to_pandas等方法，可通过hints参数设置。图示设置方法仅针对单个作业生效。

```
from odps.df import DataFrame
users - DataFrame(o.get_table('odps2_test'))
users.persist('copy_test',hints={'odps.sql.type.system.odps2':'true'})
```

如果通过Dataframe调用，且需要全局生效，需要使用Option参数 `options.sql.use_odps2_extension = True` 。

6.MapReduce

6.1. 概述

6.1.1. MapReduce

本文为您介绍MaxCompute支持的MapReduce编程接口及使用限制。

MaxCompute提供三个版本的MapReduce编程接口：

- MaxCompute MapReduce：MaxCompute的原生接口，执行速度快、开发快捷、不暴露文件系统。
- MaxCompute扩展MapReduce（MR2）：对MaxCompute MapReduce的扩展，支持更复杂的作业调度逻辑。MapReduce的实现方式与MaxCompute原生接口一致。
- Hadoop兼容版本MapReduce：高度兼容Hadoop MapReduce，与MaxCompute MapReduce MR2不兼容。

以上三个版本在基本概念、作业提交、输入与输出、资源使用等方面基本一致，仅各版本的Java SDK有所不同。本文仅对MapReduce的基本原理做简单介绍，更多详情请参见Hadoop Map/Reduce教程。

> **说明** 您无法通过MapReduce读写外部表中的数据。

应用场景

MapReduce支持下列场景：

- 搜索：网页爬取、倒排索引、PageRank。
- Web访问日志分析：
  - 分析和挖掘用户在Web上的访问、购物行为特征，实现个性化推荐。
  - 分析用户访问行为。
- 文本统计分析：
  - 热门小说的字数统计（WordCount）、词频TFIDF分析。
  - 学术论文、专利文献的引用分析和统计。
  - 维基百科数据分析。
- 海量数据挖掘：非结构化数据、时空数据和图像数据挖掘。
- 机器学习：监督学习、无监督学习和分类算法（例如决策树、SVM）。
- 自然语言处理：
  - 基于大数据的训练和预测。
  - 基于语料库构建单词同现矩阵，频繁项集数据挖掘、重复文档检测等。
- 广告推荐：用户单击（CTR）和购买行为（CVR）预测。

MapReduce流程说明

MapReduce处理数据过程主要分成Map和Reduce两个阶段。首先执行Map阶段，再执行Reduce阶段。Map和Reduce的处理逻辑由用户自定义实现，但要符合MapReduce框架的约定。MapReduce处理数据的完整流程如下：

1. 输入数据：对文本进行分片，将每片内的数据作为单个Map Worker的输入。分片完毕后，多个Map Worker便可以同时工作。

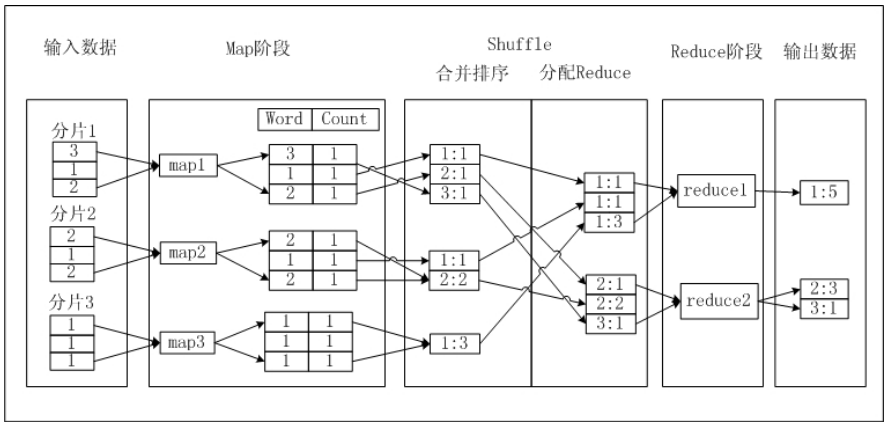
   在正式执行Map前，需要将输入数据进行分片。所谓分片，就是将输入数据切分为大小相等的数据块，每一块作为单个Map Worker的输入被处理，以便于多个Map Worker同时工作。

2. Map阶段：每个Map Worker在读入各自的数据后，进行计算处理，最终输出给Reduce。Map Worker在输出数据时，需要为每一条输出数据指定一个Key，这个Key值决定了这条数据将会被发送给哪一个Reduce Worker。Key值和Reduce Worker是多对一的关系，具有相同Key的数据会被发送给同一个Reduce Worker，单个Reduce Worker有可能会接收到多个Key值的数据。
3. 在进入Reduce阶段之前，MapReduce框架会对数据按照Key值排序，使得具有相同Key的数据彼此相邻。如果您指定了**合并操作（Combiner）**，框架会调用Combiner，将具有相同Key的数据进行聚合。Combiner的逻辑可以由您自定义实现。与经典的MapReduce框架协议不同，在MaxCompute中，Combiner的输入、输出的参数必须与Reduce保持一致，这部分的处理通常也叫做**洗牌（Shuffle）**。
4. Reduce阶段：进入Reduce阶段，相同Key的数据会传送至同一个Reduce Worker。同一个Reduce Worker会接收来自多个Map Worker的数据。每个Reduce Worker会对Key相同的多个数据进行Reduce操作。最后，一个Key的多条数据经过Reduce的作用后，将变成一个值。
5. 输出结果数据。

> **说明** 上文仅是对MapReduce框架的简单介绍，更多详情请查阅功能介绍。

下文将以WordCount为例，为您介绍MaxCompute MapReduce各个阶段的概念。

假设存在一个文本a.txt，文本内每行是一个数字，您要统计每个数字出现的次数。文本内的数字称为Word，数字出现的次数称为Count。如果MaxCompute MapReduce完成这一功能，需要经历以下流程，图示如下。



操作步骤

1. 输入数据：对文本进行分片，将每片内的数据作为单个Map Worker的输入。
2. Map阶段：Map处理输入，每获取一个数字，将数字的Count设置为1，并将此<Word, Count>对输出，此时以Word作为输出数据的Key。
3. Shuffle>合并排序：在Shuffle阶段前期，首先对每个Map Worker的输出，按照Key值（即Word值）进行排序。排序后进行Combiner操作，即将Key值（Word值）相同的Count累加，构成一个新的<Word, Count>对。此过程被称为合并排序。
4. Shuffle>分配Reduce：在Shuffle阶段后期，数据被发送到Reduce端。Reduce Worker收到数据后依赖Key值再次对数据排序。
5. Reduce阶段：每个Reduce Worker对数据进行处理时，采用与Combiner相同的逻辑，将Key值（Word值）相同的Count累加，得到输出结果。
6. 输出结果数据。

> **说明** 由于MaxCompute的所有数据都被存放在表中，因此MaxCompute MapReduce的输入、输出只能是表，不允许您自定义输出格式，不提供类似文件系统的接口。

使用限制

- MapReduce使用限制请参见使用限制汇总。
- 有关本地运行的MapReduce使用限制，请参见本地运行和分布式环境运行差异。
- 按量计费开发者版资源仅支持MaxCompute SQL（支持使用UDF）、PyODPS作业任务，暂不支持MapReduce、Spark等其他任务。

6.1.2. 扩展MapReduce（MR2）

相比于传统的MapReudce，MaxCompute提供的扩展MapReduce模型（简称MR2）改变了底层的调度和IO模型，可避免作业时冗余的IO操作。

与MaxCompute相比，MR2在Map、Reduce等函数编写方式上基本一致，较大的不同点发生在执行作业时，示例请参见Pipeline示例。

MR2模型产生背景

传统的MapReduce模型要求在经过每一轮MapReduce操作后，得到的数据结果必须存储到分布式文件系统中（例如，HDFS或MaxCompute数据表）。MapReduce模型通常由多个MapReduce作业组成，每个作业执行完成后都需要将数据写入磁盘，然而后续的Map任务很可能只需要读取一遍这些数据，为之后的Shuffle阶段做准备，这种情况就产生了冗余的磁盘IO操作。

MaxCompute的计算调度逻辑可以支持更复杂的编程模型， 针对上述情况，可以在Reduce后直接执行下一次的Reduce操作，而不需要中间插入一个Map操作。因此，MaxCompute提供了扩展的MapReduce模型，即可以支持Map后连接任意多个Reduce操作，例如Map>Reduce>Reduce。

与Hadoop Chain Mapper/Reducer对比

Hadoop Chain Mapper/Reducer也支持类似的串行化Map或Reduce操作，但和MaxCompute的扩展MapReduce（MR2）模型有本质的区别。

Chain Mapper/Reducer基于传统的MapReduce模型，仅可以在原有的Mapper或Reducer后增加一个或多个Mapper操作（不允许增加Reducer）。这样的优点是，您可以复用之前的Mapper业务逻辑，把一个Map或Reduce拆成多个Mapper阶段，同时本质上并没有改变底层的调度和I/O模型。

6.1.3. Hadoop兼容版本MapReduce

本文为您介绍开源兼容MapReduce的应用背景，以及Hadoop MapReduce插件的基本使用方式。

产生背景

MaxCompute有一套原生的MapReduce编程模型和接口，简单来说，这套接口的输入输出都是MaxCompute中的表，处理的数据以Record为组织形式，它可以很好地描述表中的数据处理过程。

但是与社区的Hadoop相比，二者的编程接口差异较大。Hadoop用户如果要将原来的Hadoop MapReduce作业迁移到MaxCompute的MapReduce中执行，需要重写MapReduce的代码，使用MaxCompute的接口进行编译和调试，运行正常后再打成一个JAR包，才能放到MaxCompute平台中运行。这个过程十分繁琐，需要耗费很多的开发和测试人力。如果能够完全不改或者少量地修改原来的Hadoop MapReduce代码，便可以在MaxCompute平台上运行，将会比较理想。

现在MaxCompute平台提供了一个Hadoop MapReduce到MaxCompute MapReduce的适配工具，已经在一定程度上实现了Hadoop MapReduce作业的二进制级别的兼容，您可以在不改代码的情况下通过指定一些配置，便可将原来在Hadoop上运行的MapReduce JAR包直接运行在MaxCompute上。目前该工具处于测试阶段，暂时还不能支持自定义Comparator和自定义Key类型。

下文以WordCount程序为例，为您介绍HadoopMapReduce插件的基本使用方式。

> **说明**
> - 更多开源兼容性的介绍请参见开源版本SDK兼容性。
> - 更多Hadoop MapReduce SDK的介绍请参见MapReduce官方文档。

下载HadoopMapReduce插件

请下载Hadoop MapReduce，包名为*openmr_hadoop2openmr-1.0.jar*。

> **说明** 此JAR包中已经包含hadoop-2.7.2版本的相关依赖，在作业的JAR包中请不要携带Hadoop的依赖，避免版本冲突。

准备WordCount的JAR包

编译导出WordCount的JAR包（wordcount_test.jar），WordCount程序的源码如下所示：

```
package com.aliyun.odps.mapred.example.hadoop;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.util.StringTokenizer;
public class WordCount {
    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
        ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

准备测试数据

1. 执行如下语句创建输入表wc_in和输出表wc_out。

   ```
   create table if not exists wc_in(line string);
   create table if not exists wc_out(key string, cnt bigint);
   ```

2. 通过Tunnel命令将示例数据导入输入表wc_in中。

   需要导入文本文件data.txt的数据内容如下。

   ```
   hello maxcompute
   hello mapreduce
   ```

   您可通过MaxCompute客户端的 `Tunnel` 命令将data.txt的数据导入wc_in中，如下所示。

   ```
   tunnel upload data.txt wc_in;
   ```

准备表与HDFS文件路径的映射关系配置

在配置文件*wordcount-table-res.conf*中配置表与HDFS文件路径的映射关系。配置文件如下所示。

```
{
  "file:/foo": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver",
      "properties": {
          "text.resolver.columns.combine.enable": "true",
          "text.resolver.seperator": "\t"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_in",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "exact"
  },
  "file:/bar": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.BinaryFileResolver",
      "properties": {
          "binary.resolver.input.key.class" : "org.apache.hadoop.io.Text",
          "binary.resolver.input.value.class" : "org.apache.hadoop.io.LongWritable"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_out",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "fuzzy"
  }
}
```

配置项说明：

整个配置是一个JSON文件，描述HDFS上的文件与MaxCompute上的表之间的映射关系，一般要配置输入和输出两部分，一个HDFS路径对应一个resolver配置、tableInfos配置以及matchMode配置。

- resolver：用于配置如何对待文件中的数据，目前有com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver和com.aliyun.odps.mapred.hadoop2openmr.resolver.BinaryFileResolver两个内置的resolver可以选用。除了指定好resolver的名字，还需要为相应的resolver配置一些特性指导它正确的进行数据解析。
  - TextFileResolver：对于纯文本的数据，输入输出都会当成纯文本对待。当作为输入resolver配置时，需要配置的properties有text.resolver.columns.combine.enable和text.resolver.seperator。当text.resolver.columns.combine.enable配置为true时，会把输入表的所有列按照text.resolver.seperator指定的分隔符组合成一个字符串作为输入。否则，会把输入表的前两列分别作为key、value。
  - BinaryFileResolver：可以处理二进制的数据，自动将数据转换为MaxCompute可以支持的数据类型，如：Bigint、Bool、Double等。当作为输出resolver配置时，需要配置的特性有binary.resolver.input.key.class和binary.resolver.input.value.class，分别代表中间结果的key和value类型。
- tableInfos：HDFS对应的MaxCompute表，目前只支持配置表的名称tblName，而partSpec和label请保持和示例一致。
- matchMode：路径的匹配模式，可选项为exact和fuzzy，分别代表精确匹配和模糊匹配，如果设置为fuzzy，则可以通过正则来匹配HDFS的输入路径。

作业提交

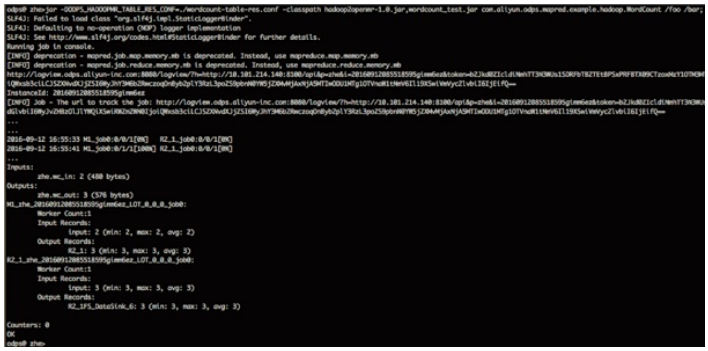
在MaxCompute客户端运行如下命令提交作业。MaxCompute客户端安装和配置方法请参见客户端。

```
jar -DODPS_HADOOPMR_TABLE_RES_CONF=./wordcount-table-res.conf -classpath hadoop2openmr-1.0.jar,wordcount_test.jar com.aliyun.odps
.mapred.example.hadoop.WordCount /foo/bar;
```

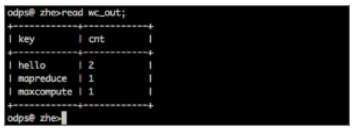
说明

- wordcount-table-res.conf是配置了/foo/bar的映射。
- wordcount_test.jar包是您的Hadoop MapReduce的JAR包。
- com.aliyun.odps.mapred.example.hadoop.WordCount是您要运行的作业类名。
- /foo/bar是指在HDFS上的路径，在JSON配置文件中映射成了wc_in和wc_out。
- 配置好映射后，您需要通过datax或DataWorks数据集成功能，手动导入Hadoop HDFS的输入文件到wc_in进行MR计算，并手动导出结果数据wc_out到您的HDFS输出目录/bar。
- 此处假设已经将hadoop2openmr-1.0.jar、wordcount_test.jar和wordcount-table-res.conf放置到odpscmd的当前目录，否则在指定配置和-classpath的路径时需要做相应的修改。

运行过程如下图所示：



当作业运行完成后，便可查看结果表wc_out的内容，验证作业是否成功结束，结果是否符合预期。



6.2. 使用限制

本文对MapReduce使用限制项进行说明，不遵循使用限制可能导致业务受到影响。

MaxCompute MapReduce限制项汇总，如下表所示。

边界名	边界值	分类	配置项名称	默认值	是否可配置	说明
Instance内存占用	[256 MB,12 GB]	内存限制	`odps.stage.mapper(reducer).mem` 和 `odps.stage.mapper(reducer).jvm.mem`	2048 MB + 1024 MB	是	单个Map Instance或Reduce Instance占用Memory，有框架Memory（默认2048 MB）和JVM的Heap Memory（默认1024 MB）两部分。
Resource数量	256个	数量限制	-	无	否	单个Job引用的Resource数量不超过256个，Table、Archive按照一个单位计算。
输入路数和输出路数	1024个和256个	数量限制	-	无	否	单个Job的输入路数不能超过1024（同一个表的一个分区算一路输入，总的不同表个数不能超过64个），单个Job的输出路数不能超过256。
Counter数量	64个	数量限制	-	无	否	单个Job中自定义Counter的数量不能超过64，Counter的Group Name和Counter Name中不能带有井号（#），两者长度和不能超过100。
Map Instance	[1,100000]	数量限制	odps.stage.mapper.num	无	是	单个Job的Map Instance个数由框架根据Split Size计算得出，如果没有输入表，可以通过odps.stage.mapper.num直接设置，最终个数范围[1,100000]。
Reduce Instance	[0,2000]	数量限制	odps.stage.reducer.num	无	是	单个Job默认Reduce Instance个数为Map Instance个数的1/4，用户设置作为最终的Reduce Instance个数，范围[0,2000]。可能出现这样的情形：Reduce处理的数据量会比Map大很多倍，导致Reduce阶段比较慢，而Reduce只能最多2000。

边界名	边界值	分类	配置项名称	默认值	是否可配置	说明
重试次数	3	数量限制	-	无	否	单个Map Instance或Reduce Instance失败重试次数为3，一些不可重试的异常会直接导致作业失败。
Local Debug模式	Instance个数不超100	数量限制	-	无	否	Local Debug模式下： • 默认Map Instance个数为2，不能超过100。 • 默认Reduce Instance个数为1，不能超过100。 • 默认一路输入下载记录数100，不能超过10000。
重复读取Resource次数	64次	数量限制	-	无	否	单个Map Instance或Reduce Instance重复读一个Resource次数限制<=64次。
Resource字节数	2 GB	长度限制	-	无	否	单个Job引用的Resource总计字节数大小不超过2 GB。
Split Size	大于等于1	长度限制	odps.stage.mapper.split.size	256 MB	是	框架会参考设置的Split Size值来划分Map，决定Map的个数。
STRING列内容长度	8 MB	长度限制	-	无	否	MaxCompute表STRING列内容长度不允许超出限制。
Worker运行超时时间	[1,3600]	时间限制	odps.function.timeout	600	是	Map或者Reduce Worker在无数据读写且没有通过 `context.progress()` 主动发送心跳的情况下的超时时间，默认值是600s。
MapReduce引用Table资源支持的字段类型	BIGINT、DOUBLE、STRING、DATETIME、BOOLEAN	数据类型限制	-	无	否	MapReduce任务引用表资源时，若表字段有其他类型字段执行报错。
MapReduce是否支持读取OSS数据	-	功能限制	-	无	否	MapReduce不支持读取OSS数据。
MapReduce是否支持MaxCompute 2.0新类型	-	功能限制	-	无	否	MapReduce不支持MaxCompute 2.0新类型。

6.3. 快速入门

本文以开发WordCount MapReduce作业为例，为您介绍如何通过MaxCompute Studio编写MapReduce程序及生成JAR包，并在MaxCompute客户端上运行MapReduce作业。

前提条件

请确认您已满足如下条件：

- 已安装并配置MaxCompute客户端。

  更多安装并配置MaxCompute客户端操作，请参见安装并配置MaxCompute客户端。
- 已安装MaxCompute Studio并连接MaxCompute项目。

  更多安装MaxCompute Studio及连接MaxCompute项目操作，请参见安装MaxCompute Studio和管理项目连接。
- 请确保您购买的产品规格非按量计费开发者版。

  按量计费开发者版不支持处理MapReduce作业。您可以登录MaxCompute控制台，在**服务资源**页签，查看当前已购买的规格类型。
- 已准备好源数据文件并保存至本地。

  本文的示例数据文件名称为data.txt，内容为 `hello,odps` ，保存位置为MaxCompute客户端的 `bin` 目录。

注意事项

如果您使用Maven开发MapReduce程序，可以从Maven库中搜索odps-sdk-mapred、odps-sdk-commons和odps-sdk-core获取不同版本的Java SDK，pom.xml文件需要配置的依赖信息如下。

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-mapred</artifactId>
    <version>0.36.4-public</version>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-commons</artifactId>
    <version>0.36.4-public</version>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-core</artifactId>
    <version>0.36.4-public</version>
</dependency>
```

操作流程

1. 步骤一：开发MapReduce程序

   通过MaxCompute Studio编写、运行调试MapReduce程序。

2. 步骤二：生成并上传MapReduce JAR包

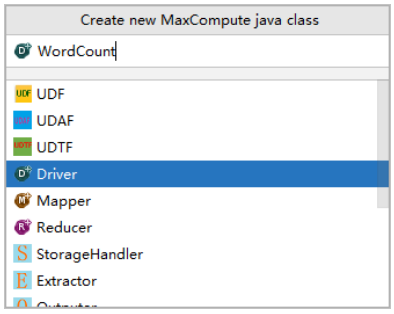
   将编译成功的WordCount.java脚本打包为JAR包并上传至MaxCompute项目中。

3. 步骤三：运行MapReduce作业

   基于上传至MaxCompute项目的JAR包，使用 `jar` 命令运行MapReduce作业。

步骤一：开发MapReduce程序

1. 创建MaxCompute Java Module。
   i. 启动IntelliJ IDEA，在顶部菜单栏，单击**File > New > Module...**。
   ii. 在**New Module**对话框的左侧导航栏，单击**MaxCompute Java**。
   iii. 配置**Module SDK**，单击**Next**。
   iv. 填写**Module name**，例如mapreduce，单击**Finish**。
2. 编写并运行调试WordCount MapReduce程序。
   i. 在**Project**区域，右键单击Module的源码目录**src > main > java**，选择**new > MaxCompute Java**。
   ii. 在**Create new MaxCompute java class**对话框，单击**Driver**，并填写**Name**，例如WordCount，按Enter键。

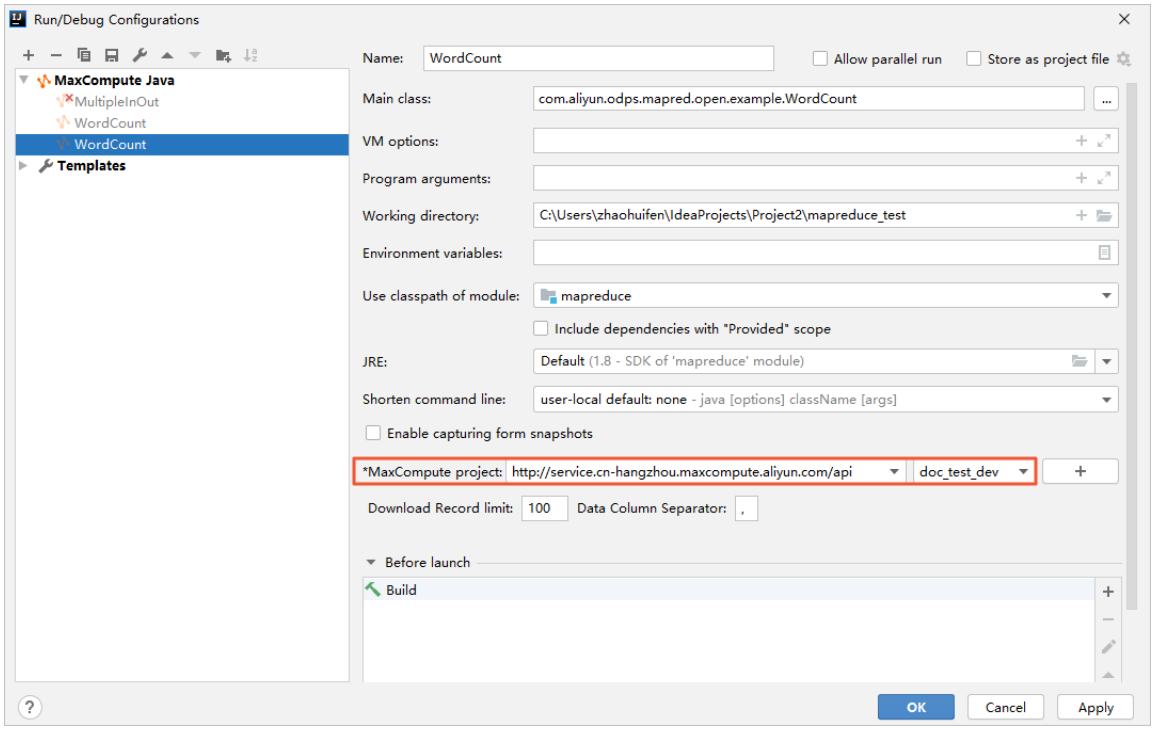
   

   iii. 在新建的WordCount.java脚本界面，编写WordCount MapReduce程序，实现统计单词个数功能。

   WordCount完整代码示例，请参见代码示例。

   iv. 在左侧导航栏的WordCount.java脚本上，单击右键，选择**Run**。

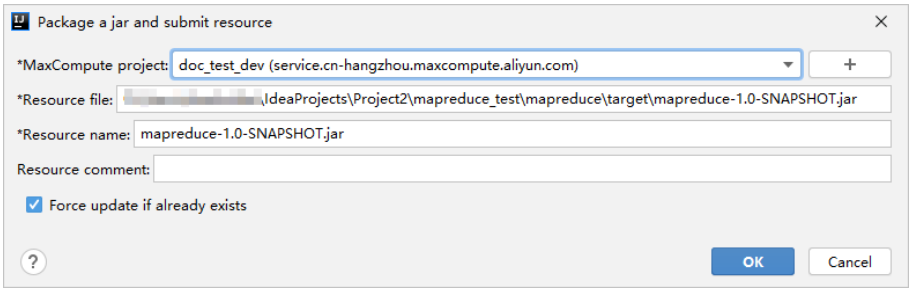
v. 在**Run/Debug Configurations**对话框，配置**MaxCompute project**为目标MaxCompute项目。



vi. 单击**OK**，运行并调试WordCount.java脚本，确保脚本编译成功。

步骤二：生成并上传MapReduce JAR包

1. 在IntelliJ IDEA左侧导航栏的WordCount.java脚本上，单击右键，选择**Deploy to server...**。
2. 在**Package a jar and submit resource**对话框，配置打包参数并单击**OK**，完成打包及上传操作。



详细参数说明，请参见操作步骤。

> **说明** 如果您使用Maven开发MapReduce程序，打包为JAR包后，需要通过MaxCompute客户端手动将JAR包上传至MaxCompute项目。更多上传JAR包操作，请参见添加资源。命令示例如下。

```
add jar mapreduce-1.0-SNAPSHOT.jar;
```

步骤三：运行MapReduce作业

1. 登录MaxCompute客户端或在MaxCompute Studio中打开MaxCompute客户端。

   MaxCompute Studio中集成了MaxCompute客户端，您可以在MaxCompute Studio中直接运行MaxCompute客户端。更多信息，请参见集成MaxCompute客户端。

2. 创建输入表和输出表。

   输入表中的数据为MapReduce作业的源数据，输出表中存储MapReduce作业对输入表中的数据进行处理后的结果。命令示例如下。

```
--创建输入表wc_in。
create table wc_in (key STRING, value STRING);
--创建输出表wc_out。
create table wc_out (key STRING, cnt BIGINT);
```

   更多建表语法信息，请参见创建表。

3. 使用Tunnel Upload命令向表wc_in中插入数据。

命令示例如下。

```
tunnel upload data.txt wc_in;
```

更多Tunnel信息，请参见Tunnel。

4. 使用 `jar` 命令调用生成的JAR包，运行MapReduce作业。

   命令示例如下。

   ```
   jar -resources mapreduce-1.0-SNAPSHOT.jar -classpath mapreduce-1.0-SNAPSHOT.jar com.aliyun.odps.mapred.open.example.WordCount wc_in wc_out;
   ```

   - `-resources mapreduce-1.0-SNAPSHOT.jar` ： `-resources` 指定MapReduce作业调用的资源名称，即步骤二中上传的JAR包mapreduce-1.0-SNAPSHOT.jar。
   - `-classpath mapreduce-1.0-SNAPSHOT.jar` ： `-classpath` 指定MainClass所在的JAR包路径。
   - `com.aliyun.odps.mapred.open.example.WordCount` ：MapReduce程序中定义的MainClass。
   - `wc_in wc_out` ：输入表和输出表。

   更多 `jar` 命令信息，请参见语法介绍。

5. 执行如下命令查看wc_out表的写入结果。

   ```
   select * from wc_out;
   ```

   返回结果如下。

   ```
   +------------+------------+
   | key        | cnt        |
   +------------+------------+
   | hello      | 1          |
   | odps       | 1          |
   +------------+------------+
   ```

6.4. 使用说明

6.4.1. 基本概念

本章节为您介绍MapReduce的基本概念。

Map/Reduce

Map和Reduce分别支持对应的map/reduce方法、setup方法及cleanup方法。setup方法在map/reduce方法之前调用，每个Worker调用且仅调用一次。

cleanup方法在map/reduce方法之后调用，每个Worker调用且仅调用一次。

相关的使用示例请参见示例程序。

排序

支持将Map输出的key record中的某几列作为排序（Sort）列，不支持您自定义的比较器（comparator）。您可以在排序列中选择某几列作为Group列，不支持您自定义的Group比较器。Sort列一般用来对您的数据进行排序，而Group列一般用来进行二次排序。

相关的使用示例请参见二次排序源代码。

哈希

支持设置哈希（partition）列及用户自定义哈希函数（partitioner）。哈希列的使用优先级高于自定义哈希函数。

哈希函数用于将Map端的输出数据按照哈希逻辑分配到不同的Reduce Worker上。

归并

归并（Combiner）函数将Shuffle阶段相邻的Record进行归并。您可以根据不同的业务逻辑选择是否使用归并函数。

归并函数是MapReduce计算框架的一种优化，通常情况下，Combiner的逻辑与Reduce相同。当Map输出数据后，框架会在Map端对相同key值的数据进行本地的归并操作。

相关的使用示例请参见示例程序。

6.4.2. MapReduce作业提交

本文向您介绍如何使用Jar命令在MaxCompute客户端运行和提交MapReduce作业。

MaxCompute客户端提供Jar命令用于运行MapReduce作业，举例如下。

```
jar -conf \home\admin\myconf -resources a.txt,example.jar -classpath ..\lib\example.jar:.\other_lib.jar -D java.library.path=.\native;
```

语法介绍

```
jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS];
        -conf <configuration_file>         Specify an application configuration file
        -resources <resource_name_list>    file\table resources used in mapper or reducer, seperate by comma
        -classpath <local_file_list>       classpaths used to run mainClass
        -D <name>=<value>                  Property value pair, which will be used to run mainClass
        -l                                 Run job in local mode
```

参数说明

`<GENERIC_OPTIONS>` 包括（均为可选参数）：

- -conf <configuration file>：指定JobConf配置文件。该文件可以影响SDK中JobConf的设置。

  JobConf配置文件的模板如下。

  ```
  <configuration>
        <property>
           <name>import.filename</name>
           <value>resource.txt</value>
        </property>
     </configuration>
  ```

  在上述模板中，通过JobConf配置文件定义一个名为import.filename的变量，该变量的值为resource.txt。

  您可以在MapReduce程序中通过JobConf接口获取该变量的值。通过SDK中JobConf接口您可以达到相同的目的，详情请参见资源使用示例 。

  示例如下。

  ```
  add jar data\mapreduce-examples.jar;
      jar -resources mapreduce-examples.jar -classpath data\mapreduce-examples.jar
          org.alidata.odps.mr.examples.WordCount wc_in wc_out;
      add file data\src.txt;
      add jar data\mapreduce-examples.jar;
      jar -resources src.txt,mapreduce-examples.jar -classpath data\mapreduce-examples.jar
          org.alidata.odps.mr.examples.WordCount wc_in wc_out;
      add file data\a.txt;
      add table wc_in as test_table;
      add jar data\work.jar;
      jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
          -classpath data\work.jar:otherlib.jar
          -D import.filename=resource.txt org.alidata.odps.mr.examples.WordCount args;
  ```

- -resources <resource_name_list>：MapReduce作业运行时使用的资源声明。一般情况下，resource_name_list中需要指定Map/Reduce函数所用的资源名称。

  > **说明**
  > - 如果在Map/Reduce函数中读取了其他MaxCompute资源，则这些资源名称也需要被添加到resource_name_list中。
  > - 资源之间使用逗号分隔，使用跨项目空间使用资源时，需要前面加上 `PROJECT/resources/` 。例如， `-resources otherproject/resources/resfile` 。
  > - 在Map/Reduce函数中读取资源的示例，请参见资源使用示例。

- -classpath <local_file_list>：本地执行时的classpath，主要用于指定main函数所在的Jar包的本地路径（包含相对路径和绝对路径）。

  包名之间使用系统默认的文件分割符作分割。通常情况下，Windows系统中使用分号（;），Linux系统中使用逗号（,）。如果您在云端服务器运行MapReduce任务，则使用逗号（,）进行分隔。

  > **说明** 通常，您可能更习惯于将main函数与Map/Reduce函数编写在一个包中，例如WordCount 代码示例。因此，在执行示例程序时，-resources及-classpath的参数中都出现了mapreduce-examples.jar。但二者意义不同，-resources引用的是Map/Reduce函数，运行于分布式环境中。而-classpath引用的是main函数，运行于本地，指定的Jar包路径也是本地文件路径。

- -D <prop_name>=<prop_value>：本地执行时，<mainClass>的Java属性，可以定义多个。
- -l：以本地模式执行MapReduce作业，主要用于程序调试。

6.4.3. 输入与输出

本文向您介绍MapReduce的输入与输出。

- MaxCompute MapReduce的输入、输出，支持MaxCompute内置类型的Bigint、Double、String、Datetime和Boolean类型，不支持您自定义数据类型。
- 接受多表输入，且输入表的Schema可以不同。在Map函数中，您可以获取当前Record对应的Table信息。
- 输入可以为空，不支持视图（View）作为输入。

- Reduce接受多路输出，可以输出到不同表，或者同一张表的不同分区。不同输出的Schema可以不同。不同输出间通过label进行区分，默认输出不必加label，但目前不接受没有输出的情况。

有关输入输出的使用示例请参见多路输入输出示例。

6.4.4. 资源使用

您可以在Map/Reduce中读取MaxCompute资源，Map/Reduce的任意Worker都会将资源加载到内存中，以供您的代码使用。

相关的使用示例请参见使用资源示例。

6.4.5. 本地运行

本文向您介绍本地和分布式环境运行MapReduce的差异和MapReduce本地运行示例。

各阶段介绍

本地运行前：通过在Jar命令中设置-local参数，在本地模拟MapReduce的运行过程，从而进行本地调试。

本地运行时：客户端会从MaxCompute中下载本地调试所需要的输入表的元信息、数据，所需要的资源以及输出表的元信息，并将这些信息保存到一个名为 `warehouse` 的本地目录中。

本地运行结束后：程序运行结束后，会将计算结果输出到 `warehouse` 目录内的一个文件中。如果本地的 `warehouse` 目录下已经下载了输入表及被引用的资源，在下一次运行时，会直接引用 `warehouse` 下的数据及文件，无需重复下载。

本地运行和分布式环境运行差异

在本地运行的过程中，仍然会启动多个Map及Reduce进程处理数据，但这些进程不是并发运行，而是依次串行运行。

此外，这个本地模拟运行过程与真正的分布式运行有如下差别：

- 输入表行数限制：目前最多只支持下载100行数据。
- 资源的使用：在分布式环境中，MaxCompute会限制引用资源的大小，详情请参见MR限制汇总。但在本地运行环境中，不会有资源大小的限制。
- 安全限制：MaxCompute MapReduce及UDF程序在分布式环境中运行时受到Java沙箱的限制。但在本地运行时，没有此限制。

本地运行示例

本地运行的示例如下。

```
odps:my_project> jar -l com.aliyun.odps.mapred.example.WordCount wc_in wc_out
Summary:
counters: 10
    map-reduce framework
            combine_input_groups=2
            combine_output_records=2
            map_input_bytes=4
            map_input_records=1
            map_output_records=2
            map_output_[wc_out]_bytes=0
            map_output_[wc_out]_records=0
            reduce_input_groups=2
            reduce_output_[wc_out]_bytes=8
            reduce_output_[wc_out]_records=2
OK
```

关于WordCount示例的代码介绍请参见WordCount示例。

如果您是第一次运行本地调试命令，命令成功结束后，会在当前路径下看到一个名为*warehouse*的路径。 `warehouse` 的目录结构如下所示。

```
<warehouse>
   |____my_project(项目空间目录)
         |____ <__tables__>
         |       |__wc_in(表数据目录)
         |       |       |____ data(文件)
         |       |       |
         |       |       |____ <__schema__> (文件)
         |       |__wc_out (表数据目录)
         |               |____ data(文件)
         |               |
         |               |____ <__schema__> (文件)
         |
         |____ <__resources__>
                 |
                 |___table_resource_name (表资源)
                 |         |____<__ref__>
                 |
                 |___ file_resource_name (文件资源)
```

- `my_project` 的同级目录表示项目空间。`wc_in` 及 `wc_out` 表示数据表，您在Jar命令中读写的表文件数据会被下载到这级目录下。
- *<__schema__>*文件中的内容表示表的元信息，其文件格式定义如下。

  ```
  project=local_project_name
  table=local_table_name
  columns=col1_name:col1_type,col2_name:col2_type
  partitions=p1:STRING,p2:BIGINT    --本示例中不需要此字。
  ```

  其中，列名与列类型使用英文冒号分隔，列与列之间使用英文逗号分隔。*<__schema__>*文件的最前面需要声明Project名字及Table名字，即 `project_name.table_name` ，并通过英文逗号与列的定义做分隔，即 `project_name.table_name,col1_name:col1_type,col2_name:col2_type,……` 。
- `tables` 目录中*data*文件表示表的数据。列的数量及数据必须与*_schema_*文件的定义相符，不能多列或者少列，列之间使用逗号分隔。

  `wc_in` 的*_schema_*文件内容如下。

  ```
  my_project.wc_in,key:STRING,value:STRING
  ```

  *data*文件内容如下。

  ```
  0,2
  ```

  客户端会从MaxCompute中下载表的元信息及部分数据内容，并保存到上述两个文件中。如果再次运行此示例，将直接使用 `wc_in` 目录下的数据，不会再次下载。

  > **说明** 仅在MapReduce的本地运行模式下支持从MaxCompute中下载数据的功能。

  `wc_out` 的*_schema_*文件内容如下。

  ```
  my_project.wc_out,key:STRING,cnt:BIGINT
  ```

  *data*文件内容如下。

  ```
  0,1
  2,1
  ```

  客户端会从MaxCompute中下载 `wc_out` 表的元信息，并保存到*_schema_*文件中。而本地运行后，生成的结果数据，则保存到*data*文件中。

  > **说明**
  > - 您也可以自行编辑*_schema_*及*data*文件，而后将这两个文件放置在对应的表目录下。
  > - 在本地运行时，客户端检测到表目录已经存在，则不会从MaxCompute中下载这个表的信息。本地的表目录可以是MaxCompute中不存在的表。

6.5. 示例程序

6.5.1. WordCount示例

本文为您介绍MapReduce WordCount示例程序。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。

i. 创建测试表。

```
create table wc_in (key string, value string);
create table wc_out(key string, cnt bigint);
```

ii. 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

2. 准备好WordCount测试表和资源。
3. 使用Tunnel导入数据。

```
tunnel upload data wc_in;
```

导入wc_in表的数据如下。

```
hello,odps
```

测试步骤

在MaxCompute客户端中执行WordCount。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.WordCount wc_in wc_out
```

预期结果

作业成功结束后，输出表wc_out中的内容如下。

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| odps       | 1          |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
public class WordCount {
    public static class TokenizerMapper extends MapperBase {
        private Record word;
        private Record one;
        @Override
            public void setup(TaskContext context) throws IOException {
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
            System.out.println("TaskID:" + context.getTaskID().toString());
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                word.set(new Object[] { record.get(i).toString() });
                context.write(word, one);
            }
        }
    }
    /**
       * A combiner class that combines map output by sum them.
       **/
    public static class SumCombiner extends ReducerBase {
```

```
    public static class SumCombiner extends ReducerBase {
        private Record count;
        @Override
            public void setup(TaskContext context) throws IOException {
            count = context.createMapOutputValueRecord();
        }
        /**Combiner实现的接口和Reducer一样，是可以立即在Mapper本地执行的一个Reduce，作用是减少Mapper的输出量。*/
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long c = 0;
            while (values.hasNext()) {
                Record val = values.next();
                c += (Long) val.get(0);
            }
            count.set(0, c);
            context.write(key, count);
        }
    }
    /**
       * A reducer class that just emits the sum of the input values.
       **/
    public static class SumReducer extends ReducerBase {
        private Record result = null;
        @Override
            public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long count = 0;
            while (values.hasNext()) {
                Record val = values.next();
                count += (Long) val.get(0);
            }
            result.set(0, key.get(0));
            result.set(1, count);
            context.write(result);
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: WordCount <in_table> <out_table>");
            System.exit(2);
        }
        JobConf job = new JobConf();
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(SumCombiner.class);
        job.setReducerClass(SumReducer.class);
        /**设置Mapper中间结果的key和value的Schema，Mapper的中间结果输出也是Record的形式。*/
        job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
        /**设置输入和输出的表信息。*/
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        JobClient.runJob(job);
    }
}
```

6.5.2. MapOnly示例

对于MapOnly的作业，Map直接将<Key,Value>信息输出到MaxCompute的表中。您只需要指定输出表即可，无需指定Map输出的Key/Value元信息。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好MapOnly的测试表和资源。
   i. 创建测试表。

```
create table wc_in (key string, value string);
create table wc_out(key string, cnt bigint);
```

ii. 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. 使用Tunnel导入数据。

```
tunnel upload data wc_in;
```

导入wc_in表的数据文件data的内容。

```
hello,odps
hello,odps
```

测试步骤

在MaxCompute客户端中执行MapOnly。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.MapOnly wc_in wc_out map
```

预期结果

作业成功结束后，输出表wc_out中的内容如下。

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| hello      | 1          |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
public class MapOnly {
    public static class MapperClass extends MapperBase {
        @Override
            public void setup(TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.setup", false);
            /**Main函数在jobconf里设置了option.mapper.setup为true，才会执行下面的逻辑。*/
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, "setup");
                result.set(1, 1L);
                context.write(result);
            }
        }
        @Override
            public void map(long key, Record record, TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.map", false);
            /**Main函数在jobconf里设置了option.mapper.map为true，才会执行下面的逻辑。*/
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, record.get(0));
                result.set(1, 1L);
                context.write(result);
            }
        }
        @Override
            public void cleanup(TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.cleanup", false);
            /**Main函数在jobconf里设置了option.mapper.cleanup为true，才会执行下面的逻辑。*/
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, "cleanup");
                result.set(1, 1L);
                context.write(result);
            }
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2 && args.length != 3) {
            System.err.println("Usage: OnlyMapper <in_table> <out_table> [setup|map|cleanup]");
            System.exit(2);
        }
        JobConf job = new JobConf();
        job.setMapperClass(MapperClass.class);
        /**对于MapOnly的作业，必须显式设置reducer的个数为0。*/
        job.setNumReduceTasks(0);
        /**设置输入输出的表信息。*/
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        if (args.length == 3) {
            String options = new String(args[2]);
            /**jobconf中可以设置自定义的<key,value>值，在mapper中通过context的getJobConf可以获取到相关的设置。*/
            if (options.contains("setup")) {
                job.setBoolean("option.mapper.setup", true);
            }
            if (options.contains("map")) {
                job.setBoolean("option.mapper.map", true);
            }
            if (options.contains("cleanup")) {
                job.setBoolean("option.mapper.cleanup", true);
            }
        }
        JobClient.runJob(job);
    }
}
```

6.5.3. 多路输入输出示例

本文为您介绍MapReduce的多路输入输出示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备多路输入输出的测试表和资源。
   i. 创建测试表。

```
create table wc_in1(key string, value string);
create table wc_in2(key string, value string);
create table mr_multiinout_out1 (key string, cnt bigint);
create table mr_multiinout_out2 (key string, cnt bigint) partitioned by (a string, b string);
alter table mr_multiinout_out2 add partition (a='1', b='1');
alter table mr_multiinout_out2 add partition (a='2', b='2');
```

   ii. 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. 使用Tunnel导入数据。

```
tunnel upload data1 wc_in1;
tunnel upload data2 wc_in2;
```

导入wc_in1表的数据如下。

```
hello,odps
```

导入wc_in2表的数据如下。

```
hello,world
```

测试步骤

在MaxCompute客户端中执行MultipleInOut。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.MultipleInOut wc_in1,wc_in2 mr_multiinout_out1,mr_multiinout_out2|a=1/b=1|out1,mr_multiinout_
out2|a=2/b=2|out2;
```

预期结果

作业成功结束后，mr_multiinout_out1中的内容如下。

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| default    | 1          |
+------------+------------+
```

mr_multiinout_out2中的内容如下。

```
+--------+------------+---+---+
| key    | cnt        | a | b |
+--------+------------+---+---+
| odps   | 1          | 1 | 1 |
| world  | 1          | 1 | 1 |
| out1   | 1          | 1 | 1 |
| hello  | 2          | 2 | 2 |
| out2   | 1          | 2 | 2 |
+--------+------------+---+---+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedHashMap;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
```

```
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
     * Multi input & output example.
     **/
public class MultipleInOut {
    public static class TokenizerMapper extends MapperBase {
        Record word;
        Record one;
        @Override
            public void setup(TaskContext context) throws IOException {
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                word.set(new Object[] { record.get(i).toString() });
                context.write(word, one);
            }
        }
    }
    public static class SumReducer extends ReducerBase {
        private Record result;
        private Record result1;
        private Record result2;
        @Override
            public void setup(TaskContext context) throws IOException {
            /**对于不同的输出需要创建不同的Record，通过Label来区分。*/
            result = context.createOutputRecord();
            result1 = context.createOutputRecord("out1");
            result2 = context.createOutputRecord("out2");
        }
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long count = 0;
            while (values.hasNext()) {
                Record val = values.next();
                count += (Long) val.get(0);
            }
            long mod = count % 3;
            if (mod == 0) {
                result.set(0, key.get(0));
                result.set(1, count);
                /**不指定Label，默认(default)输出。*/
                context.write(result);
            } else if (mod == 1) {
                result1.set(0, key.get(0));
                result1.set(1, count);
                context.write(result1, "out1");
            } else {
                result2.set(0, key.get(0));
                result2.set(1, count);
                context.write(result2, "out2");
            }
        }
        @Override
            public void cleanup(TaskContext context) throws IOException {
            Record result = context.createOutputRecord();
            result.set(0, "default");
            result.set(1, 1L);
            context.write(result);
            Record result1 = context.createOutputRecord("out1");
            result1.set(0, "out1");
            result1.set(1, 1L);
            context.write(result1, "out1");
```

```
            Record result2 = context.createOutputRecord("out2");
            result2.set(0, "out2");
            result2.set(1, 1L);
            context.write(result2, "out2");
        }
    }
    /**将分区字符串如"ds=1/pt=2"转为Map的形式。*/
    public static LinkedHashMap<String, String> convertPartSpecToMap(
        String partSpec) {
        LinkedHashMap<String, String> map = new LinkedHashMap<String, String>();
        if (partSpec != null && !partSpec.trim().isEmpty()) {
            String[] parts = partSpec.split("/");
            for (String part : parts) {
                String[] ss = part.split("=");
                if (ss.length != 2) {
                    throw new RuntimeException("ODPS-0730001: error part spec format: "
                                               + partSpec);
                }
                map.put(ss[0], ss[1]);
            }
        }
        return map;
    }
    public static void main(String[] args) throws Exception {
        String[] inputs = null;
        String[] outputs = null;
        if (args.length == 2) {
            inputs = args[0].split(",");
            outputs = args[1].split(",");
        } else {
            System.err.println("MultipleInOut in... out...");
            System.exit(1);
        }
        JobConf job = new JobConf();
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(SumReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
        /**解析用户的输入表字符串。*/
        for (String in : inputs) {
            String[] ss = in.split("\\|");
            if (ss.length == 1) {
                InputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
            } else if (ss.length == 2) {
                LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
                InputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
            } else {
                System.err.println("Style of input: " + in + " is not right");
                System.exit(1);
            }
        }
        /**解析用户的输出表字符串。*/
        for (String out : outputs) {
            String[] ss = out.split("\\|");
            if (ss.length == 1) {
                OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
            } else if (ss.length == 2) {
                LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
                OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
            } else if (ss.length == 3) {
                if (ss[1].isEmpty()) {
                    LinkedHashMap<String, String> map = convertPartSpecToMap(ss[2]);
                    OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
                } else {
                    LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
                    OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map)
                                         .label(ss[2]).build(), job);
                }
            } else {
                System.err.println("Style of output: " + out + " is not right");
                System.exit(1);
            }
        }
        JobClient.runJob(job);
    }
}
```

```
}
```

6.5.4. 多任务示例

本文为您介绍MapReduce的多任务示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好MultiJobs测试表和资源。
   i. 创建测试表。

```
create table mr_empty (key string, value string);
create table mr_multijobs_out (value bigint);
```

   ii. 添加测试资源。

```
add table mr_multijobs_out as multijobs_res_table -f;
add jar data\resources\mapreduce-examples.jar -f;
```

测试步骤

在MaxCompute客户端中执行MultiJobs。

```
jar -resources mapreduce-examples.jar,multijobs_res_table -classpath data\resources\mapreduce-examples.jar
 com.aliyun.odps.mapred.open.example.MultiJobs mr_multijobs_out;
```

预期结果

作业成功结束后，输出表mr_multijobs_out中的内容如下。

```
+------------+
| value      |
+------------+
| 0          |
+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
     * MultiJobs
     *
     * Running multiple job
     *
     **/
public class MultiJobs {
    public static class InitMapper extends MapperBase {
        @Override
            public void setup(TaskContext context) throws IOException {
            Record record = context.createOutputRecord();
            long v = context.getJobConf().getLong("multijobs.value", 2);
            record.set(0, v);
            context.write(record);
        }
    }
    public static class DecreaseMapper extends MapperBase {
        @Override
            public void cleanup(TaskContext context) throws IOException {
            /**从JobConf中获取main函数中定义的变量值。*/
```

```
            long expect = context.getJobConf().getLong("multijobs.expect.value", -1);
            long v = -1;
            int count = 0;
            /**读取资源表里面的数据，这个表是上一个job的输出表。*/
            Iterator<Record> iter = context.readResourceTable("multijobs_res_table");
            while (iter.hasNext()) {
                Record r = iter.next();
                v = (Long) r.get(0);
                if (expect != v) {
                    throw new IOException("expect: " + expect + ", but: " + v);
                }
                count++;
            }
            if (count != 1) {
                throw new IOException("res_table should have 1 record, but: " + count);
            }
            Record record = context.createOutputRecord();
            v--;
            record.set(0, v);
            context.write(record);
            /**设置counter，counter在作业成功结束后，可以在main函数中获取到。*/
            context.getCounter("multijobs", "value").setValue(v);
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: TestMultiJobs <table>");
            System.exit(1);
        }
        String tbl = args[0];
        long iterCount = 2;
        System.err.println("Start to run init job.");
        JobConf initJob = new JobConf();
        initJob.setLong("multijobs.value", iterCount);
        initJob.setMapperClass(InitMapper.class);
        InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), initJob);
        OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), initJob);
        initJob.setMapOutputKeySchema(SchemaUtils.fromString("key:string"));
        initJob.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
        /**MapOnly作业需要显式设置reducer的数目为0。*/
        initJob.setNumReduceTasks(0);
        JobClient.runJob(initJob);
        while (true) {
            System.err.println("Start to run iter job, count: " + iterCount);
            JobConf decJob = new JobConf();
            decJob.setLong("multijobs.expect.value", iterCount);
            decJob.setMapperClass(DecreaseMapper.class);
            InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), decJob);
            OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), decJob);
            /**MapOnly作业需要显式设置reducer的数目为0。*/
            decJob.setNumReduceTasks(0);
            RunningJob rJob = JobClient.runJob(decJob);
            iterCount--;
            /**如果迭代次数已经达到，则退出循环。*/
            if (rJob.getCounters().findCounter("multijobs", "value").getValue() == 0) {
                break;
            }
        }
        if (iterCount != 0) {
            throw new IOException("Job failed.");
        }
    }
}
```

6.5.5. 二次排序示例

本文为您介绍MapReduce的二次排序示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好SecondarySort的测试表和资源。

i. 创建测试表。

```
create table ss_in(key bigint, value bigint);
create table ss_out(key bigint, value bigint)
```

ii. 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. 使用Tunnel导入数据。

```
tunnel upload data ss_in;
```

导入ss_in表的数据如下。

```
1,2
2,1
1,1
2,2
```

测试步骤

在MaxCompute客户端中执行SecondarySort。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.SecondarySort ss_in ss_out;
```

预期结果

作业成功结束后，输出表ss_out中的内容如下。

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1          | 1          |
| 1          | 2          |
| 2          | 1          |
| 2          | 2          |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
/**
     *
     * This is an example ODPS Map/Reduce application. It reads the input table that
     * must contain two integers per record. The output is sorted by the first and
     * second number and grouped on the first number.
     *
     **/
public class SecondarySort {
    /**
       * Read two integers from each line and generate a key, value pair as ((left,
       * right), right).
       **/
    public static class MapClass extends MapperBase {
        private Record key;
        private Record value;
        @Override
            public void setup(TaskContext context) throws IOException {
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
```

```
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            long left = 0;
            long right = 0;
            if (record.getColumnCount() > 0) {
                left = (Long) record.get(0);
                if (record.getColumnCount() > 1) {
                    right = (Long) record.get(1);
                }
                key.set(new Object[] { (Long) left, (Long) right });
                value.set(new Object[] { (Long) right });
                context.write(key, value);
            }
        }
    }
    /**
       * A reducer class that just emits the sum of the input values.
       **/
    public static class ReduceClass extends ReducerBase {
        private Record result = null;
        @Override
            public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            result.set(0, key.get(0));
            while (values.hasNext()) {
                Record value = values.next();
                result.set(1, value.get(0));
                context.write(result);
            }
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: secondarysrot <in> <out>");
            System.exit(2);
        }
        JobConf job = new JobConf();
        job.setMapperClass(MapClass.class);
        job.setReducerClass(ReduceClass.class);
        /**将多列设置为Key。*/
        //compare first and second parts of the pair
        job.setOutputKeySortColumns(new String[] { "i1", "i2" });
        //partition based on the first part of the pair
        job.setPartitionColumns(new String[] { "i1" });
        //grouping comparator based on the first part of the pair
        job.setOutputGroupingColumns(new String[] { "i1" });
        //the map output is LongPair, Long
        job.setMapOutputKeySchema(SchemaUtils.fromString("i1:bigint,i2:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("i2x:bigint"));
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        JobClient.runJob(job);
        System.exit(0);
    }
}
```

6.5.6. 使用资源示例

本文为您介绍MapReduce的使用资源示例。

测试准备

1. 准备好测试程序的Jar包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 在MaxCompute客户端中执行如下操作，准备好测试表和资源。

i. 创建测试表。

```
create table mr_upload_src(key bigint, value string);
```

ii. 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
add file data\resources\import.txt -f;
```

iii. *import.txt*的数据内容如下。

```
1000,odps
```

测试步骤

在MaxCompute客户端中执行Upload。

```
jar -resources mapreduce-examples.jar,import.txt -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Upload import.txt mr_upload_src;
```

预期结果

作业成功结束后，输出表mr_upload_src中的内容如下。

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1000       | odps       |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.BufferedInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
     * Upload
     *
     * Import data from text file into table
     *
     **/
public class Upload {
    public static class UploadMapper extends MapperBase {
        @Override
            public void setup(TaskContext context) throws IOException {
            Record record = context.createOutputRecord();
            StringBuilder importdata = new StringBuilder();
            BufferedInputStream bufferedInput = null;
            try {
                byte[] buffer = new byte[1024];
                int bytesRead = 0;
                String filename = context.getJobConf().get("import.filename");
                bufferedInput = context.readResourceFileAsStream(filename);
                while ((bytesRead = bufferedInput.read(buffer)) != -1) {
                    String chunk = new String(buffer, 0, bytesRead);
                    importdata.append(chunk);
                }
                String lines[] = importdata.toString().split("\n");
                for (int i = 0; i < lines.length; i++) {
                    String[] ss = lines[i].split(",");
                    record.set(0, Long.parseLong(ss[0].trim()));
                    record.set(1, ss[1].trim());
                    context.write(record);
                }
            } catch (FileNotFoundException ex) {
                throw new IOException(ex);
            } catch (IOException ex) {
                throw new IOException(ex);
            } finally {
            }
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: Upload <import_txt> <out_table>");
            System.exit(2);
        }
        JobConf job = new JobConf();
        job.setMapperClass(UploadMapper.class);
        /**设置资源名字，可以在map中通过jobconf获取到。*/
        job.set("import.filename", args[0]);
        /**maponly作业需要显式设置reducer的数目为0。*/
        job.setNumReduceTasks(0);
        job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
        InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        JobClient.runJob(job);
    }
}
```

您可以通过以下两种方式设置JobConf。

- 通过SDK中JobConf的接口设置，本示例即是通过此方法实现。
- 在Jar命令行中，通过-conf参数指定新的JobConf文件。

6.5.7. 使用Counter示例

本文为您介绍MapReduce的使用Counter示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好UserDefinedCounters测试表和资源。
   i. 创建测试表。
   ```
   create table wc_in (key string, value string);
   create table wc_out(key string, cnt bigint);
   ```
   ii. 添加测试资源。
   ```
   add jar data\resources\mapreduce-examples.jar -f;
   ```
3. 使用Tunnel导入数据。
   ```
   tunnel upload data wc_in;
   ```
   导入wc_in表的数据如下。
   ```
   hello,odps
   ```

测试步骤

在MaxCompute客户端中执行UserDefinedCounters。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.UserDefinedCounters wc_in wc_out
```

预期结果

作业成功结束后，可以看到Counters的输出如下。

```
Counters: 3
com.aliyun.odps.mapred.open.example.UserDefinedCounters$MyCounter
MAP_TASKS=1
REDUCE_TASKS=1
TOTAL_TASKS=2
```

输出表wc_out中的内容如下。

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| odps       | 1          |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.counter.Counters;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
/**
```

```
     *
     * User Defined Counters
     *
     **/
public class UserDefinedCounters {
    enum MyCounter {
        TOTAL_TASKS, MAP_TASKS, REDUCE_TASKS
    }
    public static class TokenizerMapper extends MapperBase {
        private Record word;
        private Record one;
        @Override
            public void setup(TaskContext context) throws IOException {
            super.setup(context);
            Counter map_tasks = context.getCounter(MyCounter.MAP_TASKS);
            Counter total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
            map_tasks.increment(1);
            total_tasks.increment(1);
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                word.set(new Object[] { record.get(i).toString() });
                context.write(word, one);
            }
        }
    }
    public static class SumReducer extends ReducerBase {
        private Record result = null;
        @Override
            public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
            Counter reduce_tasks = context.getCounter(MyCounter.REDUCE_TASKS);
            Counter total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
            reduce_tasks.increment(1);
            total_tasks.increment(1);
        }
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long count = 0;
            while (values.hasNext()) {
                Record val = values.next();
                count += (Long) val.get(0);
            }
            result.set(0, key.get(0));
            result.set(1, count);
            context.write(result);
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err
                .println("Usage: TestUserDefinedCounters <in_table> <out_table>");
            System.exit(2);
        }
        JobConf job = new JobConf();
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(SumReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        RunningJob rJob = JobClient.runJob(job);
        /**在作业成功结束后，可以获取到Job里面的自定义Counter的值。*/
        Counters counters = rJob.getCounters();
        long m = counters.findCounter(MyCounter.MAP_TASKS).getValue();
        long r = counters.findCounter(MyCounter.REDUCE_TASKS).getValue();
        long total = counters.findCounter(MyCounter.TOTAL_TASKS).getValue();
        System.exit(0);
    }
```

```
}
```

6.5.8. Grep示例

本文为您介绍MapReduce的Grep示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好Grep测试表和资源。
   i. 创建测试表。
   ```
   create table mr_src(key string, value string);
   create table mr_grep_tmp (key string, cnt bigint);
   create table mr_grep_out (key bigint, value string);
   ```
   ii. 添加测试资源。
   ```
   add jar data\resources\mapreduce-examples.jar -f;
   ```
3. 使用Tunnel导入数据。
   ```
   tunnel upload data mr_src;
   ```
   导入mr_src表的数据如下。
   ```
   hello,odps
   hello,world
   ```

测试步骤

在MaxCompute客户端中执行Grep。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Grep mr_src mr_grep_tmp mr_grep_out hello;
```

预期结果

作业成功结束后，输出表mr_grep_out中的内容如下。

```
+------------+------------+
| key        | value      |
+------------+------------+
| 2          | hello      |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.Mapper;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
     *
     * Extracts matching regexs from input files and counts them.
     *
     **/
public class Grep {
    /**
```

```
   * RegexMapper
   **/
public class RegexMapper extends MapperBase {
    private Pattern pattern;
    private int group;
    private Record word;
    private Record one;
    @Override
        public void setup(TaskContext context) throws IOException {
        JobConf job = (JobConf) context.getJobConf();
        pattern = Pattern.compile(job.get("mapred.mapper.regex"));
        group = job.getInt("mapred.mapper.regex.group", 0);
        word = context.createMapOutputKeyRecord();
        one = context.createMapOutputValueRecord();
        one.set(new Object[] { 1L });
    }
    @Override
        public void map(long recordNum, Record record, TaskContext context) throws IOException {
        for (int i = 0; i < record.getColumnCount(); ++i) {
            String text = record.get(i).toString();
            Matcher matcher = pattern.matcher(text);
            while (matcher.find()) {
                word.set(new Object[] { matcher.group(group) });
                context.write(word, one);
            }
        }
    }
}
/**
   * LongSumReducer
   **/
public class LongSumReducer extends ReducerBase {
    private Record result = null;
    @Override
        public void setup(TaskContext context) throws IOException {
        result = context.createOutputRecord();
    }
    @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException {
        long count = 0;
        while (values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        result.set(0, key.get(0));
        result.set(1, count);
        context.write(result);
    }
}
/**
   * A {@link Mapper} that swaps keys and values.
   **/
public class InverseMapper extends MapperBase {
    private Record word;
    private Record count;
    @Override
        public void setup(TaskContext context) throws IOException {
        word = context.createMapOutputValueRecord();
        count = context.createMapOutputKeyRecord();
    }
    /**
     * The inverse function. Input keys and values are swapped.
     **/
    @Override
        public void map(long recordNum, Record record, TaskContext context) throws IOException {
        word.set(new Object[] { record.get(0).toString() });
        count.set(new Object[] { (Long) record.get(1) });
        context.write(count, word);
    }
}
/**
   * IdentityReducer
   **/
public class IdentityReducer extends ReducerBase {
    private Record result = null;
```

```
        @Override
            public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        /** Writes all keys and values directly to output. **/
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException {
            result.set(0, key.get(0));
            while (values.hasNext()) {
                Record val = values.next();
                result.set(1, val.get(0));
                context.write(result);
            }
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length < 4) {
            System.err.println("Grep <inDir> <tmpDir> <outDir> <regex> [<group>]");
            System.exit(2);
        }
        JobConf grepJob = new JobConf();
        grepJob.setMapperClass(RegexMapper.class);
        grepJob.setReducerClass(LongSumReducer.class);
        grepJob.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
        grepJob.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), grepJob);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), grepJob);
        /**设置grepJob的grep的正则表达式。*/
        grepJob.set("mapred.mapper.regex", args[3]);
        if (args.length == 5) {
            grepJob.set("mapred.mapper.regex.group", args[4]);
        }
        @SuppressWarnings("unused")
            RunningJob rjGrep = JobClient.runJob(grepJob);
        /**grepJob的输出作为sortJob的输入。*/
        JobConf sortJob = new JobConf();
        sortJob.setMapperClass(InverseMapper.class);
        sortJob.setReducerClass(IdentityReducer.class);
        sortJob.setMapOutputKeySchema(SchemaUtils.fromString("count:bigint"));
        sortJob.setMapOutputValueSchema(SchemaUtils.fromString("word:string"));
        InputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), sortJob);
        OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), sortJob);
        sortJob.setNumReduceTasks(1); // write a single file
        sortJob.setOutputKeySortColumns(new String[] { "count" });
        @SuppressWarnings("unused")
            RunningJob rjSort = JobClient.runJob(sortJob);
    }
}
```

6.5.9. Join示例

MaxCompute MapReduce框架自身并不支持Join逻辑，但您可以在自己的Map或Reduce函数中实现数据的Join。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好Join的测试表和资源。
   i. 创建测试表。 对表mr_Join_src1与表mr_Join_src2执行Join操作，输出表mr_Join_out。

   ```
   create table mr_Join_src1(key bigint, value string);
   create table mr_Join_src2(key bigint, value string);
   create table mr_Join_out(key bigint, value1 string, value2 string);
   ```

   ii. 添加测试资源。

   ```
   add jar data\resources\mapreduce-examples.jar -f;
   ```

3. 使用Tunnel导入数据。

   ```
   tunnel upload data1 mr_Join_src1;
   tunnel upload data2 mr_Join_src2;
   ```

   导入mr_Join_src1数据的内容。

```
1,hello
2,odps
```

导入mr_Join_src2数据的内容。

```
1,odps
3,hello
4,odps
```

测试步骤

在MaxCompute客户端中执行Join。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Join mr_Join_src1 mr_Join_src2 mr_Join_out;
```

预期结果

作业成功结束后，输出表mr_Join_out中的内容如下。 其中value1是mr_Join_src1的value值，value2是mr_Join_src2的value值。

```
+------------+------------+------------+
| key        | value1     | value2     |
+------------+------------+------------+
|  1         | hello      |  odps      |
+------------+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
     * Join, mr_Join_src1/mr_Join_src2(key bigint, value string), mr_Join_out(key
     * bigint, value1 string, value2 string)
     *
     */
public class Join {
    public static final Log LOG = LogFactory.getLog(Join.class);
    public static class JoinMapper extends MapperBase {
        private Record mapkey;
        private Record mapvalue;
        private long tag;
        @Override
            public void setup(TaskContext context) throws IOException {
            mapkey = context.createMapOutputKeyRecord();
            mapvalue = context.createMapOutputValueRecord();
            tag = context.getInputTableInfo().getLabel().equals("left") ? 0 : 1;
        }
        @Override
            public void map(long key, Record record, TaskContext context)
            throws IOException {
            mapkey.set(0, record.get(0));
            mapkey.set(1, tag);
            for (int i = 1; i < record.getColumnCount(); i++) {
                mapvalue.set(i - 1, record.get(i));
            }
            context.write(mapkey, mapvalue);
        }
    }
```

```
    public static class JoinReducer extends ReducerBase {
        private Record result = null;
        @Override
            public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        /**reduce函数每次的输入会是key相同的所有record。*/
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long k = key.getBigint(0);
            List<Object[]> leftValues = new ArrayList<Object[]>();
            /**由于设置了outputKeySortColumn是key + tag组合，这样可以保证reduce函数的输入record中，left表的record数据在前面。*/
            while (values.hasNext()) {
                Record value = values.next();
                long tag = (Long) key.get(1);
                /**左表的数据会先缓存到内存中。*/
                if (tag == 0) {
                    leftValues.add(value.toArray().clone());
                } else {
                    /**右表的数据会与所有左表的数据进行join输出，此时左表的数据已经全部在内存里了。*/
                    /**这个实现只是一个功能展示，性能较低，不建议用于实际生产。*/
                    for (Object[] leftValue : leftValues) {
                        int index = 0;
                        result.set(index++, k);
                        for (int i = 0; i < leftValue.length; i++) {
                            result.set(index++, leftValue[i]);
                        }
                        for (int i = 0; i < value.getColumnCount(); i++) {
                            result.set(index++, value.get(i));
                        }
                        context.write(result);
                    }
                }
            }
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 3) {
            System.err.println("Usage: Join <input table1> <input table2> <out>");
            System.exit(2);
        }
        JobConf job = new JobConf();
        job.setMapperClass(JoinMapper.class);
        job.setReducerClass(JoinReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,tag:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
        job.setPartitionColumns(new String[]{"key"});
        job.setOutputKeySortColumns(new String[]{"key", "tag"});
        job.setOutputGroupingColumns(new String[]{"key"});
        job.setNumReduceTasks(1);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).label("left").build(), job);
        InputUtils.addTable(TableInfo.builder().tableName(args[1]).label("right").build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), job);
        JobClient.runJob(job);
    }
}
```

6.5.10. Sleep示例

本文为您介绍MapReduce的Sleep示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好SleepJob的测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

测试步骤

在MaxCompute客户端中执行Sleep。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Sleep 10;
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Sleep 100;
```

预期结果

作业成功结束后，对比不同Sleep时长的运行时间，可以看到效果。

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.conf.JobConf;
public class Sleep {
  private static final String SLEEP_SECS = "sleep.secs";
  public static class MapperClass extends MapperBase {
    /**由于没有输入数据，map函数不会被执行，相关逻辑只能写在setup中。*/
    @Override
    public void setup(TaskContext context) throws IOException {
      try {
        /**获取jobconf中设置的sleep秒数，进行sleep。*/
        Thread.sleep(context.getJobConf().getInt(SLEEP_SECS, 1) * 1000);
      } catch (InterruptedException e) {
        throw new RuntimeException(e);
      }
    }
  }
  public static void main(String[] args) throws Exception {
    if (args.length != 1) {
      System.err.println("Usage: Sleep <sleep_secs>");
      System.exit(-1);
    }
    JobConf job = new JobConf();
    job.setMapperClass(MapperClass.class);
    /**这个实例也是一个MapOnly的，所以需要设置reducer个数为0。*/
    job.setNumReduceTasks(0);
    /**由于没有输入表，mapper的个数需要由用户明确进行指定。*/
    job.setNumMapTasks(1);
    job.set(SLEEP_SECS, args[0]);
    JobClient.runJob(job);
  }
}
```

6.5.11. Unique示例

本文为您介绍MapReduce的Unique示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好Unique的测试表和资源。
   i. 创建测试表。
   ```
   create table ss_in(key bigint, value bigint);
   create table ss_out(key bigint, value bigint);
   ```
   ii. 添加测试资源。
   ```
   add jar data\resources\mapreduce-examples.jar -f;
   ```
3. 使用Tunnel导入数据。
   ```
   tunnel upload data ss_in;
   ```
   导入ss_in表的数据文件data的内容如下。

```
1,1
1,1
2,2
2,2
```

测试步骤

在MaxCompute客户端中执行Unique。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Unique ss_in ss_out key;
```

预期结果

作业成功结束后，输出表ss_out中的内容如下。

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1          |   1        |
| 2          |   2        |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
     * Unique Remove duplicate words
     *
     **/
public class Unique {
    public static class OutputSchemaMapper extends MapperBase {
        private Record key;
        private Record value;
        @Override
            public void setup(TaskContext context) throws IOException {
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            long left = 0;
            long right = 0;
            if (record.getColumnCount() > 0) {
                left = (Long) record.get(0);
                if (record.getColumnCount() > 1) {
                    right = (Long) record.get(1);
                }
                key.set(new Object[] { (Long) left, (Long) right });
                value.set(new Object[] { (Long) left, (Long) right });
                context.write(key, value);
            }
        }
    }
    public static class OutputSchemaReducer extends ReducerBase {
        private Record result = null;
        @Override
            public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
```

```
        }
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            result.set(0, key.get(0));
            while (values.hasNext()) {
                Record value = values.next();
                result.set(1, value.get(1));
            }
            context.write(result);
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length > 3 || args.length < 2) {
            System.err.println("Usage: unique <in> <out> [key|value|all]");
            System.exit(2);
        }
        String ops = "all";
        if (args.length == 3) {
            ops = args[2];
        }
        /** Reduce的输入分组是由setOutputGroupingColumns的设置来决定的，这个参数如果不设置，默认就是MapOutputKeySchema。*/
        // Key Unique
        if (ops.equals("key")) {
            JobConf job = new JobConf();
            job.setMapperClass(OutputSchemaMapper.class);
            job.setReducerClass(OutputSchemaReducer.class);
            job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
            job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
            job.setPartitionColumns(new String[] { "key" });
            job.setOutputKeySortColumns(new String[] { "key", "value" });
            job.setOutputGroupingColumns(new String[] { "key" });
            job.set("tablename2", args[1]);
            job.setNumReduceTasks(1);
            job.setInt("table.counter", 0);
            InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
            OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
            JobClient.runJob(job);
        }
        // Key&Value Unique
        if (ops.equals("all")) {
            JobConf job = new JobConf();
            job.setMapperClass(OutputSchemaMapper.class);
            job.setReducerClass(OutputSchemaReducer.class);
            job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
            job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
            job.setPartitionColumns(new String[] { "key" });
            job.setOutputKeySortColumns(new String[] { "key", "value" });
            job.setOutputGroupingColumns(new String[] { "key", "value" });
            job.set("tablename2", args[1]);
            job.setNumReduceTasks(1);
            job.setInt("table.counter", 0);
            InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
            OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
            JobClient.runJob(job);
        }
        // Value Unique
        if (ops.equals("value")) {
            JobConf job = new JobConf();
            job.setMapperClass(OutputSchemaMapper.class);
            job.setReducerClass(OutputSchemaReducer.class);
            job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
            job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
            job.setPartitionColumns(new String[] { "value" });
            job.setOutputKeySortColumns(new String[] { "value" });
            job.setOutputGroupingColumns(new String[] { "value" });
            job.set("tablename2", args[1]);
            job.setNumReduceTasks(1);
            job.setInt("table.counter", 0);
            InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
            OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
            JobClient.runJob(job);
        }
    }
}
```

6.5.12. Sort示例

本文为您介绍MapReduce的Sort示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好Sort的测试表和资源。
   i. 创建测试表。
   ```
   create table ss_in(key bigint, value bigint);
   create table ss_out(key bigint, value bigint);
   ```
   ii. 添加测试资源。
   ```
   add jar data\resources\mapreduce-examples.jar -f;
   ```
3. 使用Tunnel导入数据。
   ```
   tunnel upload data ss_in;
   ```
   导入ss_in表的数据文件data的内容。
   ```
   2,1
   1,1
   3,1
   ```

测试步骤

在MaxCompute客户端中执行Sort。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Sort ss_in ss_out;
```

预期结果

作业成功结束后，输出表ss_out中的内容如下。

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1          | 1          |
| 2          | 1          |
| 3          | 1          |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Date;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.example.lib.IdentityReducer;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
     * This is the trivial map/reduce program that does absolutely nothing other
     * than use the framework to fragment and sort the input values.
     *
     **/
public class Sort {
    static int printUsage() {
        System.out.println("sort <input> <output>");
        return -1;
    }
    /**
       * Implements the identity function, mapping record's first two columns to
       * outputs.
       **/
    public static class IdentityMapper extends MapperBase {
        private Record key;
        private Record value;
        @Override
            public void setup(TaskContext context) throws IOException {
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            key.set(new Object[] { (Long) record.get(0) });
            value.set(new Object[] { (Long) record.get(1) });
            context.write(key, value);
        }
    }
    /**
       * The main driver for sort program. Invoke this method to submit the
       * map/reduce job.
       *
       * @throws IOException
       *           When there is communication problems with the job tracker.
       **/
    public static void main(String[] args) throws Exception {
        JobConf jobConf = new JobConf();
        jobConf.setMapperClass(IdentityMapper.class);
        jobConf.setReducerClass(IdentityReducer.class);
        /**为了全局有序，这里设置了reducer的个数为1，所有的数据都会集中到一个reducer上面。*/
        /**只能用于小数据量，大数据量需要考虑其他的方式，比如TeraSort。*/
        jobConf.setNumReduceTasks(1);
        jobConf.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint"));
        jobConf.setMapOutputValueSchema(SchemaUtils.fromString("value:bigint"));
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), jobConf);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), jobConf);
        Date startTime = new Date();
        System.out.println("Job started: " + startTime);
        JobClient.runJob(jobConf);
        Date end_time = new Date();
        System.out.println("Job ended: " + end_time);
        System.out.println("The job took " + (end_time.getTime() - startTime.getTime()) / 1000 + " seconds.");
    }
}
```

6.5.13. 分区表输入示例

本文为您介绍MapReduce的分区表输入示例。

- 示例一：

```
public static void main(String[] args) throws Exception {
    JobConf job = new JobConf();
    ...
        LinkedHashMap<String, String> input = new LinkedHashMap<String, String>();
    input.put("pt", "123456");
    InputUtils.addTable(TableInfo.builder().tableName("input_table").partSpec(input).build(), job);
    LinkedHashMap<String, String> output = new LinkedHashMap<String, String>();
    output.put("ds", "654321");
    OutputUtils.addTable(TableInfo.builder().tableName("output_table").partSpec(output).build(), job);
    JobClient.runJob(job);
}
```

- 示例二：

```
package com.aliyun.odps.mapred.open.example;
...
    public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: WordCount <in_table> <out_table>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(SumCombiner.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    Account account = new AliyunAccount("my_access_id", "my_access_key");
    Odps odps = new Odps(account);
    odps.setEndpoint("odps_endpoint_url");
    odps.setDefaultProject("my_project");
    Table table = odps.tables().get(tblname);
    TableInfoBuilder builder = TableInfo.builder().tableName(tblname);
    for (Partition p : table.getPartitions()) {
        if (applicable(p)) {
            LinkedHashMap<String, String> partSpec = new LinkedHashMap<String, String>();
            for (String key : p.getPartitionSpec().keys()) {
                partSpec.put(key, p.getPartitionSpec().get(key));
            }
            InputUtils.addTable(builder.partSpec(partSpec).build(), job);
        }
    }
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    JobClient.runJob(job);
}
```

> 说明
> - 上述示例是使用MaxCompute SDK和MapReduce SDK组合实现MapReduce任务读取范围分区的示例。
> - 此段代码不能够编译执行，仅给出了main函数的示例。
> - 示例中applicable函数是用户逻辑，用于决定该分区是否可以作为MapReduce作业的输入。

6.5.14. Pipeline示例

本文为您介绍MapReduce的Pipeline示例。

测试准备

1. 准备好测试程序的JAR包，假设名字为*mapreduce-examples.jar*，本地存放路径为*data\resources*。
2. 准备好Pipeline的测试表和资源。
   i. 创建测试表。

   ```
   create table wc_in (key string, value string);
   create table wc_out(key string, cnt bigint);
   ```

   ii. 添加测试资源。

   ```
   add jar data\resources\mapreduce-examples.jar -f;
   ```

3. 使用Tunnel导入数据。

```
tunnel upload data wc_in;
```

导入wc_in表的数据文件data的内容。

```
hello,odps
```

测试步骤

在MaxCompute客户端中执行WordCount Pipeline。

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.WordCountPipeline wc_in wc_out;
```

预期结果

作业成功结束后，输出表wc_out中的内容如下。

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| odps       | 1          |
+------------+------------+
```

代码示例

Pom依赖信息，请参见注意事项。

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.Job;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.pipeline.Pipeline;
public class WordCountPipelineTest {
    public static class TokenizerMapper extends MapperBase {
        Record word;
        Record one;
        @Override
            public void setup(TaskContext context) throws IOException {
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.setBigint(0, 1L);
        }
        @Override
            public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                String[] words = record.get(i).toString().split("\\s+");
                for (String w : words) {
                    word.setString(0, w);
                    context.write(word, one);
                }
            }
        }
    }
    public static class SumReducer extends ReducerBase {
        private Record value;
        @Override
            public void setup(TaskContext context) throws IOException {
            value = context.createOutputValueRecord();
        }
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long count = 0;
            while (values.hasNext()) {
                Record val = values.next();
```

```
                count += (Long) val.get(0);
            }
            value.set(0, count);
            context.write(key, value);
        }
    }
    public static class IdentityReducer extends ReducerBase {
        private Record result;
        @Override
            public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        @Override
            public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            while (values.hasNext()) {
                result.set(0, key.get(0));
                result.set(1, values.next().get(0));
                context.write(result);
            }
        }
    }
    public static void main(String[] args) throws OdpsException {
        if (args.length != 2) {
            System.err.println("Usage: WordCountPipeline <in_table> <out_table>");
            System.exit(2);
        }
        Job job = new Job();
        /**构造Pipeline的过程中，如果不指定Mapper的OutputKeySortColumns、PartitionColumns、OutputGroupingColumns，框架会默认使用其Outpu
tKey作为此三者的默认配置。
         */
        Pipeline pipeline = Pipeline.builder()
            .addMapper(TokenizerMapper.class)
            .setOutputKeySchema(
            new Column[] { new Column("word", OdpsType.STRING) })
            .setOutputValueSchema(
            new Column[] { new Column("count", OdpsType.BIGINT) })
            .setOutputKeySortColumns(new String[] { "word" })
            .setPartitionColumns(new String[] { "word" })
            .setOutputGroupingColumns(new String[] { "word" })
            .addReducer(SumReducer.class)
            .setOutputKeySchema(
            new Column[] { new Column("word", OdpsType.STRING) })
            .setOutputValueSchema(
            new Column[] { new Column("count", OdpsType.BIGINT)})
            .addReducer(IdentityReducer.class).createPipeline();
        /**将pipeline的设置到jobconf中，如果需要设置combiner，是通过jobconf来设置。*/
        job.setPipeline(pipeline);
        /**设置输入输出表。*/
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());
        /**作业提交并等待结束。*/
        job.submit();
        job.waitForCompletion();
        System.exit(job.isSuccessful() == true ? 0 : 1);
    }
}
```

6.6. Java SDK

6.6.1. 原生SDK概述

本文为您介绍较为常用的MapReduce核心接口。

如果您使用Maven，可以从Maven库中搜索odps-sdk-mapred获取不同版本的Java SDK，相关配置信息如下。

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-mapred</artifactId>
    <version>0.26.2-public</version>
</dependency>
```

数据类型

MapReduce支持的数据类型为BIGINT、STRING、DOUBLE、BOOLEAN、DATETIME和DECIMAL类型。MaxCompute数据类型与Java数据类型的对应关系如下。

MaxCompute SQL Type	Java Type
BIGINT	Long
STRING	STRING
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	DATE
DECIMAL	BIGDECIMAL

MapReduce主要接口

主要接口	描述
MapperBase	用户自定义的Map函数需要继承自此类。处理输入表的记录对象，加工处理成键值对集合输出到Reduce阶段，或者不经过Reduce阶段直接输出结果记录到结果表。不经过Reduce阶段而直接输出计算结果的作业，也可称之为MapOnly作业。
ReducerBase	用户自定义的Reduce函数需要继承自此类。对与一个键（Key）关联的一组数值集（Values）进行归约计算。
TaskContext	是MapperBase及ReducerBase多个成员函数的输入参数之一，含有任务运行的上下文信息。
JobClient	用于提交和管理作业，提交方式包括阻塞（同步）方式及非阻塞（异步） 方式。
RunningJob	作业运行时对象，用于跟踪运行中的MapReduce作业实例。
JobConf	描述一个MapReduce任务的配置，通常在主程序（main函数）中定义JobConf对象，然后通过JobClient提交作业给MaxCompute服务。

MapperBase

主要函数接口。

主要接口	描述
void cleanup(TaskContext context)	在Map阶段结束时，map方法之后调用。
void map(long key, Record record, TaskContext context)	map方法，处理输入表的记录。
void setup(TaskContext context)	在Map阶段开始时，map方法之前调用。

ReducerBase

主要函数接口。

主要接口	描述
void cleanup(TaskContext context)	在Reduce阶段结束时，reduce方法之后调用。
void reduce(Record key, Iterator<Record > values, TaskContext context)	reduce方法，处理输入表的记录。
void setup(TaskContext context)	在Reduce阶段开始时，reduce方法之前调用。

TaskContext

主要函数接口。

主要接口	描述
TableInfo[] getOutputTableInfo()	获取输出的表信息。
Record createOutputRecord()	创建默认输出表的记录对象。
Record createOutputRecord(String label)	创建给定label输出表的记录对象。
Record createMapOutputKeyRecord()	创建Map输出Key的记录对象。
Record createMapOutputValueRecord()	创建Map输出Value的记录对象。

主要接口	描述
void write(Record record)	写记录到默认输出，用于Reduce端写出数据，可以在Reduce端多次调用。
void write(Record record, String label)	写记录到给定label输出，用于Reduce端写出数据。可以在 Reduce端多次调用。
void write(Record key, Record value)	Map写记录到中间结果，可以在Map函数中多次调用。 可以在Map端多次调用。
BufferedInputStream readResourceFileAsStream(String resourceName)	读取文件类型资源。
Iterator<Record > readResourceTable(String resourceName)	读取表类型资源。
Counter getCounter(Enum<? > name)	获取给定名称的Counter对象。
Counter getCounter(String group, String name)	获取给定组名和名称的Counter对象。
void progress()	向MapReduce框架报告心跳信息。 如果用户方法处理时间很长，且中间没有调用框架，可以调用这个方法避免task超时，框架默认600秒超时。

- MaxCompute的TaskContext接口中提供了progress功能，但此功能是为防止Worker长时间运行未结束，被框架误认为超时而被杀的情况出现。此接口更类似于向框架发送心跳信息，并不是用来汇报Worker进度。
- MaxCompute MapReduce默认Worker超时时间为10分钟（系统默认配置，不受用户控制），如果超过10分钟，Worker仍然没有向框架发送心跳（调用progress接口），框架会强制停止该Worker，MapReduce任务失败退出。因此，建议您在Mapper/Reducer函数中，定期调用progress接口，防止框架认为Worker超时，误杀任务。

JobConf

主要函数接口。

主要接口	描述
void setResources(String resourceNames)	声明本作业使用的资源。只有声明的资源才能在运行Mapper/Reducer时通过TaskContext对象读取。
void setMapOutputKeySchema(Column[] schema)	设置Mapper输出到Reducer的Key属性。
void setMapOutputValueSchema(Column[] schema)	设置Mapper输出到Reducer的Value属性。
void setOutputKeySortColumns(String[] cols)	设置Mapper输出到Reducer的Key排序列。
void setOutputGroupingColumns(String[] cols)	设置Key分组列。
void setMapperClass(Class<? extends Mapper > theClass)	设置作业的Mapper函数。
void setPartitionColumns(String[] cols)	设置作业指定的分区列。默认是Mapper输出Key的所有列。
void setReducerClass(Class<? extends Reducer > theClass)	设置作业的Reducer。
void setCombinerClass(Class<? extends Reducer > theClass)	设置作业的combiner。在Map端运行，作用类似于单个Map对本地的相同Key值做Reduce。
void setSplitSize(long size)	设置分片大小，单位MB，默认值256。
void setNumReduceTasks(int n)	设置Reducer任务数，默认为Mapper任务数的1/4。
void setMemoryForMapTask(int mem)	设置Mapper任务中单个Worker的内存大小，单位MB， 默认值2048。
void setMemoryForReduceTask(int mem)	设置Reducer任务中单个Worker的内存大小，单位MB， 默认值 2048。

- 通常情况下，GroupingColumns包含在KeySortColumns中，KeySortColumns和PartitionColumns要包含在Key中。
- 在Map端，Mapper输出的Record会根据设置的PartitionColumns计算哈希值，决定分配到哪个Reducer，会根据KeySortColumns对Record进行排序。
- 在Reduce端，输入Records，再按照KeySortColumns排序后，会根据GroupingColumns指定的列对输入的Records进行分组，即会顺序遍历输入的Records，把GroupingColumns所指定列相同的Records作为一次reduce函数调用的输入。

JobClient

主要函数接口。

主要接口	描述
static RunningJob runJob(JobConf job)	阻塞（同步）方式提交MapReduce作业后立即返回。
static RunningJob submitJob(JobConf job)	非阻塞（异步）方式提交MapReduce作业后立即返回。

RunningJob

主要函数接口。

主要接口	描述
String getInstanceID()	获取作业运行实例ID，用于查看运行日志和作业管理。
boolean isComplete()	查询作业是否结束。
boolean isSuccessful()	查询作业实例是否运行成功。
void waitForCompletion()	等待直至作业实例结束。一般用于异步方式提交的作业。
JobStatus getJobStatus()	查询作业实例运行状态。
void killJob()	结束此作业。
Counters getCounters()	获取Conter信息。

InputUtils

主要函数接口。

主要接口	描述
static void addTable(TableInfo table, JobConf conf)	添加表table到任务输入，可以被调用多次，新加入的表以append方式添加到输入队列中。
static void setTables(TableInfo [] tables, JobConf conf)	添加多张表到任务输入中。

OutputUtils

主要函数接口。

主要接口	描述
static void addTable(TableInfo table, JobConf conf)	添加表table到任务输出，可以被调用多次，新加入的表以append方式添加到输出队列中。
static void setTables(TableInfo[] tables, JobConf conf)	添加多张表到任务输出中。

Pipeline

Pipeline是MR2的主体类。可以通过Pipeline.builder构建一个Pipeline。Pipeline的主要接口如下。

```
    public Builder addMapper(Class<? extends Mapper> mapper)
    public Builder addMapper(Class<? extends Mapper> mapper,
           Column[] keySchema, Column[] valueSchema, String[] sortCols,
           SortOrder[] order, String[] partCols,
           Class<? extends Partitioner> theClass, String[] groupCols)
    public Builder addReducer(Class<? extends Reducer> reducer)
    public Builder addReducer(Class<? extends Reducer> reducer,
           Column[] keySchema, Column[] valueSchema, String[] sortCols,
           SortOrder[] order, String[] partCols,
           Class<? extends Partitioner> theClass, String[] groupCols)
    public Builder setOutputKeySchema(Column[] keySchema)
    public Builder setOutputValueSchema(Column[] valueSchema)
    public Builder setOutputKeySortColumns(String[] sortCols)
    public Builder setOutputKeySortOrder(SortOrder[] order)
    public Builder setPartitionColumns(String[] partCols)
    public Builder setPartitionerClass(Class<? extends Partitioner> theClass)
    public Builder setOutputGroupingColumns(String[] cols)
```

示例如下。

```
Job job = new Job();
Pipeline pipeline = Pipeline.builder()
 .addMapper(TokenizerMapper.class)
 .setOutputKeySchema(
     new Column[] { new Column("word", OdpsType.STRING) })
 .setOutputValueSchema(
     new Column[] { new Column("count", OdpsType.BIGINT) })
 .addReducer(SumReducer.class)
 .setOutputKeySchema(
     new Column[] { new Column("count", OdpsType.BIGINT) })
 .setOutputValueSchema(
     new Column[] { new Column("word", OdpsType.STRING),
     new Column("count", OdpsType.BIGINT) })
 .addReducer(IdentityReducer.class).createPipeline();
job.setPipeline(pipeline);
job.addInput(...)
job.addOutput(...)
job.submit();
```

如上所示，您可以在main函数中构建一个Map之后，连续接两个Reduce的MapReduce任务。如果您比较熟悉MapReduce的基础功能，即可轻松使用MR2。

> 说明
> - 建议您在使用MR2功能前，先了解MapReduce的基础用法。
> - JobConf仅能够配置Map后接单Reduce的MapReduce任务。

6.6.2. 兼容版本SDK概述

本文为您介绍MaxCompute兼容版本的MapReduce与Hadoop MapReduce兼容性。

MaxCompute兼容版本的MapReduce与Hadoop MapReduce兼容性的详细列表，如下表所示。

类型	接口	是否兼容
Mapper	void map(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context)	是
Mapper	void run(org.apache.hadoop.mapreduce.Mapper.Context context)	是
Mapper	void setup(org.apache.hadoop.mapreduce.Mapper.Context context)	是
Reducer	void cleanup(org.apache.hadoop.mapreduce.Reducer.Context context)	是
Reducer	void reduce(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Reducer.Context context)	是
Reducer	void run(org.apache.hadoop.mapreduce.Reducer.Context context)	是
Reducer	void setup(org.apache.hadoop.mapreduce.Reducer.Context context)	是
Partitioner	int getPartition(KEY key, VALUE value, int numPartitions)	是
MapContext（继承TaskInputOutputContext）	InputSplit getInputSplit()	否，抛异常
ReduceContext	nextKey()	是
ReduceContext	getValues()	是
TaskInputOutputContext	getCurrentKey()	是
TaskInputOutputContext	getCurrentValue()	是
TaskInputOutputContext	getOutputCommitter()	否，抛异常
TaskInputOutputContext	nextKeyValue()	是
TaskInputOutputContext	write(KEYOUT key, VALUEOUT value)	是
TaskAttemptContext	getCounter(Enum<?> counterName)	是
TaskAttemptContext	getCounter(String groupName, String counterName)	是

类型	接口	是否兼容
TaskAttemptContext	setStatus(String msg)	空实现
TaskAttemptContext	getStatus()	空实现
TaskAttemptContext	getTaskAttemptID()	否，抛异常
TaskAttemptContext	getProgress()	否，抛异常
TaskAttemptContext	progress()	是
Job	addArchiveToClassPath(Path archive)	否
Job	addCacheArchive(URI uri)	否
Job	addCacheFile(URI uri)	否
Job	addFileToClassPath(Path file)	否
Job	cleanupProgress()	否
Job	createSymlink()	否，抛异常
Job	failTask(TaskAttemptID taskId)	否
Job	getCompletionPollInterval(Configuration conf)	空实现
Job	getCounters()	是
Job	getFinishTime()	是
Job	getHistoryUrl()	是
Job	getInstance()	是
Job	getInstance(Cluster ignored)	是
Job	getInstance(Cluster ignored, Configuration conf)	是
Job	getInstance(Configuration conf)	是
Job	getInstance(Configuration conf, String jobName)	空实现
Job	getInstance(JobStatus status, Configuration conf)	否，抛异常
Job	getJobFile()	否，抛异常
Job	getJobName()	空实现
Job	getJobState()	否，抛异常
Job	getPriority()	否，抛异常
Job	getProgressPollInterval(Configuration conf)	空实现
Job	getReservationId()	否，抛异常
Job	getSchedulingInfo()	否，抛异常
Job	getStartTime()	是
Job	getStatus()	否，抛异常
Job	getTaskCompletionEvents(int startFrom)	否，抛异常
Job	getTaskCompletionEvents(int startFrom, int numEvents)	否，抛异常
Job	getTaskDiagnostics(TaskAttemptID taskid)	否，抛异常
Job	getTaskOutputFilter(Configuration conf)	否，抛异常
Job	getTaskReports(TaskType type)	否，抛异常
Job	getTrackingURL()	是

类型	接口	是否兼容
Job	isComplete()	是
Job	isRetired()	否，抛异常
Job	isSuccessful()	是
Job	isUber()	空实现
Job	killJob()	是
Job	killTask(TaskAttemptID taskId)	否
Job	mapProgress()	是
Job	monitorAndPrintJob()	是
Job	reduceProgress()	是
Job	setCacheArchives(URI[] archives)	否，抛异常
Job	setCacheFiles(URI[] files)	否，抛异常
Job	setCancelDelegationTokenUponJobCompletion(boolean value)	否，抛异常
Job	setCombinerClass(Class<? extends Reducer> cls)	是
Job	setCombinerKeyGroupingComparatorClass(Class<? extends RawComparator> cls)	是
Job	setGroupingComparatorClass(Class<? extends RawComparator> cls)	是
Job	setInputFormatClass(Class<? extends InputFormat> cls)	空实现
Job	setJar(String jar)	是
Job	setJarByClass(Class<?> cls)	是
Job	setJobName(String name)	空实现
Job	setJobSetupCleanupNeeded(boolean needed)	空实现
Job	setMapOutputKeyClass(Class<?> theClass)	是
Job	setMapOutputValueClass(Class<?> theClass)	是
Job	setMapperClass(Class<? extends Mapper> cls)	是
Job	setMapSpeculativeExecution(boolean speculativeExecution)	空实现
Job	setMaxMapAttempts(int n)	空实现
Job	setMaxReduceAttempts(int n)	空实现
Job	setNumReduceTasks(int tasks)	是
Job	setOutputFormatClass(Class<? extends OutputFormat> cls)	否，抛异常
Job	setOutputKeyClass(Class<?> theClass)	是
Job	setOutputValueClass(Class<?> theClass)	是
Job	setPartitionerClass(Class<? extends Partitioner> cls)	是
Job	setPriority(JobPriority priority)	否，抛异常
Job	setProfileEnabled(boolean newValue)	空实现
Job	setProfileParams(String value)	空实现
Job	setProfileTaskRange(boolean isMap, String newValue)	空实现
Job	setReducerClass(Class<? extends Reducer> cls)	是

类型	接口	是否兼容
Job	setReduceSpeculativeExecution(boolean speculativeExecution)	空实现
Job	setReservationId(ReservationId reservationId)	否，抛异常
Job	setSortComparatorClass(Class<? extends RawComparator> cls)	否，抛异常
Job	setSpeculativeExecution(boolean speculativeExecution)	是
Job	setTaskOutputFilter(Configuration conf, org.apache.hadoop.mapreduce.Job.TaskStatusFilter newValue)	否，抛异常
Job	setupProgress()	否，抛异常
Job	setUser(String user)	空实现
Job	setWorkingDirectory(Path dir)	空实现
Job	submit()	是
Job	toString()	否，抛异常
Job	waitForCompletion(boolean verbose)	是
Task Execution & Environment	mapreduce.map.java.opts	空实现
Task Execution & Environment	mapreduce.reduce.java.opts	空实现
Task Execution & Environment	mapreduce.map.memory.mb	空实现
Task Execution & Environment	mapreduce.reduce.memory.mb	空实现
Task Execution & Environment	mapreduce.task.io.sort.mb	空实现
Task Execution & Environment	mapreduce.map.sort.spill.percent	空实现
Task Execution & Environment	mapreduce.task.io.soft.factor	空实现
Task Execution & Environment	mapreduce.reduce.merge.inmem.thresholds	空实现
Task Execution & Environment	mapreduce.reduce.shuffle.merge.percent	空实现
Task Execution & Environment	mapreduce.reduce.shuffle.input.buffer.percent	空实现
Task Execution & Environment	mapreduce.reduce.input.buffer.percent	空实现
Task Execution & Environment	mapreduce.job.id	空实现
Task Execution & Environment	mapreduce.job.jar	空实现
Task Execution & Environment	mapreduce.job.local.dir	空实现
Task Execution & Environment	mapreduce.task.id	空实现
Task Execution & Environment	mapreduce.task.attempt.id	空实现
Task Execution & Environment	mapreduce.task.is.map	空实现
Task Execution & Environment	mapreduce.task.partition	空实现
Task Execution & Environment	mapreduce.map.input.file	空实现
Task Execution & Environment	mapreduce.map.input.start	空实现
Task Execution & Environment	mapreduce.map.input.length	空实现
Task Execution & Environment	mapreduce.task.output.dir	空实现
JobClient	cancelDelegationToken(Token <DelegationTokenIdentifier> token)	否，抛异常
JobClient	close()	空实现
JobClient	displayTasks(JobID jobId, String type, String state)	否，抛异常

类型	接口	是否兼容
JobClient	getAllJobs()	否，抛异常
JobClient	getCleanupTaskReports(JobID jobId)	否，抛异常
JobClient	getClusterStatus()	否，抛异常
JobClient	getClusterStatus(boolean detailed)	否，抛异常
JobClient	getDefaultMaps()	否，抛异常
JobClient	getDefaultReduces()	否，抛异常
JobClient	getDelegationToken(Text renewer)	否，抛异常
JobClient	getFs()	否，抛异常
JobClient	getJob(JobID jobid)	否，抛异常
JobClient	getJob(String jobid)	否，抛异常
JobClient	getJobsFromQueue(String queueName)	否，抛异常
JobClient	getMapTaskReports(JobID jobId)	否，抛异常
JobClient	getMapTaskReports(String jobId)	否，抛异常
JobClient	getQueueAclsForCurrentUser()	否，抛异常
JobClient	getQueueInfo(String queueName)	否，抛异常
JobClient	getQueues()	否，抛异常
JobClient	getReduceTaskReports(JobID jobId)	否，抛异常
JobClient	getReduceTaskReports(String jobId)	否，抛异常
JobClient	getSetupTaskReports(JobID jobId)	否，抛异常
JobClient	getStagingAreaDir()	否，抛异常
JobClient	getSystemDir()	否，抛异常
JobClient	getTaskOutputFilter()	否，抛异常
JobClient	getTaskOutputFilter(JobConf job)	否，抛异常
JobClient	init(JobConf conf)	否，抛异常
JobClient	isJobDirValid(Path jobDirPath, FileSystem fs)	否，抛异常
JobClient	jobsToComplete()	否，抛异常
JobClient	monitorAndPrintJob(JobConf conf, RunningJob job)	否，抛异常
JobClient	renewDelegationToken(Token<DelegationTokenIdentifier> token)	否，抛异常
JobClient	run(String[] argv)	否，抛异常
JobClient	runJob(JobConf job)	是
JobClient	setTaskOutputFilter(JobClient.TaskStatusFilter newValue)	否，抛异常
JobClient	setTaskOutputFilter(JobConf job, JobClient.TaskStatusFilter newValue)	否，抛异常
JobClient	submitJob(JobConf job)	是
JobClient	submitJob(String jobFile)	否，抛异常
JobConf	deleteLocalFiles()	否，抛异常
JobConf	deleteLocalFiles(String subdir)	否，抛异常

类型	接口	是否兼容
JobConf	normalizeMemoryConfigValue(long val)	空实现
JobConf	setCombinerClass(Class<? extends Reducer> theClass)	是
JobConf	setCompressMapOutput(boolean compress)	空实现
JobConf	setInputFormat(Class<? extends InputFormat> theClass)	否，抛异常
JobConf	setJar(String jar)	否，抛异常
JobConf	setJarByClass(Class cls)	否，抛异常
JobConf	setJobEndNotificationURI(String uri)	否，抛异常
JobConf	setJobName(String name)	空实现
JobConf	setJobPriority(JobPriority prio)	否，抛异常
JobConf	setKeepFailedTaskFiles(boolean keep)	否，抛异常
JobConf	setKeepTaskFilesPattern(String pattern)	否，抛异常
JobConf	setKeyFieldComparatorOptions(String keySpec)	否，抛异常
JobConf	setKeyFieldPartitionerOptions(String keySpec)	否，抛异常
JobConf	setMapDebugScript(String mDbgScript)	空实现
JobConf	setMapOutputCompressorClass(Class<? extends CompressionCodec> codecClass)	空实现
JobConf	setMapOutputKeyClass(Class<?> theClass)	是
JobConf	setMapOutputValueClass(Class<?> theClass)	是
JobConf	setMapperClass(Class<? extends Mapper> theClass)	是
JobConf	setMapRunnerClass(Class<? extends MapRunnable> theClass)	否，抛异常
JobConf	setMapSpeculativeExecution(boolean speculativeExecution)	空实现
JobConf	setMaxMapAttempts(int n)	空实现
JobConf	setMaxMapTaskFailuresPercent(int percent)	空实现
JobConf	setMaxPhysicalMemoryForTask(long mem)	空实现
JobConf	setMaxReduceAttempts(int n)	空实现
JobConf	setMaxReduceTaskFailuresPercent(int percent)	空实现
JobConf	setMaxTaskFailuresPerTracker(int noFailures)	空实现
JobConf	setMaxVirtualMemoryForTask(long vmem)	空实现
JobConf	setMemoryForMapTask(long mem)	是
JobConf	setMemoryForReduceTask(long mem)	是
JobConf	setNumMapTasks(int n)	是
JobConf	setNumReduceTasks(int n)	是
JobConf	setNumTasksToExecutePerJvm(int numTasks)	空实现
JobConf	setOutputCommitter(Class<? extends OutputCommitter> theClass)	否，抛异常
JobConf	setOutputFormat(Class<? extends OutputFormat> theClass)	空实现
JobConf	setOutputKeyClass(Class<?> theClass)	是
JobConf	setOutputKeyComparatorClass(Class<? extends RawComparator> theClass)	否，抛异常

类型	接口	是否兼容
JobConf	setOutputValueClass(Class<?> theClass)	是
JobConf	setOutputValueGroupingComparator(Class<? extends RawComparator> theClass)	否，抛异常
JobConf	setPartitionerClass(Class<? extends Partitioner> theClass)	是
JobConf	setProfileEnabled(boolean newValue)	空实现
JobConf	setProfileParams(String value)	空实现
JobConf	setProfileTaskRange(boolean isMap, String newValue)	空实现
JobConf	setQueueName(String queueName)	否，抛异常
JobConf	setReduceDebugScript(String rDbgScript)	空实现
JobConf	setReducerClass(Class<? extends Reducer> theClass)	是
JobConf	setReduceSpeculativeExecution(boolean speculativeExecution)	空实现
JobConf	setSessionId(String sessionId)	空实现
JobConf	setSpeculativeExecution(boolean speculativeExecution)	否，抛异常
JobConf	setUseNewMapper(boolean flag)	是
JobConf	setUseNewReducer(boolean flag)	是
JobConf	setUser(String user)	空实现
JobConf	setWorkingDirectory(Path dir)	空实现
FileInputFormat	不涉及	否，抛异常
TextInputFormat	不涉及	是
InputSplit	mapred.min.split.size.	否，抛异常
FileSplit	map.input.file	否，抛异常
RecordWriter	不涉及	否，抛异常
RecordReader	不涉及	否，抛异常
OutputFormat	不涉及	否，抛异常
OutputCommitter	abortJob(JobContext jobContext, int status)	否，抛异常
OutputCommitter	abortJob(JobContext context, JobStatus.State runState)	否，抛异常
OutputCommitter	abortTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	abortTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	cleanupJob(JobContext jobContext)	否，抛异常
OutputCommitter	cleanupJob(JobContext context)	否，抛异常
OutputCommitter	commitJob(JobContext jobContext)	否，抛异常
OutputCommitter	commitJob(JobContext context)	否，抛异常
OutputCommitter	commitTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	setupJob(JobContext jobContext)	否，抛异常
OutputCommitter	setupJob(JobContext jobContext)	否，抛异常

类型	接口	是否兼容
OutputCommitter	setupTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	setupTask(TaskAttemptContext taskContext)	否，抛异常
Counter	getDisplayName()	是
Counter	getName()	是
Counter	getValue()	是
Counter	increment(long incr)	是
Counter	setValue(long value)	是
Counter	setDisplayName(String displayName)	是
DistributedCache	CACHE_ARCHIVES	否，抛异常
DistributedCache	CACHE_ARCHIVES_SIZES	否，抛异常
DistributedCache	CACHE_ARCHIVES_TIMESTAMPS	否，抛异常
DistributedCache	CACHE_FILES	否，抛异常
DistributedCache	CACHE_FILES_SIZES	否，抛异常
DistributedCache	CACHE_FILES_TIMESTAMPS	否，抛异常
DistributedCache	CACHE_LOCALARCHIVES	否，抛异常
DistributedCache	CACHE_LOCALFILES	否，抛异常
DistributedCache	CACHE_SYMLINK	否，抛异常
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf)	否，抛异常
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf, FileSystem fs)	否，抛异常
DistributedCache	addCacheArchive(URI uri, Configuration conf)	否，抛异常
DistributedCache	addCacheFile(URI uri, Configuration conf)	否，抛异常
DistributedCache	addFileToClassPath(Path file, Configuration conf)	否，抛异常
DistributedCache	addFileToClassPath(Path file, Configuration conf, FileSystem fs)	否，抛异常
DistributedCache	addLocalArchives(Configuration conf, String str)	否，抛异常
DistributedCache	addLocalFiles(Configuration conf, String str)	否，抛异常
DistributedCache	checkURIs(URI[] uriFiles, URI[] uriArchives)	否，抛异常
DistributedCache	createAllSymlink(Configuration conf, File jobCacheDir, File workDir)	否，抛异常
DistributedCache	createSymlink(Configuration conf)	否，抛异常
DistributedCache	getArchiveClassPaths(Configuration conf)	否，抛异常
DistributedCache	getArchiveTimestamps(Configuration conf)	否，抛异常
DistributedCache	getCacheArchives(Configuration conf)	否，抛异常
DistributedCache	getCacheFiles(Configuration conf)	否，抛异常
DistributedCache	getFileClassPaths(Configuration conf)	否，抛异常
DistributedCache	getFileStatus(Configuration conf, URI cache)	否，抛异常
DistributedCache	getFileTimestamps(Configuration conf)	否，抛异常
DistributedCache	getLocalCacheArchives(Configuration conf)	否，抛异常

类型	接口	是否兼容
DistributedCache	getLocalCacheFiles(Configuration conf)	否，抛异常
DistributedCache	getSymlink(Configuration conf)	否，抛异常
DistributedCache	getTimestamp(Configuration conf, URI cache)	否，抛异常
DistributedCache	setArchiveTimestamps(Configuration conf, String timestamps)	否，抛异常
DistributedCache	setCacheArchives(URI[] archives, Configuration conf)	否，抛异常
DistributedCache	setCacheFiles(URI[] files, Configuration conf)	否，抛异常
DistributedCache	setFileTimestamps(Configuration conf, String timestamps)	否，抛异常
DistributedCache	setLocalArchives(Configuration conf, String str)	否，抛异常
DistributedCache	setLocalFiles(Configuration conf, String str)	否，抛异常
IsolationRunner	不涉及	否，抛异常
Profiling	不涉及	空实现
Debugging	不涉及	空实现
Data Compression	不涉及	是
Skipping Bad Records	不涉及	否，抛异常
Job Authorization	mapred.acls.enabled	否，抛异常
Job Authorization	mapreduce.job.acl-view-job	否，抛异常
Job Authorization	mapreduce.job.acl-modify-job	否，抛异常
Job Authorization	mapreduce.cluster.administrators	否，抛异常
Job Authorization	mapred.queue.queue-name.acl-administer-jobs	否，抛异常
MultipleInputs	不涉及	否，抛异常
Multi{anchor:_GoBack}pleOutputs	不涉及	是
org.apache.hadoop.mapreduce.lib.db	不涉及	否，抛异常
org.apache.hadoop.mapreduce.security	不涉及	否，抛异常
org.apache.hadoop.mapreduce.lib.jobcontrol	不涉及	否，抛异常
org.apache.hadoop.mapreduce.lib.chain	不涉及	否，抛异常
org.apache.hadoop.mapreduce.lib.db	不涉及	否，抛异常

6.6.3. Java沙箱

MaxCompute MapReduce及UDF程序在分布式环境中运行时，受到Java沙箱的限制（MapReduce作业的主程序，例如MapReduce Main则不受此限制）。

Java沙箱限制说明

- 不允许直接访问本地文件，只能通过MaxCompute MapReduce、Graph提供的接口间接访问。包括：
  - Resources选项指定的资源，包括文件、Jar包和资源表等。
  - 通过System.out和System.err输出日志信息。可以通过MaxCompute客户端的Log命令查看日志信息。
- 不允许直接访问分布式文件系统，只能通过MaxCompute MapReduce、Graph访问表的记录。
- 不允许JNI调用限制。
- 不允许创建Java线程，不允许启动子进程执行Linux命令。
- 不允许访问网络，包括获取本地IP地址等。
- Java反射限制：suppressAccessChecks权限被禁止，无法setAccessible某个私有的属性或方法，以达到读取私有属性或调用私有方法的目的。

访问本地文件限制

在代码中，直接使用下列方法访问本地文件会报 `access denied` 异常：

- java.io.File

```
public boolean delete()
public void deleteOnExit()
public boolean exists()
public boolean canRead()
public boolean isFile()
public boolean isDirectory()
public boolean isHidden()
public long lastModified()
public long length()
public String[] list()
public String[] list(FilenameFilter filter)
public File[] listFiles()
public File[] listFiles(FilenameFilter filter)
public File[] listFiles(FileFilter filter)
public boolean canWrite()
public boolean createNewFile()
public static File createTempFile(String prefix, String suffix)
public static File createTempFile(String prefix,  String suffix,File directory)
public boolean mkdir()
public boolean mkdirs()
public boolean renameTo(File dest)
public boolean setLastModified(long time)
public boolean setReadOnly()
```

- java.io.RandomAccessFile

```
RandomAccessFile(String name, String mode)
RandomAccessFile(File file, String mode)
```

- java.io.FileInputStream

```
FileInputStream(FileDescriptor fdObj)
FileInputStream(String name)
FileInputStream(File file)
```

- java.io.FileOutputStream

```
FileOutputStream(FileDescriptor fdObj)
FileOutputStream(File file)
FileOutputStream(String name)
FileOutputStream(String name, boolean append)
```

- java.lang.Class

```
public ProtectionDomain getProtectionDomain()
```

- java.lang.ClassLoader

```
ClassLoader()
ClassLoader(ClassLoader parent)
```

- java.lang.Runtime

```
public Process exec(String command)
public Process exec(String command, String envp[])
public Process exec(String cmdarray[])
public Process exec(String cmdarray[], String envp[])
public void exit(int status)
public static void runFinalizersOnExit(boolean value)
public void addShutdownHook(Thread hook)
public boolean removeShutdownHook(Thread hook)
public void load(String lib)
public void loadLibrary(String lib)
```

- java.lang.System

```
public static void exit(int status)
public static void runFinalizersOnExit(boolean value)
public static void load(String filename)
public static void loadLibrary( String libname)
public static Properties getProperties()
public static void setProperties(Properties props)
public static String getProperty(String key) // 只允许部分key可以访问
public static String getProperty(String key, String def) // 只允许部分key可以访问
public static String setProperty(String key, String value)
public static void setIn(InputStream in)
public static void setOut(PrintStream out)
public static void setErr(PrintStream err)
public static synchronized void setSecurityManager(SecurityManager s)
```

System.getProperty允许的key列表如下。

```
java.version
java.vendor
java.vendor.url
java.class.version
os.name
os.version
os.arch
file.separator
path.separator
line.separator
java.specification.version
java.specification.vendor
java.specification.name
java.vm.specification.version
java.vm.specification.vendor
java.vm.specification.name
java.vm.version
java.vm.vendor
java.vm.name
file.encoding
user.timezone
```

- java.lang.Thread

```
Thread()
Thread(Runnable target)
Thread(String name)
Thread(Runnable target, String name)
Thread(ThreadGroup group, ...)
public final void checkAccess()
public void interrupt()
public final void suspend()
public final void resume()
public final void setPriority (int newPriority)
public final void setName(String name)
public final void setDaemon(boolean on)
public final void stop()
public final synchronized void stop(Throwable obj)
public static int enumerate(Thread tarray[])
public void setContextClassLoader(ClassLoader cl)
```

- java.lang.ThreadGroup

```
ThreadGroup(String name)
ThreadGroup(ThreadGroup parent, String name)
public final void checkAccess()
public int enumerate(Thread list[])
public int enumerate(Thread list[], boolean recurse)
public int enumerate(ThreadGroup list[])
public int enumerate(ThreadGroup list[], boolean recurse)
public final ThreadGroup getParent()
public final void setDaemon(boolean daemon)
public final void setMaxPriority(int pri)
public final void suspend()
public final void resume()
public final void destroy()
public final void interrupt()
public final void stop()
```

- java.lang.reflect.AccessibleObject

```
public static void setAccessible(...)
public void setAccessible(...)
```

- java.net.InetAddress

```
public String getHostName()
public static InetAddress[] getAllByName(String host)
public static InetAddress getLocalHost()
```

- java.net.DatagramSocket

```
public InetAddress getLocalAddress()
```

- java.net.Socket

```
Socket(...)
```

- java.net.ServerSocket

```
ServerSocket(...)
public Socket accept()
protected final void implAccept(Socket s)
public static synchronized void setSocketFactory(...)
public static synchronized void setSocketImplFactory(...)
```

- java.net.DatagramSocket

```
DatagramSocket(...)
public synchronized void receive(DatagramPacket p)
```

- java.net.MulticastSocket

```
MulticastSocket(...)
```

- java.net.URL

```
URL(...)
public static synchronized void setURLStreamHandlerFactory(...)
java.net.URLConnection
public static synchronized void setContentHandlerFactory(...)
public static void setFileNameMap(FileNameMap map)
```

- java.net.HttpURLConnection

```
public static void setFollowRedirects(boolean set)
java.net.URLClassLoader
URLClassLoader(...)
```

- java.security.AccessControlContext

```
public AccessControlContext(AccessControlContext acc, DomainCombiner combiner)
public DomainCombiner getDomainCombiner()
```

6.7. MapReduce常见问题

本文为您介绍在使用MapReduce时的常见问题。

问题类别	常见问题

问题类别	常见问题
功能咨询	• MapReduce的输入源可以是视图吗？ • MapReduce的结果写入到表或分区时，会覆盖还是追加数据？ • MapReduce中是否可以调用Shell文件？ • reduce.setup能否读入输入表？ • Mapper是否支持同一表格多分区输入？ • Mapper可以直接从Record中读取分区数据段的信息吗？ • Label和分区是什么关系？ • MapReduce中是否可以只有Map？ • Mapper中输入表中的每条Record数据，是否可以按列名读取？ • write(Record key, Record value)与write(Record record)的区别是什么？ • MaxCompute MapReduce中，为什么要指定两个JAR，Libjars和Classpath？ • Hadoop MapReduce的源码，能否直接应用于MaxCompute MapReduce？ • MapReduce如何实现排序？ • MapReduce中的Backups是什么？
开发MapReduce	• 开发MapReduce时，如何在命令行传入多个资源？ • 在Main方法中如何判断一个表为空表？ • 在MaxCompute MapReduce中，如何设置Java代码，才能打印出日志？ • 结果表中会保留两个MapReduce计算之后的重复数据吗？ • Hadoop中可以选择多个Node进行分布处理（一个Node表示一台机器），那么MaxCompute MapReduce进行分布处理时，该如何设置Node？ • 不使用Combiner时输出正常，使用Combiner后Reduce没有输入，是什么原因？ • 在MapOnly中，为什么程序没有指定输出Table的Schema格式？ • 如何在本地调用MaxCompute服务器运行MapReduce任务？
常见报错	• 执行MaxCompute MapReduce时，报错BufferOverflowException，如何解决？ • 执行MaxCompute MapReduce时，报错Resource not found，如何解决？ • 执行MaxCompute MapReduce时，报错Class Not Found，如何解决？ • 执行MaxCompute MapReduce时，报错ODPS-0010000，如何解决？ • 执行MaxCompute MapReduce时，报错Table not found，如何解决？ • 执行MaxCompute MapReduce时，报错ODPS-0123144，如何解决？ • 执行MaxCompute MapReduce时，报错java.security.AccessControlException，如何解决？ • 执行MaxCompute MapReduce时，报错 java.io.IOException，如何解决？ • 执行MaxCompute MapReduce时，报错Exceed maximum read times，如何解决？ • 执行MaxCompute MapReduce时，Reduce还没开始执行第一句，便报错为内存溢出，如何解决？ • 执行MaxCompute MapReduce时，报错内存溢出，如何解决？ • 执行MaxCompute MapReduce时，开启了600个Reducer加载一个配置文件，但是配置文件很小且报错java.lang.OutOfMemoryError，如何解决？ • 执行MaxCompute MapReduce时，报错ODPS-0420095，如何解决？ • 在MapReduce中使用资源时，报错文件太多，如何解决？ • 在MapReduce程序中使用第三方类，打了一个Assembly的JAR包，运行时报错找不到类，如何解决？ • 使用开源兼容MapReduce在MaxCompute上运行时，报错显示下标越界，如何解决？

MapReduce的输入源可以是视图吗？

不可以，只能是表。

MapReduce的结果写入到表或分区时，会覆盖还是追加数据？

会覆盖掉原有的表数据或分区数据。

MapReduce中是否可以调用Shell文件？

不可以，受Java沙箱限制。更多Java沙箱信息，请参见Java沙箱。

reduce.setup能否读入输入表？

不能读入输入表，只能读入缓存表。

Mapper是否支持同一表格多分区输入？

Mapper支持同一表格多分区输入，同一张表的多个分区可以理解为独立的两张表。

Mapper可以直接从Record中读取分区数据段的信息吗？

Mapper无法从Record中获取分区字段信息，但是可以使用如下代码。PartitionSpec为分区信息。

```
PartitionSpec ps = context.getInputTableInfo().getPartitionSpec();
String area = ps.get("area");
```

Label和分区是什么关系？

Label是给不同的输出处打上的一个标签，当输出的时候可以识别是从哪里输出的。

MapReduce中是否可以只有Map？

可以，MapReduce支持Map-Only。如果是Map-Only作业，需要显式指定Reducer数量为0，即 `job.setNumReduceTasks(0)` 。

Mapper中输入表中的每条Record数据，是否可以按列名读取？

可以。输入表中的每条Record数据不仅可以按序号 `record.get(i)` 读取，还可以按照列名来获取Record，例如 `record.get("size")` 。

`write(Record key, Record value)` 与 `write(Record record)` 的区别是什么？

- `write(Record key, Record value)` ：输出中间结果，例如 `key.set("id", v1), value.set("size", v2)` 。Map产出的中间结果，需要网络传输给Reduce，没有关联表进行类型推断，所以必须声明字段类型用于序列化。输出的字段类型是MaxCompute的字段类型。

  ```
  job.setMapOutputKeySchema(SchemaUtils.fromString("id:string"));
  job.setMapOutputValueSchema(SchemaUtils.fromString("size:bigint"));
  ```

- `write(Record record)` ：输出结果到最终表，有表进行类型推断，没必要声明字段类型。

MaxCompute MapReduce中，为什么要指定两个JAR，Libjars和Classpath？

本地客户端会执行一些作业配置等操作，涉及远程执行。因此本地有一个执行器，远程有一个执行器。

远程执行器执行加载远程的Classpath，即 `-libjars mapreduce-examples.jar` 。本地执行器加载本地Classpath，所以也要指定 `-classpath lib/mapreduce-examples.jar` 。

Hadoop MapReduce的源码，能否直接应用于MaxCompute MapReduce？

不能。MaxCompute MapReduce与Hadoop MapReduce API有一定差别，但整体风格一致。您需要修改Hadoop的源码，并在MaxCompute MapReduce SDK下编译通过后，才能在MaxCompute上运行。

MapReduce如何实现排序？

排序代码如下。

```
//设置排序的字段（这里根据i1,i2两个字段进行排序）。
job.setOutputKeySortColumns(new String[] { "i1", "i2" });
//设置字段的排序顺序（这里设置i1为正序排序，i2为逆序排序）。
job.setOutputKeySortOrder(new SortOrder[] { SortOrder.ASC, SortOrder.DESC });
```

setOutputKeySortOrder方法的用法，如下所示。

```
public void setOutputKeySortOrder(JobConf.SortOrder[] order)
功能：设置Key排序列的顺序。
参数：Order表示排序列的顺序。可选值为ASC（升序）、DESC（降序）。
```

MapReduce中的Backups是什么？

Backups就是一种加速调优。MaxCompute会查看您的任务，如果存在一些任务的工作量特别大，便为其开启备份任务，两个任务执行一样的数据，结果用最先执行结束的任务，因此存在Backups。但是如果任务非常大，Backups也无法起作用，因为原本的任务和Backups都没有执行完成。

开发MapReduce时，如何在命令行传入多个资源？

用逗号（,）分隔符进行分隔，例如输入命令 `jar -resource resource1,resource2,..` 即可。

在Main方法中如何判断一个表为空表？

您可以使用如下方式判断一个表是否为空表。

```
Odps odps=SessionState.get().getOdps();
Table table=odps.tables().get('tableName');
RecordReader recordReader=table.read(1);
if(recordReader.read()==null){
//TO DO
```

在MaxCompute MapReduce中，如何设置Java代码，才能打印出日志？

推荐如下方法：

- 可以在代码中用 `System.out.println` 打印日志，对应日志输出位置是在Logview的stdout中。
- 使用出现异常时，客户端会返回异常信息，不需要打印日志信息。
- 使用common logging，日志输出到stderr中，可以在Logview的stderr看到。

结果表中会保留两个MapReduce计算之后的重复数据吗？

会。查询数据时，会得到两条一样的数据。

Hadoop中可以选择多个Node进行分布处理（一个Node表示一台机器），那么MaxCompute MapReduce进行分布处理时，该如何设置Node？

您不需要自行搭建分配Node，这正是MaxCompute的优点之一。

当运行MapReduce时，MaxCompute底层会根据算法决定使用的数据分片。

不使用Combiner时输出正常，使用Combiner后Reduce没有输入，是什么原因？

Reduce输出的单个Record和Map输出的Key-Value对不一致，导致产生上述情况。

在MapOnly中，为什么程序没有指定输出Table的Schema格式？

输出表的Schema是需要预先创建好的，在 `create table` 的时候指定。MapOnly程序内部不用指定Schema，可以直接输出。

如何在本地调用MaxCompute服务器运行MapReduce任务？

正常情况下，MaxCompute的JAR包需要在命令行工具上执行 `jar` 命令来执行，具体语法请参见MapReduce作业提交。

您也可以通过模拟的方式整合到自己的工程中，方法如下：

1. 设置包依赖。

   除了基本的SDK，还需要其他的几个依赖包，您可在客户端工具的lib文件夹中找到。lib文件夹中也包含了SDK的JAR包，建议您在导入包的时候把最新的客户端工具lib文件夹中的JAR包全部导进来。

2. 上传JAR包。

   将在本地测试通过的MapReduce程序，打成JAR包并上传，假设此JAR包的名称是*mr.jar*。更多上传资源操作，请参见资源操作。

3. 设置运行方式。

   配置Jobconf，配置示例如下。

   ```
   **配置MaxCompute连接信息。
   Account account = new AliyunAccount(accessid, accesskey);
   Odps odps = new Odps(account);
   odps.setEndpoint(endpoint);
   odps.setDefaultProject(project);
   **获取session。
   SessionState ss = SessionState.get();
   ss.setOdps(odps);
   ss.setLocalRun(false);  **设置false，表示在服务器上跑。如果要在本地调试，直接设置为true。
   **正常的设置jobconf等代码。
   Job job = new Job();
   String resource = "mr.jar";
   job.setResources(resource); **这一步类似Jar命令的jar -resources mr.jar。
   **后面的代码为正常的MapReduce代码规则。
   job.setMapperClass(XXXMapper.class);
   job.setReducerClass(XXXReducer.class);
   ```

   配置完成之后，您可以直接运行MapReduce任务。

执行MaxCompute MapReduce时，报错BufferOverflowException，如何解决？

- 问题现象

  执行MaxCompute MapReduce时，返回如下报错。

  ```
  FAILED: ODPS-0123131:User defined function exception - Traceback:
      java.nio.BufferOverflowException
      at java.nio.DirectByteBuffer.put(Unknown Source)
      at com.aliyun.odps.udf.impl.batch.TextBinary.put(TextBinary.java:35)
  ```

- 产生原因

  一次写入的数据太大，导致Buffer溢出。

- 解决措施

  MaxCompute的单个字段可以写入的数据类型限制如下。

```
String      8MB
Bigint      -9223372036854775807 ~ 9223372036854775807
Boolean     True/False
Double      -1.0 10308 ~ 1.0 10308
Date        0001-01-01 00:00:00 ~ 9999-12-31 23:59:59
```

执行MaxCompute MapReduce时，报错Resource not found，如何解决？

提交作业时，要用 `-resources` 指定需要的资源，多个资源用逗号（,）分隔。

执行MaxCompute MapReduce时，报错Class Not Found，如何解决？

执行MapReduce时，以下两种情况会报此错误：

- `classpath` 参数的类名写错，要写上完整的包名。
- 打包JAR的时候出错，注意打包时要把SRC中的源码都选上。

执行MaxCompute MapReduce时，报错ODPS-0010000，如何解决？

- 问题现象：

  执行MaxCompute MapReduce时，返回如下报错。

  ```
  ODPS-0010000: System internal error - get input pangu dir meta fail.
  ```

- 产生原因

  出现上述报错，是因为您还未创建分区，或是因为分区数据未到位，您就开始使用此分区。

- 解决措施

  先创建分区，然后再运行MapReduce任务。

执行MaxCompute MapReduce时，报错Table not found，如何解决？

- 问题现象：

  执行MaxCompute MapReduce时，返回报错如下。

  ```
  Exception in thread "main" com.aliyun.odps.OdpsException: Table not found: project_name.table_name.
  ```

- 产生原因

  目标项目不正确或目标表不存在。

- 解决措施

  MapReduce接口的Table Info Builder是ProjectName和TableName，两个接口分别设置为Project名称和Table名称。

执行MaxCompute MapReduce时，报错ODPS-0123144，如何解决？

- 问题现象：

  执行MaxCompute MapReduce时，返回报错如下。

  ```
  FAILED: ODPS-0123144: Fuxi job failed - WorkerRestar
  ```

- 产生原因：

  出现上述报错，是因为集群的备节点在计算的过程中出现超时的情况，导致主节点认为备节点故障，所以报错。超时10分钟，即会报错，暂时不支持用户自行配置。

- 解决措施

  这个报错比较常见的原因是Reduce里做了大循环，例如存在长尾数据或者笛卡尔积。您需要尽量减少这种大循环的情况。

执行MaxCompute MapReduce时，报错java.security.AccessControlException，如何解决？

- 问题现象

  执行MaxCompute MapReduce时，返回报错如下。

  ```
  FAILED: ODPS-0123131:User defined function exception - Traceback:
  java.lang.ExceptionInInitializerError
   ...
  Caused by: java.security.AccessControlException: access denied ("java.lang.RuntimePermission" "getProtectionDomain")
    at java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
  ```

- 产生原因

  出现上述报错，是因为您的代码违反了沙箱限制，详情请参见Java沙箱。

- 解决措施

您需要访问外部的资源解决此报错。但MaxCompute目前不支持访问外部资源，请将外部的处理逻辑和涉及的数据存储在MaxCompute上进行访问。您需要读取一些配置文件，详情请参见使用资源示例。

执行MaxCompute MapReduce时，报错 java.io.IOException，如何解决？

- 问题现象：

  执行MaxCompute MapReduce时，返回报错如下。

  ```
  Exception in thread "main" java.io.IOException: ODPS-0740001: Too many local-run maps: 101, must be <= 100(specified by local-run parameter 'odps.mapred.local.map.max.tasks')
  ```

- 产生原因

  local-run maps默认是100，需要调整。

- 解决措施

  您可以添加 `-Dodps.mapred.local.map.max.tasks=200` 配置。

执行MaxCompute MapReduce时，报错Exceed maximum read times，如何解决？

- 问题现象：

  执行MaxCompute MapReduce时，返回报错如下。

  ```
  ODPS-0730001: Exceed maximum read times per resource
  ```

- 产生原因：

  资源文件读取的次数太多。

- 解决措施

  请检查代码中读取对应资源的代码逻辑。一般情况下，资源读取在setup中执行一次即可，不要在Map或Reduce阶段中多次读取。

执行MaxCompute MapReduce时，Reduce还没开始执行第一句，便报错为内存溢出，如何解决？

- 产生原因

  某些数据特别大，下载到内存便溢出了。

- 解决措施

  把Combiner去掉或者在Combiner中限制Size `set odps.mapred.map.min.split.size=512;` 。

执行MaxCompute MapReduce时，报错内存溢出，如何解决？

内存溢出通常是由于内存不足导致的，可以通过调整JVM内存参数（ `odps.stage.mapper.jvm.mem`和`odps.stage.reducer.jvm.mem` ）解决。例如 `set odps.stage.mapper.jvm.mem = 2048` 将内存调整为2 GB。

执行MaxCompute MapReduce时，开启了600个Reducer加载一个配置文件，但是配置文件很小且报错java.lang.OutOfMemoryError，如何解决？

- 问题现象

  执行MaxCompute MapReduce时，返回报错如下。

  ```
  java.lang.OutOfMemoryError: Java heap space
  ```

- 产生原因

  MapReduce使用限制导致，请参见使用限制。

- 解决措施

  请参见原生SDK概述进行配置。

执行MaxCompute MapReduce时，报错ODPS-0420095，如何解决？

- 问题现象

  执行MaxCompute MapReduce时，返回报错如下。

  ```
  Exception in thread "main" java.io.IOException: com.aliyun.odps.OdpsException: ODPS-0420095: Access Denied - The task is not in release range: LOT
  ```

- 产生原因

  项目为MaxCompute开发者版资源下的项目，仅支持MaxCompute SQL（支持使用UDF）、PyODPS作业任务，暂不支持MapReduce、Spark等其它任务。

- 解决措施

  升级项目规格，请参见转换计费方式。

在MapReduce中使用资源时，报错文件太多，如何解决？

- 问题现象

  在MapReduce中使用资源时，返回报错如下。

  ```
  Caused by: com.aliyun.odps.OdpsException: java.io.FileNotFoundException: temp/mr_XXXXXX/resource/meta.user.group.config (Too many open files)
  ```

- 产生原因

  单个作业引用的资源数量不能超过256个，否则报错。Table和Archive按照一个单位计算。更多限制请参见使用限制。

- 解决措施

  调整引用的资源数量。

在MapReduce程序中使用第三方类，打了一个Assembly的JAR包，运行时报错找不到类，如何解决？

MaxCompute MapReduce在分布式环境中运行时受到Java沙箱的限制（MapReduce作业的主程序则不受此限制），具体限制请参见Java沙箱。

如果您只是需要处理Json，建议您直接使用Gson，同时无需打包带Gson的Class。Java开源组件里提供了很多字符串转日期的方法，例如SimpleDateFormat等。

使用开源兼容MapReduce在MaxCompute上运行时，报错显示下标越界，如何解决？

建议您使用MaxCompue的MapReduce接口进行代码编写。同时，非必须场景下建议使用Spark替代MapReduce。

7.数据科学（Mars）

7.1. 版本更新记录

本文为您介绍Mars近期版本的更新说明，基于此您可以了解Mars对应版本中的新增功能、增强功能等内容。

Mars近期版本的更新说明如下，详细信息请单击对应版本链接获取。

版本	变更类型	描述
v0.6.3	新功能	DataFrame： • `md.Index` 添加更多功能，详情请参见Add more functionalities for `md.Index`（#1864）。 • 支持 `{DataFrame,Series}.rename_axis` ，详情请参见Implements `{DataFrame,Series}.rename_axis`（#1870）。
	增强功能	• 允许内部序列化使用JSON，详情请参见Allow internal serialization to use JSON（#1882）。 • 优化 `{md.read_csv()` 、 `md.read_parquet()}.head()` 的性能，详情请参见Optimize performance of {md.read_csv(), md.read_parquet()}.head()（#1883）。 • 优化 `df.sort_values().head()` 的性能，详情请参见Optimize performance of `df.sort_values().head()`（#1888）。 • 支持对使用 `groupby().agg()` 的数据源进行列裁剪，详情请参见Support column pruning for groupby().agg() on data sources（#1889）。 • 提升 `named_{dataframe, series, tensor}` ，支持获取更多元数据，详情请参见Improve `named_{dataframe, series, tensor}` that it's able to get more meta（#1897）。
v0.6.2	新功能	• DataFrame 支持 `df.groupby().head()` ，详情请参见Implements head() on groupby objects（#1851）。 • Learn 支持 `mars.learn.preprocessing.{MinMaxScaler, minmax_scale}` ，详情请参见Implements `mars.learn.preprocessing.{MinMaxScaler, minmax_scale}`（#1858）。
	增强功能	• 优化Proxima的 `recall_by_id` 计算方式，详情请参见Improve proxima `recall_by_id` computation method（#1805）。 • 修复Tensor和DataFrame读写vineyard，详情请参见Revise to/from vineyard, of Tensor and DataFrame（#1806）。 • 为 `read_parquet` 添加类似 `read_csv` 的内存预估机制，详情请参见Add memory estimation for `read_parquet` as well as `read_csv`（#1815）。 • 支持在Lambda中使用复合Agg函数，详情请参见Support using compound agg function in lambda（#1819）。 • 为 `reset_index` 添加incremental_index字段，默认值为False，详情请参见Add `incremental_index` argument to `reset_index` which by default is False（#1842）。 • DataFrame和Series的 `to_pandas` 方法内部使用批方式获取数据，详情请参见Support `to_pandas` in a batch way for DataFrame and Series（#1859）。 • 支持在Kubernetes中指定记忆量表，详情请参见Support specifying memory scale in kubernetes（#1861）。
v0.6.1	新功能	• Tensor 当Tensor转稀疏时，支持缺失参数，以支持指定非0的缺失值，详情请参见Support `missing` argument for `tensor.tosparse()` and `fill_value` argument for `sparse_tensor.todense()`（#1802）。 • DataFrame ◦ 添加Replace接口，详情请参见Implements {DataFrame,Series}.replace（#1765）。 ◦ 添加Cartesian_chunk接口，可以对两个DataFrame执行笛卡尔积，并对每个Chunk应用一个函数。该接口是对Pandas API的扩展，详情请参见Add `{DataFrame, Series}.cartesian_chunk` support（#1777）。 ◦ 聚合和分组聚合对字符串连接操作做了优化，详情请参见Integrate `str.cat` into reduction and groupby-aggregation（#1781）。 ◦ 聚合函数支持了Level参数，详情请参见Implements reduction with `level` argument（#1784）。
	兼容性	聚合和分组聚合在该版本中已重写，老版本的Mars客户端连接到新版本使用这些功能可能会出错。

版本	变更类型	描述
v0.6.0	新功能	DataFrame： • DataFrame初始化时支持 `num_partitions` 参数，详情请参见Support `num_partitions` argument for DataFrame initializers（#1733）。 • 添加对命名聚合的支持，详情请参见Add support for named aggregations（#1748）。
	增强功能	`groupby.agg()` 使用ReductionCompiler统一，详情请参见Unify groupby.agg() using ReductionCompiler（#1739）。
	缺陷修复	• 修复md.read_csv：当names和usecols都指定时可能会报错，详情请参见Fix `md.read_csv` when names and usecols specified（#1738）。 • 支持字符串dtype以减少Tensor并平衡PSRS，详情请参见Support string dtype for tensor reductions（#1746）。 • 在DataFrame上修复XGBoost和LightGBM，详情请参见Fix xgboost and lightgbm on DataFrames（#1751）。 • 修复在分布式模式下重复执行相同代码的问题，详情请参见Fix repeated execution of same code in distributed mode（#1753）。 • 支持将DataFrame的某一列设置一个标量Tensor，详情请参见Support setting scalar which is a tensor for DataFrame（#1758）。

7.2. 概述

本文为您介绍Mars的功能、与PyODPS DataFrame的区别和使用场景。

使用场景

Mars与PyODPS DataFrame使用场景如下：

- Mars
  - 经常使用PyODPS DataFrame的to_pandas()方法，将PyODPS DataFrame转成Pandas DataFrame的场景。
  - 熟悉Pandas接口，但不愿意学习PyODPS DataFrame接口的场景。
  - 使用索引的场景。
  - 创建DataFrame后需要保证数据顺序的场景。

    Mars DataFrame通过iloc等方法可以获取某个偏移的数据。例如，`df.iloc[10]`可以获取第10行数据。Mars DataFrame也支持需要保证数据顺序才可以使用的特性接口`df.shift()`和`df.ffill()`。
  - 需要并行和分布化Numpy或Scikit-learn，以及支持分布式运行TensorFlow、PyTorch和XGBoost的场景。
  - 数据量在TB级别以下的场景。
- PyODPS DataFrame
  - 使用MaxCompute调度作业的场景。PyODPS DataFrame会将DataFrame编译成MaxCompute SQL。如果需要通过MaxCompute调度作业，建议您使用PyODPS DataFrame。
  - 稳定性要求较高的作业场景。PyODPS DataFrame会将作业编译至MaxCompute执行，由于MaxCompute相当稳定，而Mars相对比较新，如果对稳定性有很高要求，建议您使用PyODPS DataFrame。
  - 数据量在TB级别以上的场景，建议您使用PyODPS DataFrame。

与PyODPS DataFrame的区别

- API
  - Mars

    Mars DataFrame完全兼容Pandas。Mars Tensor兼容Numpy。Mars Learn兼容Scikit-learn。
  - PyODPS

    只有DataFrame接口，和Pandas的接口差异较大。
- 索引

- Mars

  Mars DataFrame支持索引操作，包含行和列索引。示例代码如下。

```
In [1]: import mars.dataframe as md
In [5]: import mars.tensor as mt
In [7]: df = md.DataFrame(mt.random.rand(10, 3), index=md.date_range('2020-5-1', periods=10))
In [9]: df.loc['2020-5'].execute()
Out[9]:
                   0         1         2
2020-05-01  0.061912  0.507101  0.372242
2020-05-02  0.833663  0.818519  0.943887
2020-05-03  0.579214  0.573056  0.319786
2020-05-04  0.476143  0.245831  0.434038
2020-05-05  0.444866  0.465851  0.445263
2020-05-06  0.654311  0.972639  0.443985
2020-05-07  0.276574  0.096421  0.264799
2020-05-08  0.106188  0.921479  0.202131
2020-05-09  0.281736  0.465473  0.003585
2020-05-10  0.400000  0.451150  0.956905
```

- PyODPS

  不支持索引操作。

- 数据顺序
  - Mars

    Mars DataFrame创建后，会保证数据顺序，支持时序操作（ `shift` ）、向前（ `ffill` ）、向后（ `bfill` ）和填空值操作。

```
In [3]: df = md.DataFrame([[1, None], [None, 1]])
In [4]: df.execute()
Out[4]:
     0    1
0  1.0  NaN
1  NaN  1.0
In [5]: df.ffill().execute() #空值用上一行的值。
Out[5]:
     0    1
0  1.0  NaN
1  1.0  1.0
```

  - PyODPS

    PyODPS使用MaxCompute计算和存储数据，而MaxCompute并不保证数据顺序，因此PyODPS不保证数据顺序，不支持时序操作。

- 执行层
  - Mars

    Mars包含客户端和分布式执行层。您可以通过调用 `o.create_mars_cluster` ，在MaxCompute内部创建Mars集群，并将计算作业直接提交至Mars集群，调度费用极小。在数据规模较小时，Mars更有优势。

  - PyODPS

    PyODPS是一个客户端，不包含任何服务端部分。执行PyODPS DataFrame时，系统会将计算作业编译至MaxCompute SQL。因此，PyODPS DataFrame支持的操作，取决于MaxCompute SQL。此外，您每次调用 `execute` 方法时，会提交一次MaxCompute作业，需要在集群内调度作业。

使用说明

Mars是一个基于张量的统一分布式计算框架。Mars能利用并行和分布式技术，为Python数据科学栈加速，包括Numpy、Pandas和Scikit-learn。

Mars常用接口如下：

- Mars Tensor接口

  和Numpy保持一致，且支持大规模高维数组。示例代码如下。

```
import mars.tensor as mt
a = mt.random.rand(10000, 50)
b = mt.random.rand(50, 5000)
a.dot(b).execute()
```

- Mars DataFrame接口

  和Pandas保持一致，且支持大规模数据处理和分析。示例代码如下。

```
import mars.dataframe as md
ratings = md.read_csv('Downloads/ml-20m/ratings.csv')
movies = md.read_csv('Downloads/ml-20m/movies.csv')
movie_rating = ratings.groupby('movieId', as_index=False).agg({'rating': 'mean'})
result = movie_rating.merge(movies[['movieId', 'title']], on='movieId')
result.sort_values(by='rating', ascending=False).execute()
```

- Mars Learn接口

  和Scikit-learn保持一致。示例代码如下。Mars Learn可以集成TensorFlow、PyTorch和XGBoost。

```
import mars.dataframe as md
from mars.learn.neighbors import NearestNeighbors
df = md.read_csv('data.csv')
nn = NearestNeighbors(n_neighbors=10)
nn.fit(df)
neighbors = nn.kneighbors(df).fetch()
```

参考文档

- Mars开源地址
- Mars文档
- Mars团队专栏

技术支持

如果您在使用Mars过程中遇到问题，请加入技术支持钉钉群（11701793）咨询。

7.3. 准备工作

本文为您介绍如何准备Mars运行环境。

在MaxCompute上运行Mars服务，您需要准备相应的运行环境。MaxCompute支持如下两种运行环境：

- DataWorks
  i. DataWorks中的PyODPS 3节点，已经包含了PyODPS和Mars功能。

     您可以在新建的PyODPS 3节点里执行如下命令检查PyODPS和Mars版本，确保满足要求。

```
from odps import __version__ as odps_version
from mars import __version__ as mars_version
print(odps_version)
print(mars_version)
```

     odps_version为PyODPS版本，要求PyODPS为0.9.3.1以上版本。mars_version为Mars版本，要求Mars为0.4.4以上版本。

  ii. 准备PyODPS入口。

     DataWorks的PyODPS 3节点会自动创建PyODPS入口，因此您无需手动创建，即可直接使用。

- 其他环境
  i. 您需要先安装pip，安装完成后，在系统的命令行执行窗口（例如Windows的cmd窗口），通过pip命令安装PyODPS和Mars。命令示例如下。
     - （可选）确保pip版本为最新版本。

```
pip install -U pip
```

     - 安装PyODPS最新版本。https://mirrors.aliyun.com/pypi/simple/为阿里云镜像，可以加快下载包的速度。

```
pip install pyodps -i https://mirrors.aliyun.com/pypi/simple/
```

     - 安装Mars最新版本。

```
pip install pymars -i https://mirrors.aliyun.com/pypi/simple/
```

     - 安装ProtoBuf最新版本。

```
pip install protobuf -i https://mirrors.aliyun.com/pypi/simple/
```

     - （可选）安装PyArrow最新版本，可以加速运行。

```
pip install pyarrow -i https://mirrors.aliyun.com/pypi/simple/
```

     > **说明** 更多pip安装详情请参见安装pip。

  ii. 准备PyODPS入口。

     您需要通过AccessKey ID和AccessKey Secret等参数创建PyODPS入口。详情请参见初始化PyODPS入口。

7.4. 使用说明

本文为您介绍Mars集群上的操作、MaxCompute读取表以及获取Mars UI地址等更多操作。

如果您需要了解开发Mars作业的详细操作指导，详情请参见Mars。

Mars集群操作

- 创建Mars集群

  执行如下命令创建Mars集群。Mars集群的创建过程较慢，请您耐心等待。

  ```
  from odps import options
  options.verbose = True
  #在DataWorks PyODPS3中已经设置上述命令，因此使用PyODPS 3节点无需执行前两行代码。
  client = o.create_mars_cluster(5, 4, 16, min_worker_num=3)
  ```

  参数说明如下：

  - 5：指定该集群的Worker数量。此集群Worker数量为5。
  - 4：每个Worker的核数。此处每个Worker为4核。
  - 16：每个Worker的内存大小。此处每个Worker内存为16 GB。

    > **说明**
    > - 申请的单个Worker内存需要大于1 GB，CPU核数和内存的最佳比例为1:4，例如单Worker配置为4核、16 GB。
    > - 新建Worker个数不应超过30个，否则会对镜像服务器造成压力。如果需要超过30个Worker，请提工单申请。

  - min_worker_num：指当启动3个Worker后，即可返回Client对象，无需等待全部Worker都启动后再返回。

  当指定 `options.verbose=True` 时，新建Mars集群过程中会打印MaxCompute Instance的Logview、Mars UI以及Notebook地址。Mars UI可以用于连接Mars集群，也可以用于查看集群、任务状态。

- 提交作业

  创建Mars集群时会设置默认Session，通过 `.execute()` 方法执行任务，任务会被自动提交至集群。

  ```
  import mars.dataframe as md
  import mars.tensor as mt
  md.DataFrame(mt.random.rand(10, 3)).execute()  #使用execute方法自动提交任务至创建的集群。
  ```

- 停止并释放集群

  当Mars集群超过3天会被自动释放。不再需要使用Mars集群时，您可以通过调用 `client.stop_server()` 手动释放。

  ```
  client.stop_server()
  ```

MaxCompute表读写

Mars可以直接读写MaxCompute表：

- 读表

  通过 `o.to_mars_dataframe` 读取MaxCompute表，并返回 Mars DataFrame。

  ```
  In [1]: df = o.to_mars_dataframe('test_mars')
  In [2]: df.head(6).execute()
  Out[2]:
         col1  col2
  0         0     0
  1         0     1
  2         0     2
  3         1     0
  4         1     1
  5         1     2
  ```

- 写表

  通过 `o.persist_mars_dataframe(df, 'table_name')` 将Mars DataFrame保存为MaxCompute表。

```
In [3]: df = o.to_mars_dataframe('test_mars')
In [4]: df2 = df + 1
In [5]: o.persist_mars_dataframe(df2, 'test_mars_persist')  # 保存Mars DataFrame。
In [6]: o.get_table('test_mars_persist').to_df().head(6)  # 通过PyODPS DataFrame查看数据。
       col1  col2
0         1     1
1         1     2
2         1     3
3         2     1
4         2     2
5         2     3
```

- 使用Mars集群自带的Jupyter NoteBook

> **说明** `create_mars_cluster` 指定了 `with_notebook=True` 才可以使用Jupyter NoteBook。

新建一个NoteBook会自动设置Session，提交任务到集群，因此在这个NoteBook内不需要显示创建Session。

```
import mars.dataframe as md
md.DataFrame(mt.random.rand(10, 3)).sum().execute() # 在NoteBook中运行，execute方法自动提交到当前集群。因此在NoteBook中不需要显示创建Session。
```

> **说明**
> - 此Notebook不会自动保存，建议您根据需要自行保存。
> - 您也可以使用自己的NoteBook连接到集群，详情请参见使用已经创建的Mars集群。

其它操作

- 使用已经创建的Mars集群
  - 通过Instance ID重建Mars集群的Client。

    ```
    client = o.create_mars_cluster(instance_id=**instance-id**)
    ```

  - 如果只是想使用Mars，您可以指定Mars UI的地址，使用Mars Session连接。

    ```
    from mars.session import new_session
    new_session('**Mars UI address**').as_default() #设置为默认Session。
    ```

- 获取Mars UI地址

  Mars集群创建时指定了 `options.verbose=True` ，集群会自动打印Mars UI地址。您也可以通过 `client.endpoint` 获取Mars UI。

  ```
  print(client.endpoint)
  ```

- 获取Logview地址

  创建集群创建时指定了 `options.verbose=True` ，集群会自动打印Logview。您也可以通过 `client.get_logview_address()` 获取Logview地址。

  ```
  print(client.get_logview_address())
  ```

- 获取Jupyter Notebook地址

  Mars集群创建时指定了 `options.verbose=True` ，集群会自动打印Jupyter Notebook地址。您也可以通过 `client.get_notebook_endpoint()` 获取Jupyter Notebook地址。

  ```
  print(client.get_notebook_endpoint())
  ```

8.Spark

8.1. 概述

MaxCompute Spark是MaxCompute提供的兼容开源Spark的计算服务。它在统一的计算资源和数据集权限体系之上，提供Spark计算框架，支持您以熟悉的开发使用方式提交运行Spark作业，满足更丰富的数据处理分析需求。

使用限制

- MaxCompute Spark支持如下场景：
  - 离线计算场景，例如GraphX、Mllib、RDD、Spark-SQL、PySpark等。
  - 读写MaxCompute Table。
  - 引用MaxCompute中的文件资源。
  - 读写VPC环境下的服务。例如，RDS、Redis、HBase、ECS上部署的服务等。
  - 读写OSS非结构化存储。
- MaxCompute Spark暂不支持如下场景：
  - 交互式和流计算类需求，例如Spark-Shell、Spark-SQL-Shell、PySpark-Shell、Spark Streaming等。
  - 不支持访问MaxCompute外部表、内建函数和自定义函数（MaxCompute UDF）。
  - 不支持在使用按量计费开发者版资源的项目中执行Spark作业。按量计费开发者版仅支持MaxCompute SQL（支持使用UDF）、PyODPS作业。
- 不支持Checkpoint功能。

关键特性

- 支持原生多版本Spark作业。

  MaxCompute支持社区原生Spark、完全兼容Spark的API，同时支持多个Spark版本同时运行。MaxCompute Spark提供原生的Spark WebUI供您查看。
- 统一的计算资源。

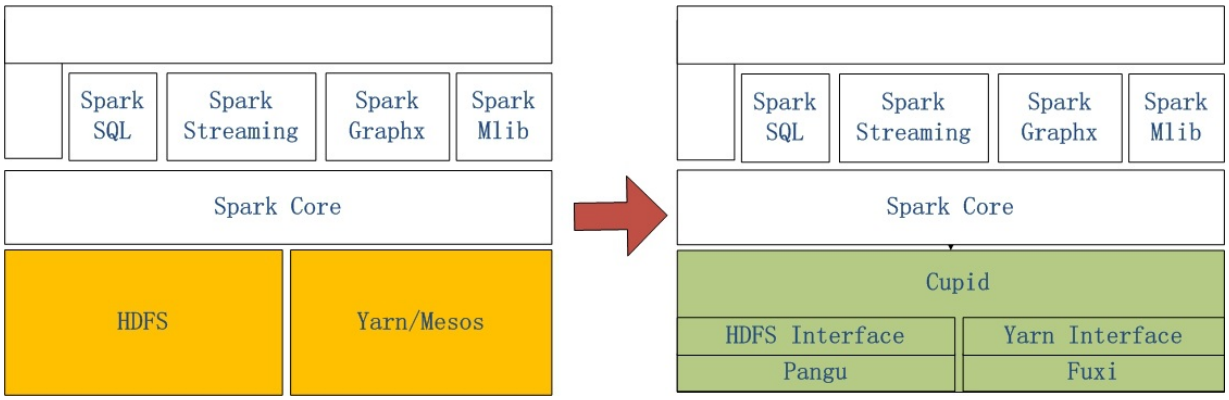
  MaxCompute Spark类似MaxCompute SQL、MapReduce等作业类型，运行在MaxCompute项目统一开通的计算资源中。
- 统一的数据和权限管理。

  完全遵循MaxCompute项目的权限体系，在访问用户权限范围内安全地查询数据。
- 与开源系统相同的使用体验。

  MaxCompute Spark与社区开源Spark保持相同的体验（例如开源应用的UI界面、在线交互等），完全符合Spark用户使用习惯。开源应用的调试过程中需要使用开源UI，MaxCompute Spark提供原生的开源实时UI和查询历史日志的功能。其中，对于部分开源应用还支持交互式体验，在后台引擎运行后即可进行实时交互。

系统结构

MaxCompute Spark是阿里云通过Spark on MaxCompute的解决方案，让原生Spark能够在MaxCompute中运行。



左侧是原生Spark的架构图，右侧的Spark on MaxCompute运行在阿里云自研的Cupid平台之上，该平台可以原生支持开源社区Yarn所支持的计算框架。

8.2. 搭建开发环境

本文为您介绍如何搭建MaxCompute Spark开发环境。

前提条件

搭建Spark开发环境前，请确保您已经在操作系统中安装如下软件：

- JDK 1.8

  以Linux操作系统为例，安装命令示例如下。JDK名称请以实际为准。具体的JDK名称可通过执行 `yum -y list java*` 获取。

  ```
  yum install -y java-1.8.0-openjdk-devel.x86_64
  ```

- Python 2.7

  以Linux操作系统为例，安装命令示例如下。Python包名称请以实际为准。

  ```
  # 获取Python包。
  wget https://www.python.org/ftp/python/2.7.10/Python-2.7.10.tgz
  # 解压缩Python包。
  tar -zxvf Python-2.7.10.tgz
  # 切换到解压后的目录，指定安装路径。
  cd Python-2.7.10
  ./configure --prefix=/usr/local/python2
  # 编译并安装Python。
  make
  make install
  ```

- Maven

  以Linux操作系统为例，安装命令示例如下。Maven包路径请以实际为准。

  ```
  # 获取Maven包。
  wget https://dlcdn.apache.org/maven/maven-3/3.8.4/binaries/apache-maven-3.8.4-bin.tar.gz
  # 解压缩Maven包。
  tar -zxvf apache-maven-3.8.4-bin.tar.gz
  ```

- Git

  以Linux操作系统为例，安装命令示例如下。

  ```
  # 获取Git包。
  wget https://github.com/git/git/archive/v2.17.0.tar.gz
  # 解压缩Git包。
  tar -zxvf v2.17.0.tar.gz
  # 安装编译源码所需依赖。
  yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel gcc perl-ExtUtils-MakeMaker
  # 切换到解压后的目。
  cd git-2.17.0
  # 编译。
  make prefix=/usr/local/git all
  # 安装Git至/usr/local/git路径。
  make prefix=/usr/local/git install
  ```

下载MaxCompute Spark客户端包并上传至操作系统

MaxCompute Spark发布包集成了MaxCompute认证功能。作为客户端工具，它通过Spark-Submit方式提交作业到MaxCompute项目中运行。MaxCompute提供了面向Spark1.x、Spark2.x和Spark3.x发布包，下载路径如下：

- Spark-1.6.3：适用于Spark1.x应用的开发。
- Spark-2.3.0：适用于Spark2.x应用的开发。
- Spark-2.4.5：适用于Spark2.x应用的开发。使用Spark-2.4.5的注意事项请参见Spark 2.4.5使用注意事项。
- Spark-3.1.1：适用于Spark3.x应用的开发。使用Spark-3.1.1的注意事项请参见Spark 3.1.1使用注意事项。

将MaxCompute Spark客户端包上传至Linux操作系统中，并解压。您可以进入Spark客户端包所在目录，执行如下命令解压包。

```
tar -xzvf spark-2.3.0-odps0.33.0.tar.gz
```

设置环境变量

您需要在操作系统的命令行执行窗口配置如下环境变量信息，以Linux系统为例，配置方法及信息如下。

- 配置Java环境变量。

- 获取Java安装路径。命令示例如下。

```
# 如果通过yum方式安装，默认安装在usr目录下，您可以按照如下命令查找。如果您自定义了安装路径，请以实际路径为准。
whereis java
ls -lrt /usr/bin/java
ls -lrt /etc/alternatives/java
# 返回信息如下。/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.1.al7.x86_64即为安装路径。
/etc/alternatives/java -> /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.1.al7.x86_64/jre/bin/java
```

```
[root@ ~]# whereis java
java: /usr/bin/java /usr/lib/java /etc/java /usr/share/java /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.3
22.b06-1.1.al7.x86 64/bin/java /usr/share/man/man1/java.1.gz
[root@ ~]# ls -lrt /usr/bin/java
lrwxrwxrwx 1 root root 22 Feb 10 16:49 /usr/bin/java -> /etc/alternatives/java
[root@ ~]# ls -lrt /etc/alternatives/java
lrwxrwxrwx 1 root root 73 Feb 10 16:49 /etc/alternatives/java -> /usr/lib/jvm/java-1.8.0-openjdk-1.
8.0.322.b06-1.1.al7.x86_64/jre/bin/java
```

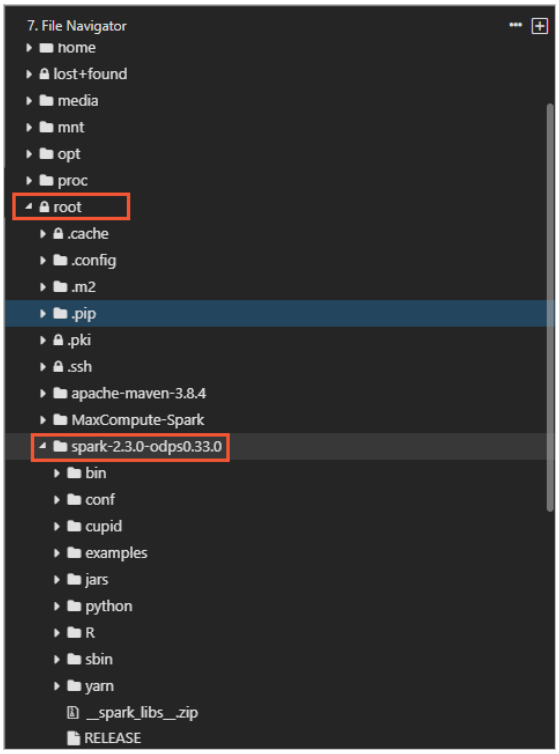
- 编辑Java环境变量信息。命令示例如下。

```
# 编辑环境变量配置文件。
vim /etc/profile
# 按下i进入编辑状态后，在配置文件末尾添加环境变量信息。
# JAVA_HOME需要修改为实际Java的安装路径。
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.1.al7.x86_64
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
# 按ESC退出编辑，按:wq退出配置文件。
# 执行如下命令使修改生效。
source /etc/profile
# 确认Java已配置成功。
java -version
# 返回结果示例如下。
openjdk version "1.8.0_322"
OpenJDK Runtime Environment (build 1.8.0_322-b06)
OpenJDK 64-Bit Server VM (build 25.322-b06, mixed mode)
```

```
>_ 2. root@ :~ ×
else
    export HISTCONTROL=ignoredups
fi

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL

# By default, we want umask to get set. This sets it for login shell
# Current threshold for system reserved uid/gids is 200
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`/usr/bin/id -gn`" = "`/usr/bin/id -un`" ]; then
    umask 002
else
    umask 022
fi

for i in /etc/profile.d/*.sh /etc/profile.d/sh.local ; do
    if [ -r "$i" ]; then
        if [ "${-#*i}" != "$-" ]; then
            . "$i"
        else
            . "$i" >/dev/null
        fi
    fi
done

unset i
unset -f pathmunge
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.1.al7.x86_64
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
export SPARK_HOME=/root/spark-2.3.0-odps0.33.0
export PATH=$SPARK_HOME/bin:$PATH
export PATH=/usr/bin/python/bin/:$PATH
export MAVEN_HOME=/root/apache-maven-3.8.4
export PATH=$MAVEN_HOME/bin:$PATH
export LC_ALL=en_US.UTF-8
export PATH=$PATH:/usr/local/git/bin
"/etc/profile" 86L, 2256C                                    86,1          Bot
```

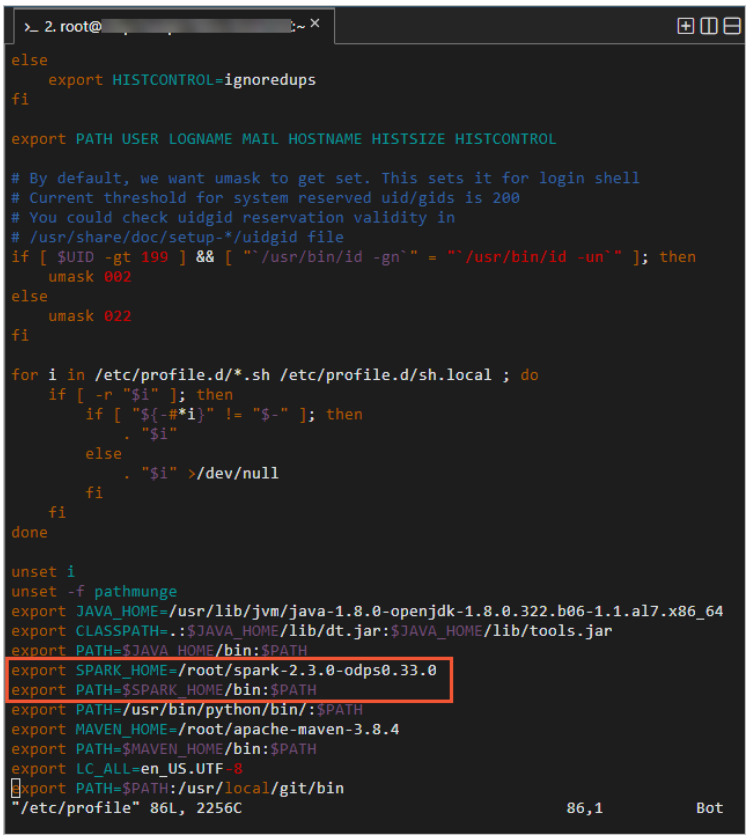
- 配置Spark环境变量。

- 获取Spark客户端包解压后的路径。图示如下，表明包所在路径为 `/root/spark-2.3.0-odps0.33.0` 。请以实际解压路径及名称为准。



- 编辑Spark环境变量信息。命令示例如下。

```
# 编辑环境变量配置文件。
vim /etc/profile
# 按下i进入编辑状态后，在配置文件末尾添加环境变量信息。
# SPARK_HOME需要修改为实际解压后的Spark客户端包所在路径。
export SPARK_HOME=/root/spark-2.3.0-odps0.33.0
export PATH=$SPARK_HOME/bin:$PATH
# 按ESC退出编辑，按:wq退出配置文件。
# 执行如下命令使修改生效。
source /etc/profile
```



- 配置Python环境变量。

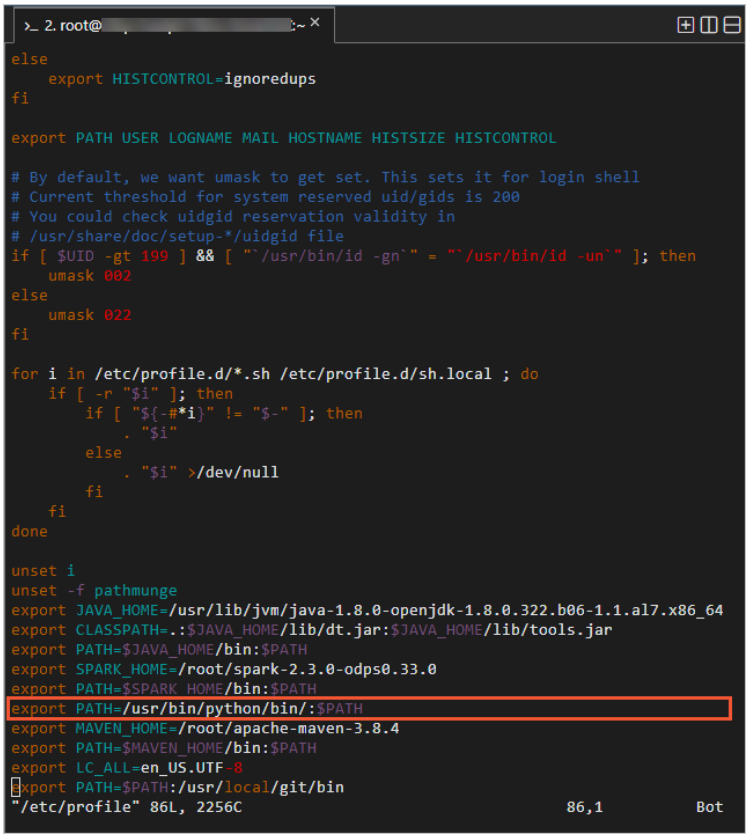
  使用PySpark的用户，需要配置该信息。

  - 获取Python安装路径。命令示例如下。

```
[root@ ~]# whereis python
python: /usr/bin/python2.7 /usr/bin/python3.6m /usr/bin/python3.6 /usr/bin/python /usr/lib/python2.7 /usr/lib/python3.6 /usr/lib64/python2.7 /usr/lib64/python3.6 /etc/python /usr/include/python2.7 /usr/include/python3.6m /usr/share/man/man1/python.1.gz
```

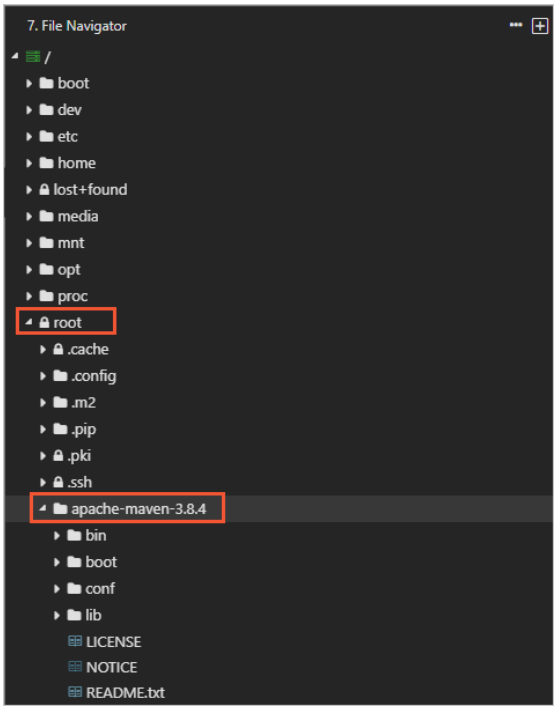
- 编辑Python环境变量信息。命令示例如下。

```
# 编辑环境变量配置文件。
vim /etc/profile
# 按下i进入编辑状态后，在配置文件末尾添加环境变量信息。
# PATH需要修改为Python的实际安装路径。
export PATH=/usr/bin/python/bin/:$PATH
# 按ESC退出编辑，按:wq退出配置文件。
# 执行如下命令使修改生效。
source /etc/profile
# 确认Python已配置成功。
python --version
# 返回结果示例如下。
Python 2.7.5
```



- 配置Maven环境变量。

- 获取Maven包解压后的路径。图示如下，表明包所在路径为 `/root/apache-maven-3.8.4` 。请以实际解压路径及名称为准。



- 编辑Maven环境变量信息。命令示例如下。

```
# 编辑环境变量配置文件。
vim /etc/profile
# 按下i进入编辑状态后，在配置文件末尾添加环境变量信息。
# MAVEN_HOME需要修改为实际解压后的Maven包所在路径。
export MAVEN_HOME=/root/apache-maven-3.8.4
export PATH=$MAVEN_HOME/bin:$PATH
# 按ESC退出编辑，按:wq退出配置文件。
# 执行如下命令使修改生效。
source /etc/profile
# 确认Maven已配置成功。
mvn -version
# 返回结果示例如下。
Apache Maven 3.8.4 (9b656c72d54e5bacbed989b64718c159fe39b537)
Maven home: /root/apache-maven-3.8.4
Java version: 1.8.0_322, vendor: Red Hat, Inc., runtime: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.1.al7.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.19.91-25.1.al7.x86_64", arch: "amd64", family: "unix"
```

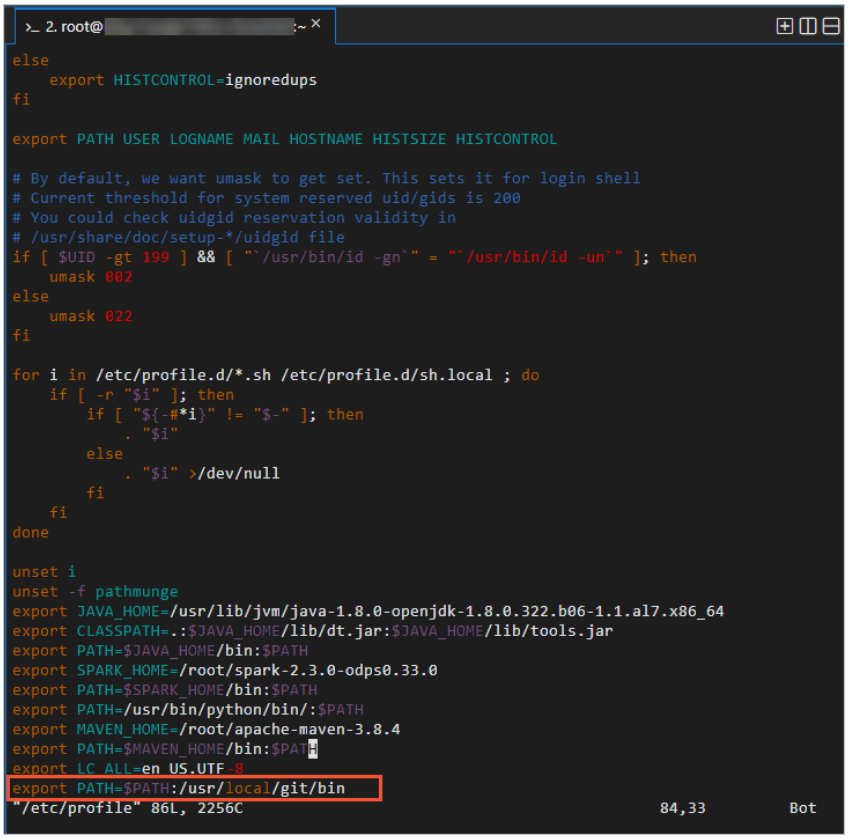
- 配置Git环境变量。
  - 获取Git安装路径。命令示例如下。

```
whereis git
```

```
[root@ ~]# whereis git
git: /usr/bin/git /usr/local/git /usr/share/man/man1/git.1.gz
```

- 编辑Git环境变量信息。命令示例如下。

```
# 编辑环境变量配置文件。
vim /etc/profile
# 按下i进入编辑状态后，在配置文件末尾添加环境变量信息。# PATH需要修改为Git的实际安装路径。
export PATH=/usr/local/git/bin/:$PATH
# 按ESC退出编辑，按:wq退出配置文件。
# 执行如下命令使修改生效。
source /etc/profile
# 确认Git已配置成功。
git --version
# 返回结果示例如下。
git version 2.24.4
```



配置spark-defaults.conf

第一次使用MaxCompute Spark客户端时，请在Spark客户端包的解压路径下，将 `conf` 文件夹下的*spark-defaults.conf.template*文件重命名为*spark-defaults.conf*后再进行相关配置。如果没有对文件进行重命名，将会导致配置无法生效。命令示例如下。

```
# 切换至Spark客户端包的解压路径，并进入conf文件夹。请以实际路径为准。
cd /root/spark-2.3.0-odps0.33.0/conf
# 修改文件名。
mv spark-defaults.conf.template spark-defaults.conf
# 编辑spark-defaults.conf。
vim spark-defaults.conf
# 按下i进入编辑状态后，在配置文件末尾添加如下配置信息。
spark.hadoop.odps.project.name = <MaxCompute_project_name>
spark.hadoop.odps.access.id = <AccessKey_id>
spark.hadoop.odps.access.key = <AccessKey_secret>
spark.hadoop.odps.end.point = <Endpoint>   # Spark客户端连接访问MaxCompute项目的Endpoint，您可以根据自己情况进行修改。详情请参见Endpoint。
spark.hadoop.odps.runtime.end.point = <VPC_endpoint>  # Spark运行环境Endpoint，所在Region的MaxCompute VPC网络的Endpoint。您可以根据自己情况进行修改。
# spark 2.3.0请将spark.sql.catalogImplementation设置为odps，spark 2.4.5请将spark.sql.catalogImplementation设置为hive。
spark.sql.catalogImplementation={odps|hive}
# 如下参数配置保持不变
spark.hadoop.odps.task.major.version = cupid_v2
spark.hadoop.odps.cupid.container.image.enable = true
spark.hadoop.odps.cupid.container.vm.engine.type = hyper
spark.hadoop.odps.cupid.webproxy.endpoint = http://service.cn.maxcompute.aliyun-inc.com/api
spark.hadoop.odps.moye.trackurl.host = http://jobview.odps.aliyun.com
```

- MaxCompute_project_name：待访问MaxCompute项目的名称。

  此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
- AccessKey_id：具备目标MaxCompute项目访问权限的AccessKey ID。

  您可以进入AccessKey管理页面获取AccessKey ID。
- AccessKey_secret：AccessKey ID对应的AccessKey Secret。

  您可以进入AccessKey管理页面获取AccessKey Secret。
- Endpoint：MaxCompute项目所属区域的外网Endpoint。

  各地域的外网Endpoint信息，请参见各地域Endpoint对照表（外网连接方式）。
- VPC_endpoint：MaxCompute项目所属区域的VPC网络的Endpoint。

  各地域的VPC网络Endpoint信息，请参见各地域Endpoint对照表（阿里云VPC网络连接方式）。

特殊场景和功能，需要开启一些其他的配置参数，详情请参见Spark配置详解。

准备项目工程

MaxCompute Spark提供了项目工程模版，建议您下载模版复制后直接在模版里开发。

> 注意 模版工程里的关于spark依赖的scope为provided，请不要更改，否则提交的作业无法正常运行。

以Linux操作系统为例，准备项目工程命令示例如下：

- 下载Spark-1.x模板并编译

```
git clone https://github.com/aliyun/MaxCompute-Spark.git
cd MaxCompute-Spark/spark-1.x
mvn clean package
```

- 下载Spark-2.x 模板并编译

```
git clone https://github.com/aliyun/MaxCompute-Spark.git
cd MaxCompute-Spark/spark-2.x
mvn clean package
```

- 下载Spark-3.x 模板并编译

```
git clone https://github.com/aliyun/MaxCompute-Spark.git
cd MaxCompute-Spark/spark-3.x
mvn clean package
```

上述命令执行完毕后，如果显示创建失败，说明环境配置有误，请按照上述配置指导仔细检查并修正环境配置信息。

配置依赖说明

在准备的MaxCompute Spark项目下，配置依赖信息。命令示例如下。

- 配置访问MaxCompute表所需的依赖。

- 使用Spark-1.x模板场景

```
# 进入spark-1.x文件夹。
cd MaxCompute-Spark/spark-1.x
# 编辑Pom文件，添加odps-spark-datasource依赖。
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-spark-datasource_2.10</artifactId>
  <version>3.3.8-public</version>
</dependency>
```

- 使用Spark-2.x模板场景

```
# 进入spark-2.x文件夹。
cd MaxCompute-Spark/spark-2.x
# 编辑Pom文件，添加odps-spark-datasource依赖。
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-spark-datasource_2.11</artifactId>
    <version>3.3.8-public</version>
</dependency>
```

- 配置访问OSS所需的依赖。

  如果作业需要访问OSS，直接添加以下依赖即可。

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>hadoop-fs-oss</artifactId>
    <version>3.3.8-public</version>
</dependency>
```

更多Spark-1.x、Spark-2.x以及Spark-3.x的依赖配置信息，请参见Spark-1.x pom文件、Spark-2.x pom文件和Spark-3.x pom文件。

引用外部文件

用户在开发过程中涉及到如下场景时，需要引用外部文件：

- 作业需要读取一些配置文件。
- 作业需要额外的资源包或第三方库。例如JAR包、Python库。

在实际操作中，您需要先上传文件后才可以引用文件，上传文件方式有以下两种，任选其中一种即可：

- 方式一：通过Spark参数上传文件

  MaxCompute Spark支持Spark社区版原生的 `--jars` 、 `--py-files` 、 `--files` 、 `--archives` 参数，您可以在提交作业时，通过这些参数上传文件，这些文件在作业运行时会被上传到用户的工作目录下。

  - 通过MaxCompute Spark客户端，使用Spark-Submit方式上传文件。

    > **说明**
    > - `--jars` ：会将配置的JAR包上传至Driver和Executor的当前工作目录，多个文件用英文逗号（,）分隔。这些JAR包都会加入Driver和Executor的Classpath。在Spark作业中直接通过 `"./your_jar_name"` 即可引用，与社区版Spark行为相同。
    > - `--files` 、 `--py-files` ：会将配置的普通文件或Python文件上传至Driver和Executor的当前工作目录，多个文件用英文逗号（,）分隔。在Spark作业中直接通过 `"./your_file_name"` 即可引用，与社区版Spark行为相同。
    > - `--archives` ：与社区版Spark行为略有不同，多个文件用英文逗号（,）分隔，配置方式为 `xxx#yyy` ，会将配置的归档文件（例如.zip）解压到Driver和Executor的当前工作目录的子目录中。例如当配置方式为 `xx.zip#yy` 时，应以 `"./yy/xx/"` 引用到归档文件中的内容；当配置方式为 `xx.zip` 时，应以 `"./xx.zip/xx/"` 引用到归档文件中的内容。如果一定要将归档内容直接解压到当前目录，即直接引用 `"./xxx/"` ，请使用下文中的 `spark.hadoop.odps.cupid.resources` 参数进行配置。

  - 通过DataWorks，添加作业需要的资源，操作详情请参见创建MaxCompute资源。

    > **说明** DataWorks中上传资源限制最大为50 MB，如果需要使用更大的资源，您需要通过MaxCompute客户端将资源上传为MaxCompute资源，并将资源添加至数据开发面板。更多MaxCompute资源信息，请参见MaxCompute资源。

- 方式二：通过MaxCompute资源上传文件

  MaxCompute Spark提供 `spark.hadoop.odps.cupid.resources` 参数，可以直接引用MaxCompute中的资源，这些资源在作业运行时会被上传到用户的工作目录下。使用方式如下：

  i. 通过MaxCompute客户端将文件上传至MaxCompute项目。单个文件最大支持500 MB。

  ii. 在Spark作业配置中添加 `spark.hadoop.odps.cupid.resources` 参数，指定Spark作业运行所需要的MaxCompute资源。格式为 `<projectname>.<resourcename>` ，如果需要引用多个文件，需要用英文逗号（,）分隔。配置示例如下：

```
spark.hadoop.odps.cupid.resources=public.python-python-2.7-ucs4.zip,public.myjar.jar
```

指定的资源将被下载到Driver和Executor的当前工作目录，资源下载到工作目录后默认名称是 `<projectname>.<resourcename>` 。

此外，您可以在配置时通过 `<projectname>.<resourcename>:<newresourcename>` 方式重命名资源名称。配置示例如下：

```
spark.hadoop.odps.cupid.resources=public.myjar.jar:myjar.jar
```

> **注意** 该配置项必须在spark-defaults.conf或DataWorks的配置项中进行配置才能生效，不能写在代码中。

通过上述两种方式将文件上传后，即可在代码中引用文件，文件读取示例如下：

```
val targetFile = "文件名"
val file = Source.fromFile(targetFile)
for (line <- file.getLines)
    println(line)
file.close
```

SparkPi冒烟测试

完成以上的工作之后，执行冒烟测试，验证MaxCompute Spark是否可以端到端连通。以Spark-2.x为例，您可以提交一个SparkPi验证功能是否正常，提交命令如下。

```
# /path/to/MaxCompute-Spark请指向正确的编译出来后的应用程序的Jar包。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.SparkPi \
/path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
# 当看到以下日志表明冒烟作业成功。
19/06/11 11:57:30 INFO Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: 11.222.166.90
         ApplicationMaster RPC port: 38965
         queue: queue
         start time: 1560225401092
         final status: SUCCEEDED
```

IDEA本地执行注意事项

通常，本地调试成功后会在集群上执行代码。但是Spark可以支持在IDEA里以Local模式直接运行代码，运行时请注意以下几点：

- 代码需要手动设置spark.master。

```
val spark = SparkSession
      .builder()
      .appName("SparkPi")
      .config("spark.master", "local[4]") // 需要设置spark.master为local[N]才能直接运行，N为并发数。
      .getOrCreate()
```

- 在IDEA里手动添加MaxCompute Spark客户端的相关依赖。

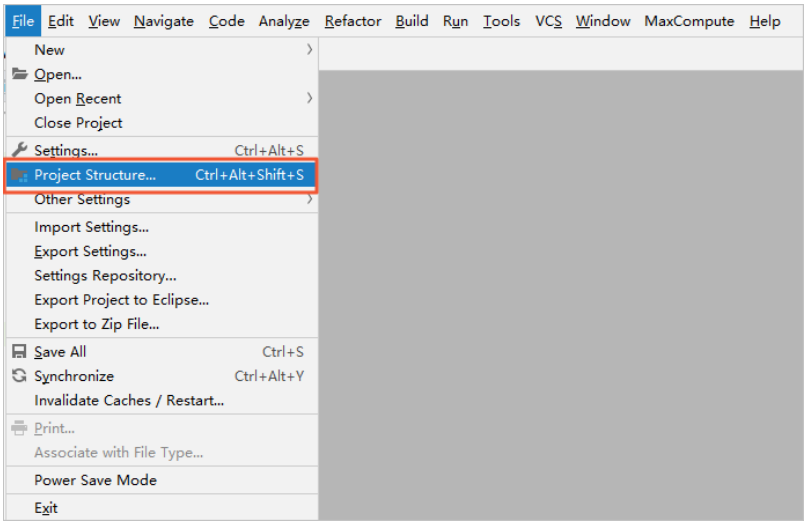
```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
```

*pom.xml*中设置要求scope为provided，所以运行时会出现NoClassDefFoundError报错。

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/spark/sql/SparkSession$
    at com.aliyun.odps.spark.examples.SparkPi$.main(SparkPi.scala:27)
    at com.aliyun.odps.spark.examples.Spa。r。kPi.main(SparkPi.scala)
Caused by: java.lang.ClassNotFoundException: org.apache.spark.sql.SparkSession$
    at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    ... 2 more
```

您可以按照以下方式手动将MaxCompute Spark下的Jars目录加入IDEA模板工程项目中，既可以保持 `scope=provided` ，又能在IDEA里直接运行不报错：

i. 在IDEA中单击顶部菜单栏上的**File**，选中**Project Structure...**。



ii. 在**Project Structure**页面，单击左侧导航栏上的**Modules**。选择资源包，并单击资源包的**Dependencies**页签。

iii. 在资源包的**Dependencies**页签下，单击左下角的**+**，选择**JARs or directories...**添加MaxCompute Spark下的Jars目录。

- Local不能直接引用*spark-defaults.conf*里的配置，需要手动指定相关配置。

  通过Spark-Submit方式提交作业，系统会读取*spark-defaults.conf*文件中的配置。通过Local模式提交作业，需要手动在代码里指定相关配置。例如，在Local模式下如果要通过Spark-Sql读取MaxCompute的表，配置如下。

```
val spark = SparkSession
      .builder()
      .appName("SparkPi")
      .config("spark.master", "local[4]") // 需设置spark.master为local[N]才能直接运行，N为并发数。
      .config("spark.hadoop.odps.project.name", "****")
      .config("spark.hadoop.odps.access.id", "****")
      .config("spark.hadoop.odps.access.key", "****")
      .config("spark.hadoop.odps.end.point", "http://service.cn.maxcompute.aliyun.com/api")
      .config("spark.sql.catalogImplementation", "odps")
      .getOrCreate()
```

Spark 2.4.5使用注意事项

- 使用Spark 2.4.5提交作业
  - 直接使用Yarn-cluster模式在本地提交任务。详情请参见Cluster模式。
  - 在DataWorks中配置参数 `spark.hadoop.odps.spark.version=spark-2.4.5-odps0.33.0` 。当前DataWorks独享资源组尚未升级到Spark 2.4.5，用户可以采用公共资源组进行调度，或联系MaxCompute技术支持团队进行升级。
- Spark 2.4.5使用变化
  - 如果使用Yarn-cluster模式在本地提交任务，需要新增环境变量 `export HADOOP_CONF_DIR=$SPARK_HOME/conf` 。
  - 如果使用Local模式进行调试，需要在 `$SPARK_HOME/conf` 目录下新建odps.conf文件，并添加如下配置。

```
odps.project.name =
odps.access.id =
odps.access.key =
odps.end.point =
```

- Spark 2.4.5参数配置变化
  - `spark.sql.catalogImplementation` 配置为 `hive` 。
  - `spark.sql.sources.default` 配置为 `hive` 。
  - `spark.sql.odps.columnarReaderBatchSize` ，向量化读每个batch包含的行数，默认值为4096。
  - `spark.sql.odps.enableVectorizedReader` ，开启向量化读，默认值为True。
  - `spark.sql.odps.enableVectorizedWriter` ，开启向量化写，默认值为True。
  - `spark.sql.odps.split.size` ，该配置可以用来调节读MaxCompute表的并发度，默认每个分区为256 MB。

Spark 3.1.1使用注意事项

- 使用Spark 3.1.1提交作业
  - 直接使用Yarn-cluster模式在本地提交任务。详情请参见Cluster模式。

- DataWorks平台尚未部署，暂时不支持直接通过DataWorks模式提交。针对Scala、Java类型的作业，可以在spark-defaults.conf配置文件添加以下参数运行Spark 3.x。

```
spark.hadoop.odps.cupid.resources = public.__spark_libs__3.1.1-odps0.33.0.zip,[projectname].[用户主jar包],[projectname].[用户其他jar包]
spark.driver.extraClassPath = ./public.__spark_libs__3.1.1-odps0.33.0.zip/*
spark.executor.extraClassPath = ./public.__spark_libs__3.1.1-odps0.33.0.zip/*
```

> **说明**
> - 采用如上方式提交任务，需要将所依赖的资源全部添加到spark.hadoop.odps.cupid.resources参数中（包括主JAR包），否则可能出现类找不到的问题。
> - 暂时还无法通过以上方式提交Pyspark作业。

- Spark 3.1.1使用变化
  - 如果使用Yarn-cluster模式在本地提交任务，需要新增环境变量 `export HADOOP_CONF_DIR=$SPARK_HOME/conf` 。
  - 如果使用Yarn-cluster模式提交PySpark作业，需要在spark-defaults.conf配置文件添加以下参数使用Python3。

```
spark.hadoop.odps.cupid.resources = public.python-3.7.9-ucs4.tar.gz
spark.pyspark.python = ./public.python-3.7.9-ucs4.tar.gz/python-3.7.9-ucs4/bin/python3
```

  - 如果使用Local模式进行调试：
    - 需要在 `$SPARK_HOME/conf` 目录下新建odps.conf文件，并添加如下配置。

```
odps.project.name =
odps.access.id =
odps.access.key =
odps.end.point =
```

    - 需要添加 `spark.hadoop.fs.defaultFS = file:///` ，示例如下。

```
val spark = SparkSession
  .builder()
  .config("spark.hadoop.fs.defaultFS", "file:///")
  .enableHiveSupport()
  .getOrCreate()
```

- Spark 3.1.1参数配置变化
  - `spark.sql.defaultCatalog` 配置为 `odps` 。
  - `spark.sql.catalog.odps` 配置为 `org.apache.spark.sql.execution.datasources.v2.odps.OdpsTableCatalog` 。
  - `spark.sql.sources.partitionOverwriteMode` 配置为 `dynamic` 。
  - `spark.sql.extensions` 配置为 `org.apache.spark.sql.execution.datasources.v2.odps.extension.OdpsExtensions` 。
  - `spark.sql.odps.enableVectorizedReader` ，开启向量化读，默认值为True。
  - `spark.sql.odps.enableVectorizedWriter` ，开启向量化写，默认值为True。
  - `spark.sql.catalog.odps.splitSizeInMB` ，该配置可以用来调节读MaxCompute表的并发度，默认每个分区为256 MB。

8.3. 运行模式

MaxCompute Spark支持三种运行方式：Local模式、Cluster模式和DataWorks执行模式。

Local模式

MaxCompute Spark支持用户以原生Spark Local模式进行作业调试。

与Yarn Cluster模式类似，您首先需要做以下准备工作：

1. 准备MaxCompute项目以及对应的AccessKey ID、AccessKey Secret。
2. 下载MaxCompute Spark客户端。
3. 准备环境变量。
4. 配置spark-defaults.conf。
5. 下载工程模版并编译。

上述操作更多信息，请参见搭建开发环境。

通过MaxCompute Spark客户端以Spark-Submit方式提交作业，代码示例如下：

```
## Java/Scala
cd $SPARK_HOME
./bin/spark-submit --master local[4] --class com.aliyun.odps.spark.examples.SparkPi \
/path/to/odps-spark-examples/spark-examples/target/spark-examples-2.0.0-SNAPSHOT-shaded.jar
## PySpark
cd $SPARK_HOME
./bin/spark-submit --master local[4] \
/path/to/odps-spark-examples/spark-examples/src/main/python/odps_table_rw.py
```

注意事项

- Local模式读写MaxCompute表速度慢，是因为Local模式是通过Tunnel来读写的，读写速度相比于Yarn Cluster模式慢。
- Local模式是在本地执行的，部分用户会经常遇到Local模式下可以访问VPC，但是在Yarn Cluster模式下无法访问VPC。

  Local模式是处于用户本机环境，网络没有隔离。而Yarn Cluster模式是处于MaxCompute的网络隔离环境中，必须要配置VPC访问相关参数。
- Local模式下访问VPC的Endpoint通常是外网Endpoint，而Yarn Cluster模式下访问VPC的Endpoint通常是VPC网络Endpoint。更多Endpoint信息，请参见Endpoint。
- IDEA Local模式下需要将相关配置写入代码中，而在Yarn Cluster模式运行时一定要将这些配置从代码中去掉。

IDEA Local模式执行

MaxCompute Spark支持用户在IDEA中以Local[N]的模式直接运行代码，而不需要通过MaxCompute Spark客户端提交，您需要注意以下两点：

- 在IDEA中运行Local模式时，不能直接引用spark-defaults.conf的配置，需要手动在代码中指定相关配置。配置示例如下：

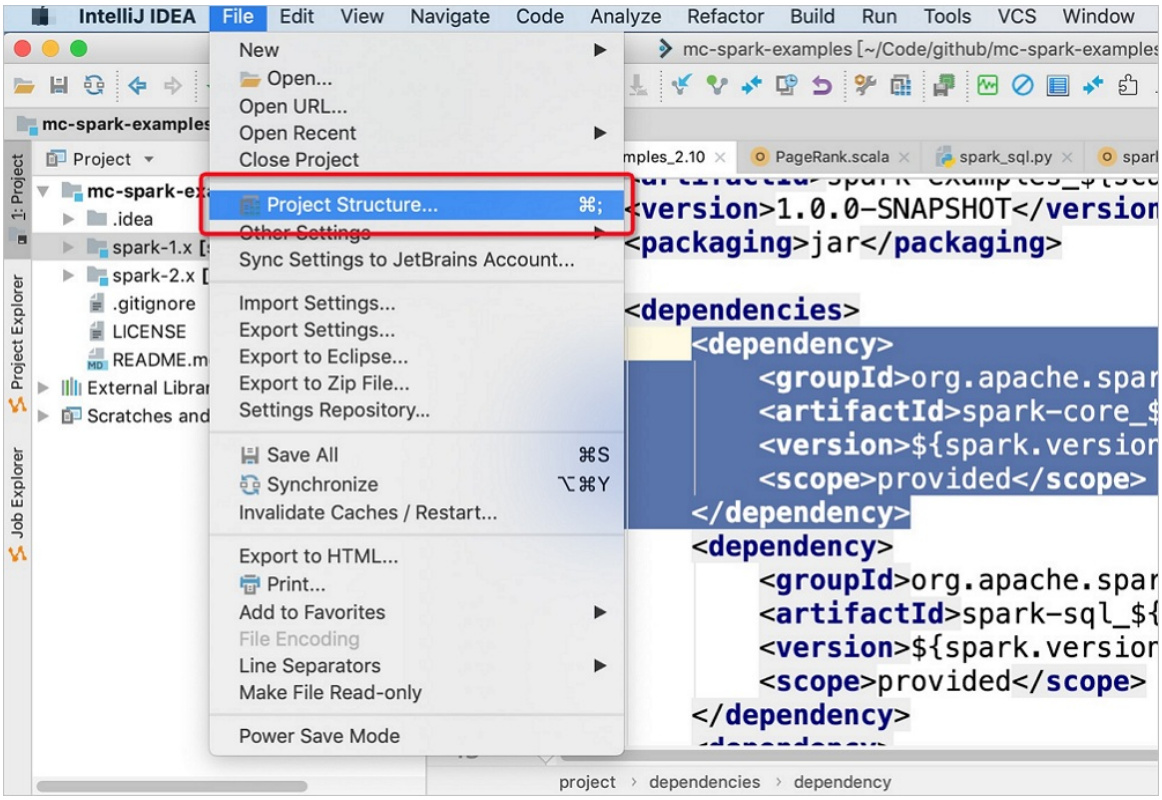
```
val spark = SparkSession
      .builder()
      .appName("SparkPi")
      .config("spark.master", "local[4]") // 需设置spark.master为local[N]才能直接运行，N为并发数。
      .config("spark.hadoop.odps.project.name", "<project_name>")
      .config("spark.hadoop.odps.access.id", "<accesskey_id>")
      .config("spark.hadoop.odps.access.key", "<accesskey_secret>")
      .config("spark.hadoop.odps.end.point", "http://service.cn.maxcompute.aliyun.com/api")
      .config("spark.sql.catalogImplementation", "odps")
      .getOrCreate()
```

- 务必注意需要在IDEA中手动添加MaxCompute Spark客户端的相关依赖（`jars`目录），否则会出现如下报错：

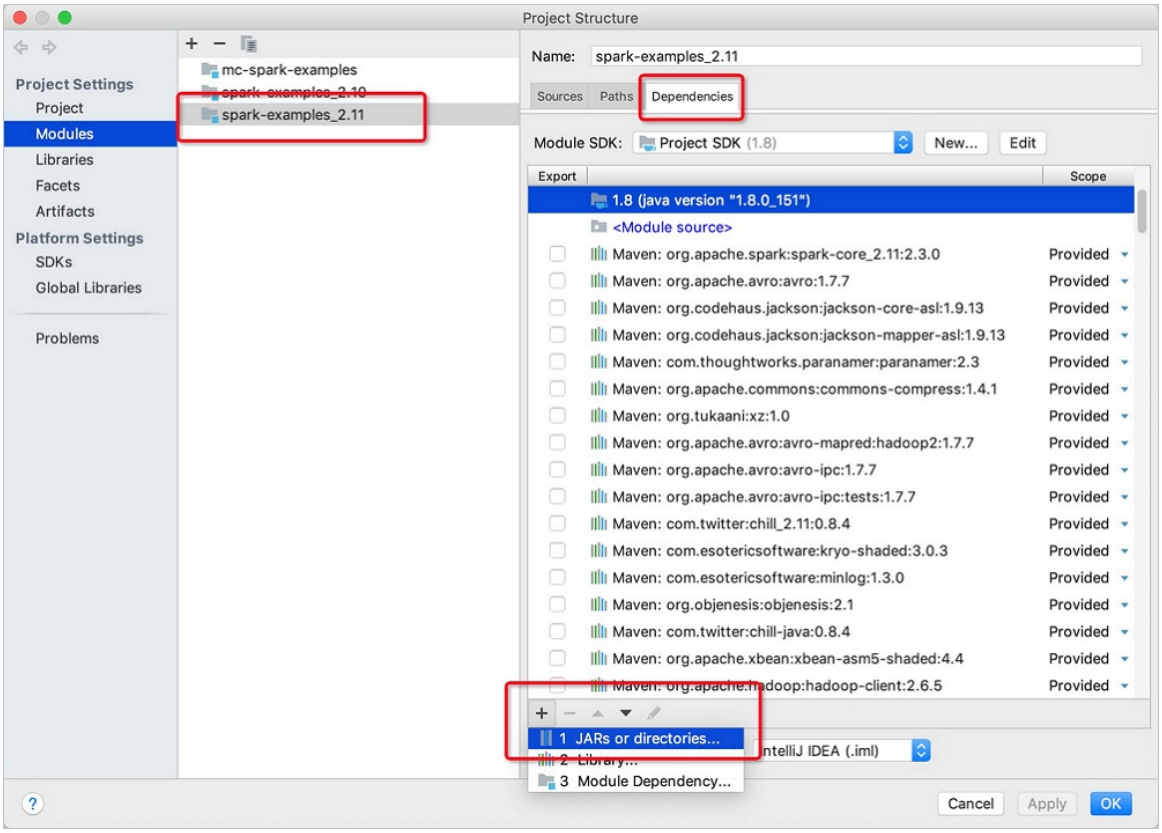
```
the value of spark.sql.catalogimplementation should be one of hive in-memory but was odps
```

  您可以按照如下流程配置依赖：

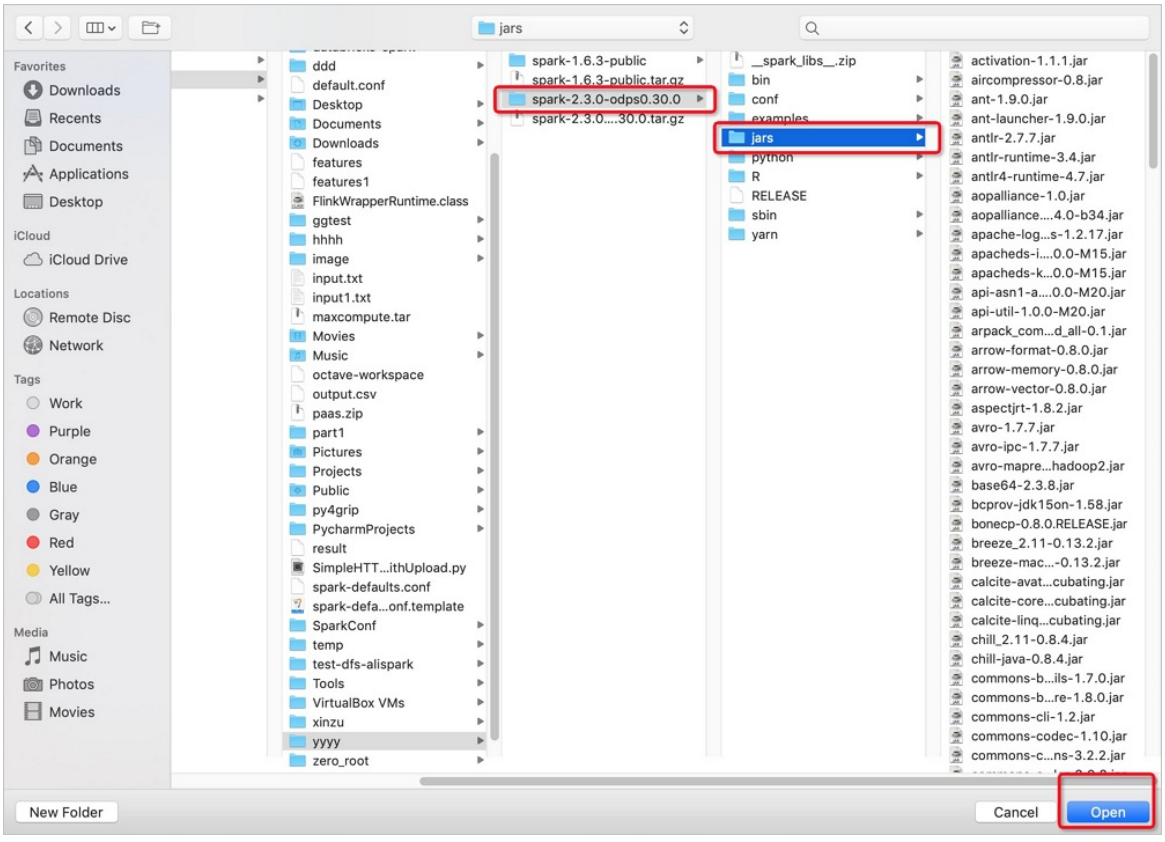
  i. 在IDEA的顶部菜单栏，选择File > Project Structure。



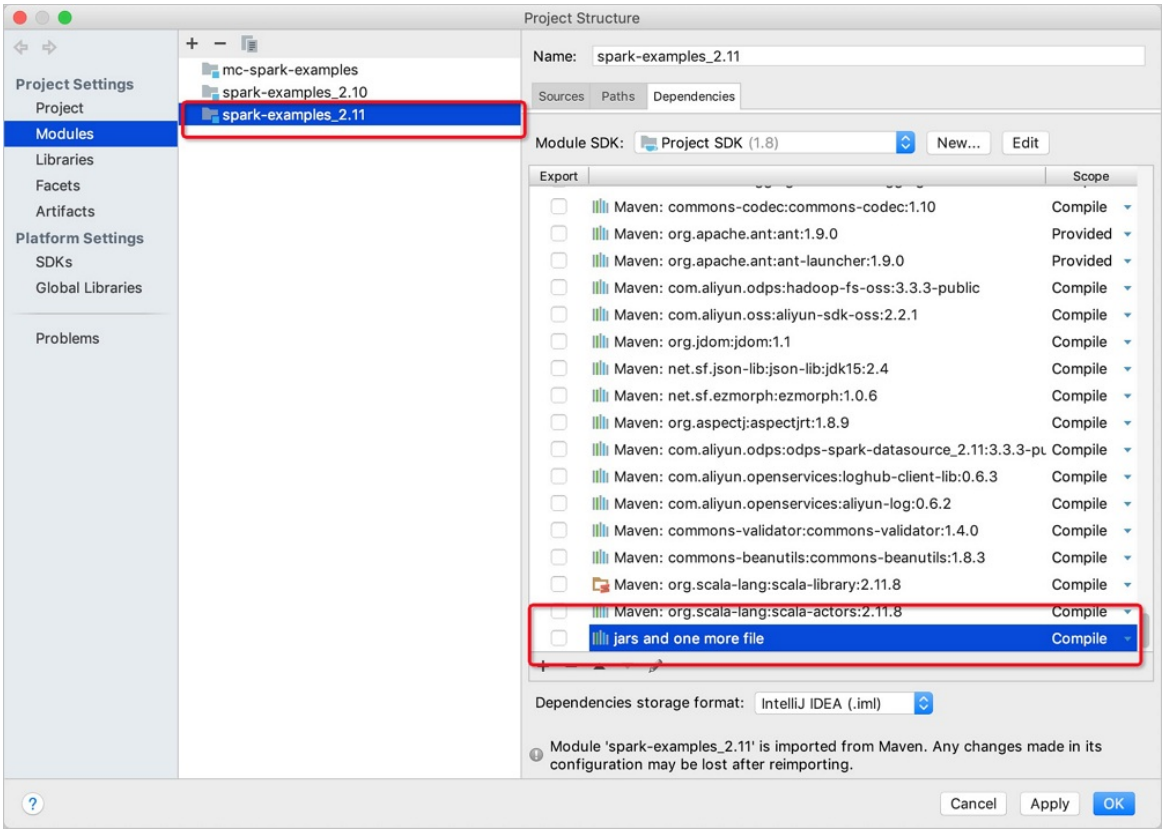
ii. 在**Project Structure**的**Modules**页面，选择目标Spark Module。单击右侧**Dependencies**后，在左下角单击+图标，选择**JARS or directories...**。



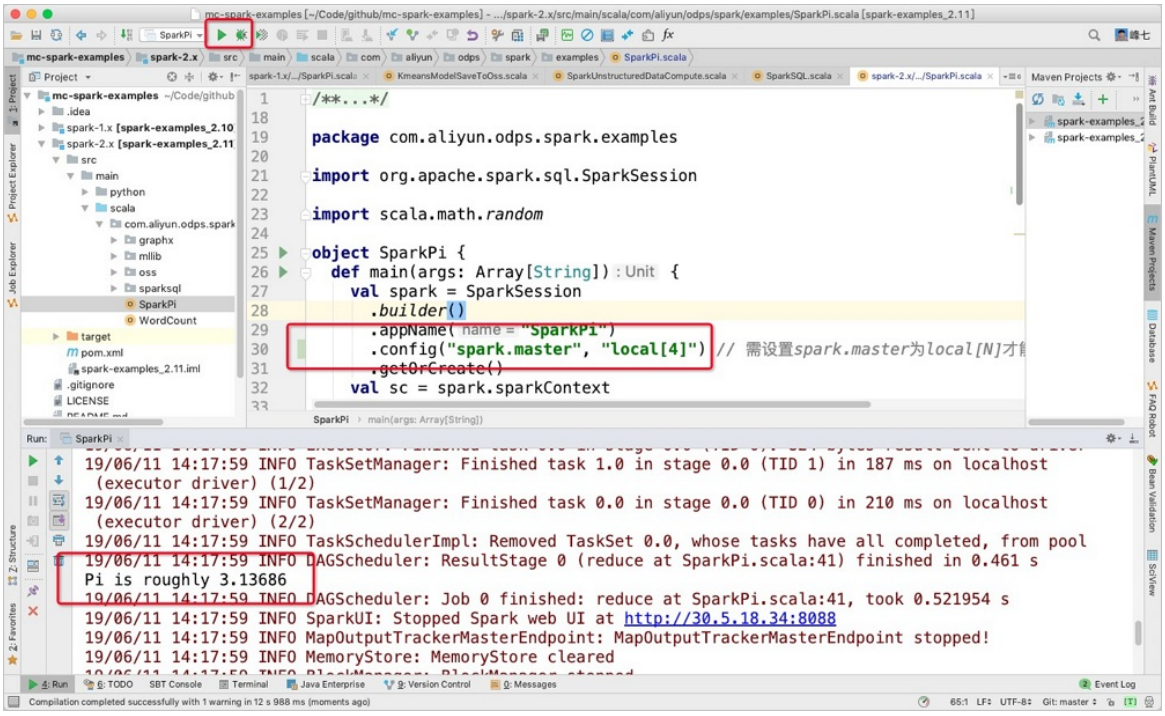
iii. 在打开的**jars**目录下，选择MaxCompute Spark版本及**jars**，单击**Open**。



iv. 单击**OK**。



v. 通过IDEA提交作业。



Cluster模式

在Cluster模式中，您需要指定自定义程序入口main。main结束（Success or Fail）时，对应的Spark作业就会结束。使用场景适合于离线作业，可与阿里云DataWorks产品结合进行作业调度，命令行提交方式如下。

```
# /path/to/MaxCompute-Spark为编译后的Application JAR包路径。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.SparkPi \
/path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

DataWorks执行模式

您可以在DataWorks中运行MaxCompute Spark离线作业（Cluster模式），以方便与其它类型执行节点集成和调度。

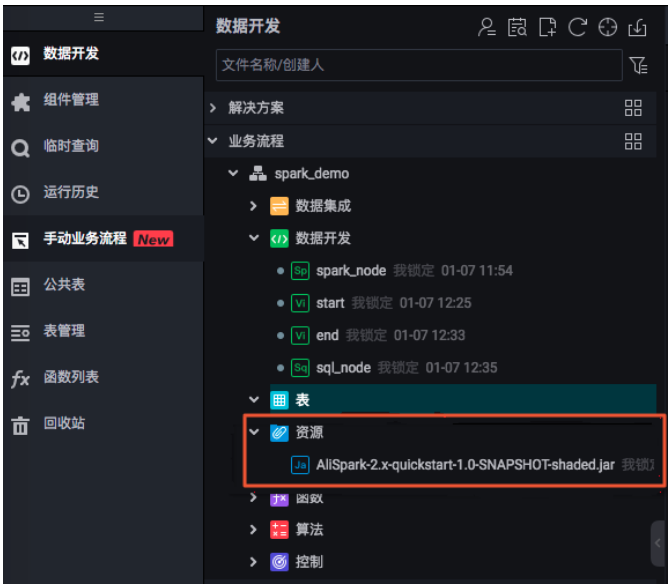
> 说明 DataWorks的Spark节点目前已经支持的Region：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）、中国（香港）、美国西部1（硅谷）、欧洲中部1（法兰克福）、亚太南部1（孟买）、亚太东南1（新加坡）。

操作步骤如下：

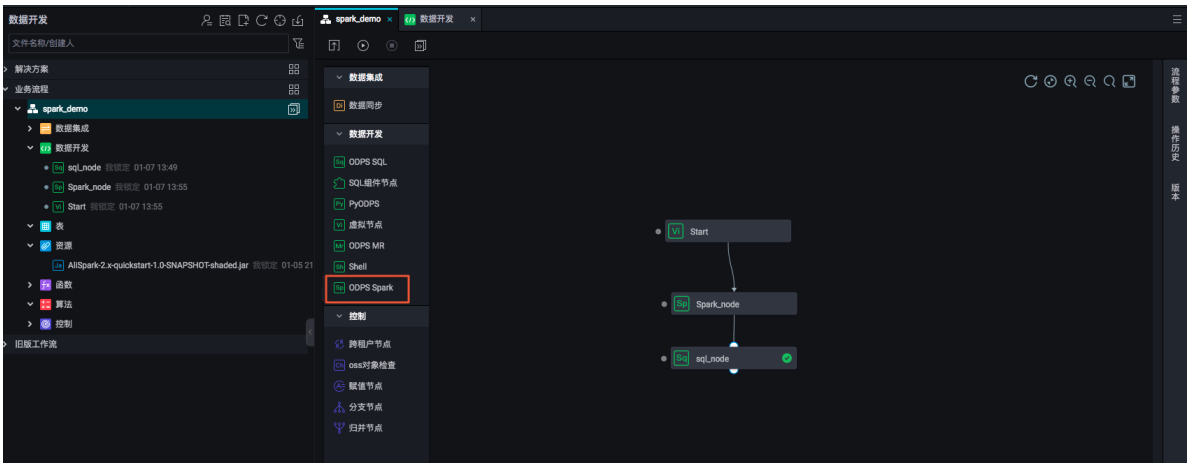
1. 您需要在DataWorks的业务流程中上传并提交（单击**提交**按钮）资源。



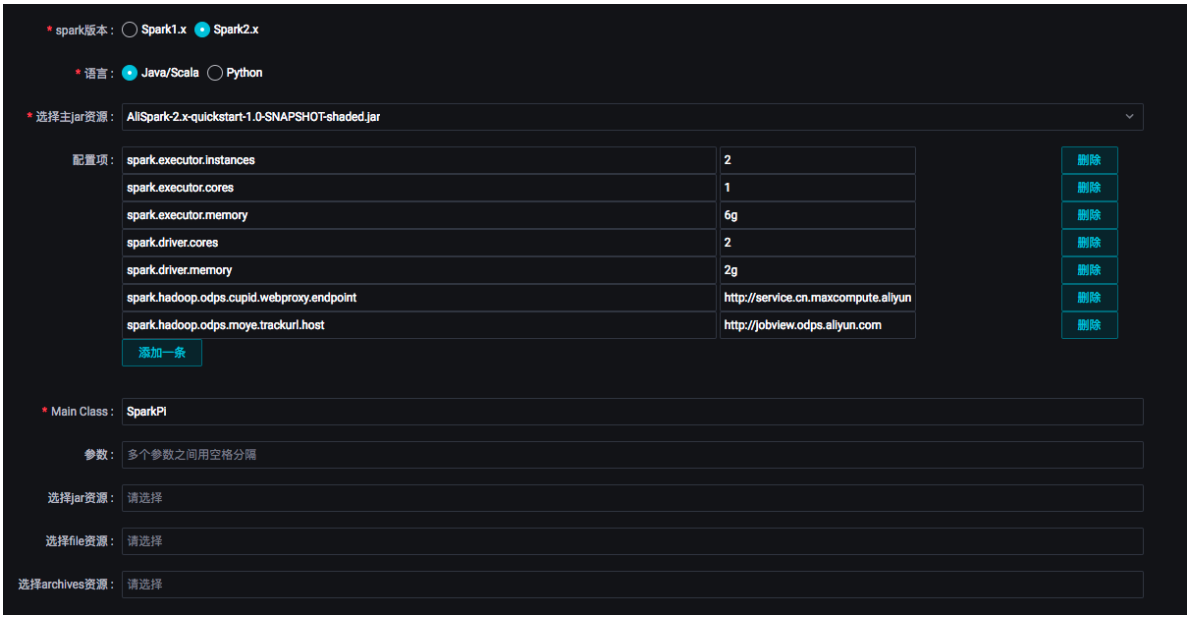
上传成功如下图所示。



2. 在创建的业务流程中，从**数据开发**组件中选择**ODPS Spark**节点。



3. 双击工作流中的Spark节点，对Spark作业进行任务定义。



ODPS Spark节点支持两种**spark版本**和**语言**。选择不同的**语言**，会显示相应不同的配置。您可以根据界面提示进行配置，参数详情请参见创建ODPS Spark节点。其中：

- **选择主jar资源**：指定任务所使用的资源文件。此处的资源文件需要您提前上传至DataWorks上。
- **配置项**：指定提交作业时的配置项。

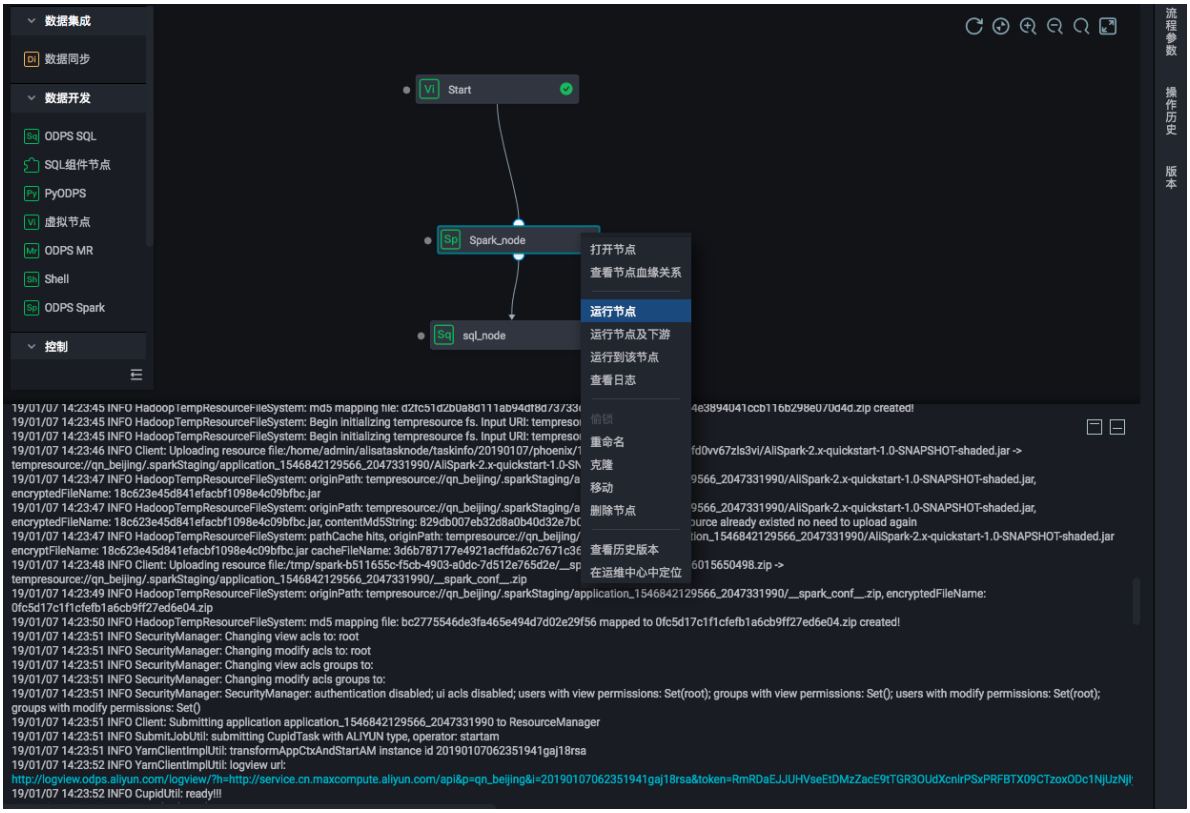
  其中 `spark.hadoop.odps.access.id` 、 `spark.hadoop.odps.access.key` 和 `spark.hadoop.odps.end.point` 无需配置，默认为MaxCompute项目的值（有特殊原因可显式配置，将覆盖默认值）。

  除此之外， `spark-defaults.conf` 中的配置需要逐条加到ODPS SPARK节点配置项中，例如Executor的数量、内存大小和 `spark.hadoop.odps.runtime.end.point` 的配置。

  ODPS SPARK节点的资源文件和配置项对应于spark-submit命令的参数和选项，如下表。此外，您也不需要上传*spark-defaults.conf*文件，而是将*spark-defaults.conf*文件中的配置都逐条加到ODPS SPARK节点配置项中。

ODPS SPARK节点	spark-submit
主Java、Python资源	`app jar or python file`
配置项	`--conf PROP=VALUE`
Main Class	`--class CLASS_NAME`
参数	`[app arguments]`
选择JAR资源	`--jars JARS`
选择Python资源	`--py-files PY_FILES`
选择File资源	`--files FILES`
选择Archives资源	`--archives ARCHIVES`

4. 手动执行Spark节点，可以查看该任务的执行日志，从打印出来的日志中可以获取该任务的Logview和Jobview的URL，编译进一步查看与诊断。



Spark作业定义完成后，即可在业务流程中对不同类型服务进行编排、统一调度执行。

8.4. 开发示例

8.4.1. 概述

本文为您介绍使用Java和Scala语言开发的示例。

下载示例工程

MaxCompute Spark提供了项目示例工程模板，建议您下载复制后直接在模板里进行开发。

执行如下命令下载示例工程。

```
# spark-1.x 模板及编译
git clone https://github.com/aliyun/MaxCompute-Spark.git
cd spark-1.x
mvn clean package
# spark-2.x 模板及编译
git clone https://github.com/aliyun/MaxCompute-Spark.git
cd spark-2.x
mvn clean package
```

> 注意 示例工程里关于Spark依赖的Scope为provided，此参数请不要更改，否则提交的作业无法正常运行。

Spark-1.x示例

Spark-1.x示例如下：

- WordCount示例（Scala）
- MaxCompute Table读写示例（Scala）
- MaxCompute Table读写示例（Python）
- MaxCompute Table读写示例（Java）

Spark-2.x示例

Spark-2.x示例如下：

- WordCount示例（Scala）

- MaxCompute Table读写示例（Scala）
- GraphX PageRank示例（Scala）
- Mllib Kmeans-ON-OSS示例（Scala）
- OSS UnstructuredData示例（Scala）
- SparkPi示例（Scala）
- 支持Spark Streaming LogHub示例（Scala）
- 支持Spark Streaming LogHub写MaxCompute示例（Scala）
- 支持Spark Streaming DataHub示例（Scala）
- 支持Spark Streaming DataHub写MaxCompute示例（Scala）
- 支持Spark Streaming Kafka示例（Scala）
- 支持Spark StructuredStreaming DataHub示例（Scala）
- 支持Spark StructuredStreaming Kafka示例（Scala）
- 支持Spark StructuredStreaming LogHub示例（Scala）
- MaxCompute Table读写PySpark示例（Python）
- PySpark写OSS示例（Python）
- Spark-SQL示例（Java）
- 从MaxCompute中读取数据写入HBase
- 读写OSS文件
- 读MaxCompute写OSS

8.4.2. Spark-1.x示例

本文为您介绍Spark-1.x依赖的配置以及Spark-1.x相关示例。

配置Spark-1.x的依赖

通过MaxCompute提供的Spark客户端提交应用，需要在*pom.xml*文件中添加以下依赖。

```
<properties>
    <spark.version>1.6.3</spark.version>
    <cupid.sdk.version>3.3.3-public</cupid.sdk.version>
    <scala.version>2.10.4</scala.version>
    <scala.binary.version>2.10</scala.binary.version>
</properties>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-streaming_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>cupid-sdk</artifactId>
    <version>${cupid.sdk.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>hadoop-fs-oss</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-spark-datasource_${scala.binary.version}</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-actors</artifactId>
    <version>${scala.version}</version>
</dependency>
```

上述代码中Scope的定义如下：

- spark-core、spark-sql等所有Spark社区发布的包，使用providedScope。
- odps-spark-datasource使用默认的compileScope。

WordCount示例（Scala）

- 代码示例

  WordCount.scala

- 提交方式

```
cd /path/to/MaxCompute-Spark/spark-1.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.WordCount \
    /path/to/MaxCompute-Spark/spark-1.x/target/spark-examples_2.10-1.0.0-SNAPSHOT-shaded.jar
```

MaxCompute Table读写示例（Scala）

- 代码示例

  SparkSQL.scala

- 提交方式

```
cd /path/to/MaxCompute-Spark/spark-1.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.sparksql.SparkSQL \
    /path/to/MaxCompute-Spark/spark-1.x/target/spark-examples_2.10-1.0.0-SNAPSHOT-shaded.jar
```

MaxCompute Table读写示例（Python）

实现MaxCompute Table读写的Python示例代码请参见spark_sql.py。

MaxCompute Table读写示例（Java）

实现MaxCompute Table读写的Java示例代码请参见JavaSparkSQL.java。

8.4.3. Spark-2.x示例

本文为您介绍Spark-2.x依赖的配置以及Spark-2.x示例说明。

配置Spark-2.x的依赖

通过MaxCompute提供的Spark客户端提交应用时，需要在*pom.xml*文件中添加以下依赖。*pom.xml*文件请参见pom.xml。

```
<properties>
    <spark.version>2.3.0</spark.version>
    <cupid.sdk.version>3.3.8-public</cupid.sdk.version>
    <scala.version>2.11.8</scala.version>
    <scala.binary.version>2.11</scala.binary.version>
</properties>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-streaming_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>cupid-sdk</artifactId>
    <version>${cupid.sdk.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>hadoop-fs-oss</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-spark-datasource_${scala.binary.version}</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-actors</artifactId>
    <version>${scala.version}</version>
</dependency>
```

上述代码中Scope的定义如下：

- spark-core、spark-sql等所有Spark社区发布的包，设置Scope为provided。
- odps-spark-datasource设置Scope为compile。

WordCount示例（Scala）

- 代码示例

  WordCount.scala

- 提交方式

```
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.WordCount \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

MaxCompute Table读写示例（Scala）

- 代码示例

  SparkSQL.scala

- 提交方式

```
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.sparksql.SparkSQL \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

GraphX PageRank示例（Scala）

- 代码示例

  PageRank.scala

- 提交方式

```
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.graphx.PageRank \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

Mllib Kmeans-ON-OSS示例（Scala）

`spark.hadoop.fs.oss.ststoken.roleArn` 和 `spark.hadoop.fs.oss.endpoint` 的填写请参见Oss-Access文档说明。

- 代码示例

  KmeansModelSaveToOss.scala

- 提交方式

```
# 编辑代码。
val modelOssDir = "oss://bucket/kmeans-model" // 填写具体的OSS Bucket路径。
val spark = SparkSession
  .builder()
  .config("spark.hadoop.fs.oss.credentials.provider", "org.apache.hadoop.fs.aliyun.oss.AliyunStsTokenCredentialsProvider")
  .config("spark.hadoop.fs.oss.ststoken.roleArn", "acs:ram::****:role/aliyunodpsdefaultrole")
  .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
  .appName("KmeansModelSaveToOss")
  .getOrCreate()
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

OSS UnstructuredData示例（Scala）

`spark.hadoop.fs.oss.ststoken.roleArn` 和 `spark.hadoop.fs.oss.endpoint` 的填写请参见Oss-Access文档说明。

- 代码示例

  SparkUnstructuredDataCompute.scala

- 提交方式

```
# 编辑代码。
val pathIn = "oss://bucket/inputdata/" // 填写具体的OSS Bucket路径。
val spark = SparkSession
  .builder()
  .config("spark.hadoop.fs.oss.credentials.provider", "org.apache.hadoop.fs.aliyun.oss.AliyunStsTokenCredentialsProvider")
  .config("spark.hadoop.fs.oss.ststoken.roleArn", "acs:ram::****:role/aliyunodpsdefaultrole")
  .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
  .appName("SparkUnstructuredDataCompute")
  .getOrCreate()
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.oss.SparkUnstructuredDataCompute \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

SparkPi示例（Scala）

- 代码示例

  SparkPi.scala

- 提交方式

```
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.SparkPi \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

支持Spark Streaming LogHub示例（Scala）

- 代码示例

  LogHubStreamingDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.streaming.loghub.LogHubStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

支持Spark Streaming LogHub写MaxCompute示例（Scala）

- 代码示例

  LogHub2OdpsDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.streaming.loghub.LogHub2OdpsDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

支持Spark Streaming DataHub示例（Scala）

- 代码示例

  DataHubStreamingDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.streaming.datahub.DataHubStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

支持Spark Streaming DataHub写MaxCompute示例（Scala）

- 代码示例

  DataHub2OdpsDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.streaming.datahub.DataHub2OdpsDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

支持Spark Streaming Kafka示例（Scala）

- 代码示例

  KafkaStreamingDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.streaming.kafka.KafkaStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

> 说明 更多信息请参见MaxCompute-Spark。

支持Spark StructuredStreaming DataHub示例（Scala）

- 代码示例

  DatahubStructuredStreamingDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.structuredstreaming.datahub.DatahubStructuredStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

支持Spark StructuredStreaming Kafka示例（Scala）

- 代码示例

  KafkaStructuredStreamingDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.structuredstreaming.kafka.KafkaStructuredStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

支持Spark StructuredStreaming LogHub示例（Scala）

- 代码示例

  LoghubStructuredStreamingDemo.scala

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.structuredstreaming.loghub.LoghubStructuredStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

MaxCompute Table读写PySpark示例（Python）

- 代码示例

  spark_sql.py

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --jars /path/to/odps-spark-datasource_2.11-3.3.8-public.jar \
    /path/to/MaxCompute-Spark/spark-2.x/src/main/python/spark_sql.py
```

PySpark写OSS示例（Python）

- 代码示例

  spark_oss.py

- 提交方式

```
# 环境变量spark-defaults.conf的配置请参见搭建开发环境。
# OSS相关配置请参见OSS Access文档说明。
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --jars /path/to/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar \
    /path/to/MaxCompute-Spark/spark-2.x/src/main/python/spark_oss.py
# spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar可以通过Spark-2.x编译得到。
```

Spark-SQL示例（Java）

Spark-SQL Java示例代码请参见JavaSparkSQL.java。

从MaxCompute中读取数据写入HBase

通过Intellij IDEA工具编写代码，实现从MaxCompute中读取数据写入HBase。

- 代码示例

```
object McToHbase {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("spark_sql_ddl")
      .config("spark.sql.catalogImplementation", "odps")
      .config("spark.hadoop.odps.end.point","http://service.cn.maxcompute.aliyun.com/api")
      .config("spark.hadoop.odps.runtime.end.point","http://service.cn.maxcompute.aliyun-inc.com/api")
      .getOrCreate()
      val sc = spark.sparkContext
      val config = HBaseConfiguration.create()
      val zkAddress = ""
      config.set(HConstants.ZOOKEEPER_QUORUM, zkAddress);
      val jobConf = new JobConf(config)
      jobConf.setOutputFormat(classOf[TableOutputFormat])
      jobConf.set(TableOutputFormat.OUTPUT_TABLE,"test")
    try{
      import spark._
      spark.sql("select '7', 'long'").rdd.map(row => {
        val id = row(0).asInstanceOf[String]
        val name = row(1).asInstanceOf[String]
        val put = new Put(Bytes.toBytes(id))
        put.addColumn(Bytes.toBytes("cf"), Bytes.toBytes("a"), Bytes.toBytes(name))
        (new ImmutableBytesWritable, put)
    }).saveAsHadoopDataset(jobConf)
  } finally {
    sc.stop()
  }
  }
}
```

- 提交方式：通过Intellij IDEA提交并运行示例代码。更多操作信息，请参见Spark在MaxCompute的运行方式。

读写OSS文件

通过Intellij IDEA工具或DataWorks，实现读写OSS文件。

- 代码示例

- 示例1：Local模式下的代码示例。

```
package com.aliyun.odps.spark.examples
import java.io.ByteArrayInputStream
import org.apache.spark.sql.SparkSession
object SparkOSS {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .config("spark.master", "local[4]") // 需要设置spark.master为local[N]才能直接运行，N为并发数。
      .config("spark.hadoop.fs.oss.accessKeyId", "")
      .config("spark.hadoop.fs.oss.accessKeySecret", "")
      .config("spark.hadoop.fs.oss.endpoint", "oss-cn-beijing.aliyuncs.com")
      .appName("SparkOSS")
      .getOrCreate()
    val sc = spark.sparkContext
    try {
      //OSS文件的读取。
      val pathIn = "oss://spark-oss/workline.txt"
      val inputData = sc.textFile(pathIn, 5)
            //RDD写入。
      inputData.repartition(1).saveAsTextFile("oss://spark-oss/user/data3")
    } finally {
      sc.stop()
    }
  }
}
```

> **说明** 执行该代码前，请您务必检查是否已添加了 `hadoop-fs-oss` 依赖，否则会报错。

- 示例2：Local模式下的代码示例。

```
package com.aliyun.odps.spark.examples
import java.io.ByteArrayInputStream
import com.aliyun.oss.{OSSClientBuilder,OSSClient}
import org.apache.spark.sql.SparkSession
object SparkOSS {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .config("spark.master", "local[4]") // 需要设置spark.master为local[N]才能直接运行，N为并发数。
      .config("spark.hadoop.fs.oss.accessKeyId", "")
      .config("spark.hadoop.fs.oss.accessKeySecret", "")
      .config("spark.hadoop.fs.oss.endpoint", "oss-cn-beijing.aliyuncs.com")
      .appName("SparkOSS")
      .getOrCreate()
    val sc = spark.sparkContext
    try {
      //OSS文件的读取。
      val pathIn = "oss://spark-oss/workline.txt"
      val inputData = sc.textFile(pathIn, 5)
      val cnt = inputData.count
      inputData.count()
      println(s"count: $cnt")
      //OSS文件的写入。
      val ossClient = new OSSClientBuilder().build("oss-cn-beijing.aliyuncs.com", "<accessKeyId>", "<accessKeySecret>")
      val filePath="user/data"
      ossClient.putObject("spark-oss",filePath , new ByteArrayInputStream(cnt.toString.getBytes()))
      ossClient.shutdown()
    } finally {
      sc.stop()
    }
  }
}
```

- 示例3：Cluster模式下的代码示例。

```
package com.aliyun.odps.spark.examples
import java.io.ByteArrayInputStream
import com.aliyun.oss.{OSSClientBuilder,OSSClient}
import org.apache.spark.sql.SparkSession
object SparkOSS {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("SparkOSS")
      .getOrCreate()
    val sc = spark.sparkContext
    try {
      //OSS文件的读取。
      val pathIn = "oss://spark-oss/workline.txt"
      val inputData = sc.textFile(pathIn, 5)
      val cnt = inputData.count
      inputData.count()
      println(s"count: $cnt")
      // inputData.repartition(1).saveAsTextFile("oss://spark-oss/user/data3")
      //OSS文件的写入。
      val ossClient = new OSSClientBuilder().build("oss-cn-beijing.aliyuncs.com", "<accessKeyId>", "<accessKeySecret>")
      val filePath="user/data"
      ossClient.putObject("spark-oss",filePath , new ByteArrayInputStream(cnt.toString.getBytes()))
      ossClient.shutdown()
    } finally {
      sc.stop()
    }
  }
}
```

- 提交方式：
  - Local模式下的代码通过Intellij IDEA开发、测试并提交。更多操作信息，请参见Spark在MaxCompute的运行方式。
  - 在DataWorks上通过ODPS Spark节点提交并运行。详情请参见创建ODPS Spark节点。

读MaxCompute写OSS

通过Intellij IDEA工具或DataWorks，实现读取MaxCompute数据并写入OSS。

- 代码示例
  - Local模式下的示例代码。

```
package com.aliyun.odps.spark.examples.userpakage
import org.apache.spark.sql.{SaveMode, SparkSession}
object SparkODPS2OSS {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder()
      .appName("Spark2OSS")
      .config("spark.master", "local[4]")// 需设置spark.master为local[N]才能直接运行，N为并发数
      .config("spark.hadoop.odps.project.name", "")
      .config("spark.hadoop.odps.access.id", "")
      .config("spark.hadoop.odps.access.key", "")
      .config("spark.hadoop.odps.end.point", "http://service.cn.maxcompute.aliyun.com/api")
      .config("spark.sql.catalogImplementation", "odps")
      .config("spark.hadoop.fs.oss.accessKeyId","")
      .config("spark.hadoop.fs.oss.accessKeySecret","")
      .config("spark.hadoop.fs.oss.endpoint","oss-cn-beijing.aliyuncs.com")
      .getOrCreate()
    try{
      //通过SparkSql查询表
      val data = spark.sql("select * from  user_detail")
     //展示查询数据
      data.show(10)
      //将查询到的数据存储到一个OSS的文件中
      data.toDF().coalesce(1).write.mode(SaveMode.Overwrite).csv("oss://spark-oss/user/data3")
    }finally {
      spark.stop()
    }
  }
}
```

- Cluster模式下的示例代码。

```
package com.aliyun.odps.spark.examples.userpakage
import org.apache.spark.sql.{SaveMode, SparkSession}
object SparkODPS2OSS {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder()
      .appName("SparkODPS2OSS")
      .getOrCreate()
    try{
      //通过SparkSql查询表
      val data = spark.sql("select * from  user_detail")
     //展示查询数据
      data.show(10)
      //将查询到的数据存储到一个OSS的文件中
      data.toDF().coalesce(1).write.mode(SaveMode.Overwrite).csv("oss://spark-oss/user/data3")
    }finally {
      spark.stop()
    }
  }
}
```

- 提交方式：
  - Local模式下的代码通过Intellij IDEA开发、测试并提交。
  - 在DataWorks上通过ODPS Spark节点提交并运行。详情请参见创建ODPS Spark节点。

> **说明** Spark开发环境的配置请参见Spark在MaxCompute的运行方式。

8.5. PySpark开发示例

本文为您介绍PySpark开发示例。

如果要访问MaxCompute表，则需要编译datasource包，详细步骤请参见搭建开发环境。

SparkSQL应用示例（Spark1.6）

详细代码

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import OdpsContext
if __name__ == '__main__':
    conf = SparkConf().setAppName("odps_pyspark")
    sc = SparkContext(conf=conf)
    sql_context = OdpsContext(sc)
    sql_context.sql("DROP TABLE IF EXISTS spark_sql_test_table")
    sql_context.sql("CREATE TABLE spark_sql_test_table(name STRING, num BIGINT)")
    sql_context.sql("INSERT INTO TABLE spark_sql_test_table SELECT 'abc', 100000")
    sql_context.sql("SELECT * FROM spark_sql_test_table").show()
    sql_context.sql("SELECT COUNT(*) FROM spark_sql_test_table").show()
```

提交运行

```
./bin/spark-submit \
--jars cupid/odps-spark-datasource_xxx.jar \
example.py
```

SparkSQL应用示例（Spark2.3）

详细代码

```
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession.builder.appName("spark sql").getOrCreate()
    spark.sql("DROP TABLE IF EXISTS spark_sql_test_table")
    spark.sql("CREATE TABLE spark_sql_test_table(name STRING, num BIGINT)")
    spark.sql("INSERT INTO spark_sql_test_table SELECT 'abc', 100000")
    spark.sql("SELECT * FROM spark_sql_test_table").show()
    spark.sql("SELECT COUNT(*) FROM spark_sql_test_table").show()
```

提交运行

- Cluster模式提交运行

```
spark-submit --master yarn-cluster \
--jars cupid/odps-spark-datasource_xxx.jar \
example.py
```

- Local模式运行

```
cd $SPARK_HOME
./bin/spark-submit --master local[4] \
--driver-class-path cupid/odps-spark-datasource_xxx.jar \
/path/to/odps-spark-examples/spark-examples/src/main/python/spark_sql.py
```

> 说明
> - Local模式访问表依赖Tunnel。
> - Local模式要用--driver-class-path而非--jars。

Package依赖

由于MaxCompute集群无法自由安装Python库，PySpark依赖其它Python库、插件、项目时，通常需要在本地打包后通过Spark-submit上传。对于特定依赖，打包环境需与线上环境保持一致。打包方式如下，请根据业务的复杂度进行选择：

- 不打包直接采用公共资源
  - 默认提供Python 2.7.13环境配置

```
spark.hadoop.odps.cupid.resources = public.python-2.7.13-ucs4.tar.gz
spark.pyspark.python = ./public.python-2.7.13-ucs4.tar.gz/python-2.7.13-ucs4/bin/python
```

第三方库列表如下。

```
$./bin/pip list
Package                       Version
----------------------------- -----------
absl-py                       0.11.0
aenum                         2.2.4
asn1crypto                    0.23.0
astor                         0.8.1
astroid                       1.6.1
atomicwrites                  1.4.0
attrs                         20.3.0
backports.functools-lru-cache 1.6.1
backports.lzma                0.0.14
backports.weakref             1.0.post1
beautifulsoup4                4.9.3
bleach                        2.1.2
boto                          2.49.0
boto3                         1.9.147
botocore                      1.12.147
bz2file                       0.98
cachetools                    3.1.1
category-encoders             2.2.2
certifi                       2019.9.11
cffi                          1.11.2
chardet                       3.0.4
click                         6.7
click-plugins                 1.1.1
cligj                         0.7.0
cloudpickle                   0.5.3
configparser                  4.0.2
contextlib2                   0.6.0.post1
cryptography                  2.6.1
cssutils                      1.0.2
cycler                        0.10.0
Cython                        0.29.5
dask                          0.18.1
DBUtils                       1.2
decorator                     4.2.1
docutils                      0.16
entrypoints                   0.2.3
enum34                        1.1.10
fake-useragent                0.1.11
Fiona                         1.8.17
funcsigs                      1.0.2
functools32                   3.2.3.post2
```

future	0.16.0
futures	3.3.0
gast	0.2.2
gensim	3.8.3
geopandas	0.6.3
getpass3	1.2
google-auth	1.23.0
google-auth-oauthlib	0.4.1
google-pasta	0.2.0
grpcio	1.33.2
h5py	2.7.0
happybase	1.1.0
html5lib	1.0.1
idna	2.10
imbalanced-learn	0.4.3
imblearn	0.0
importlib-metadata	2.0.0
ipaddress	1.0.23
ipython-genutils	0.2.0
isort	4.3.4
itchat	1.3.10
itsdangerous	0.24
jedi	0.11.1
jieba	0.42.1
Jinja2	2.10
jmespath	0.10.0
jsonschema	2.6.0
kafka-python	1.4.6
kazoo	2.5.0
Keras-Applications	1.0.8
Keras-Preprocessing	1.1.2
kiwisolver	1.1.0
lazy-object-proxy	1.3.1
libarchive-c	2.8
lightgbm	2.3.1
lml	0.0.2
lxml	4.2.1
MarkupSafe	1.0
matplotlib	2.2.5
mccabe	0.6.1
missingno	0.4.2
mistune	0.8.3
mock	2.0.0
more-itertools	5.0.0
munch	2.5.0
nbconvert	5.3.1
nbformat	4.4.0
networkx	2.1
nose	1.3.7
numpy	1.16.1
oauthlib	3.1.0
opt-einsum	2.3.2
packaging	20.4
pandas	0.24.2
pandocfilters	1.4.2
parso	0.1.1
pathlib2	2.3.5
patsy	0.5.1
pbr	3.1.1
pexpect	4.4.0
phpserialize	1.3
pickleshare	0.7.4
Pillow	6.2.0
pip	20.2.4
pluggy	0.13.1
ply	3.11
prompt-toolkit	2.0.1
protobuf	3.6.1
psutil	5.4.3
psycopg2	2.8.6
ptyprocess	0.5.2
py	1.9.0
py4j	0.10.6
pyasn1	0.4.8
pyasn1-modules	0.2.8
pycosat	0.6.3

```
pycosat                   0.6.3
pycparser                 2.18
pydot                     1.4.1
Pygments                  2.2.0
pykafka                   2.8.0
pylint                    1.8.2
pymongo                   3.11.0
PyMySQL                   0.10.1
pynliner                  0.8.0
pyodps                    0.9.3.1
pyOpenSSL                 17.5.0
pyparsing                 2.2.0
pypng                     0.0.20
pyproj                    2.2.2
PyQRCode                  1.2.1
pytest                    4.6.11
python-dateutil           2.8.1
pytz                      2020.4
PyWavelets                0.5.2
PyYAML                    3.12
redis                     3.2.1
requests                  2.25.0
requests-oauthlib         1.3.0
rope                      0.10.7
rsa                       4.5
ruamel.ordereddict        0.4.15
ruamel.yaml               0.11.14
s3transfer                0.2.0
scandir                   1.10.0
scikit-image              0.14.0
scikit-learn              0.20.3
scipy                     1.2.3
seaborn                   0.9.1
Send2Trash                1.5.0
setuptools                41.0.0
Shapely                   1.7.1
simplegeneric             0.8.1
singledispatch            3.4.0.3
six                       1.15.0
sklearn2                  0.0.13
smart-open                1.8.1
soupsieve                 1.9.6
SQLAlchemy                1.3.20
statsmodels               0.11.0
subprocess32              3.5.4
tabulate                  0.8.7
tensorflow                2.0.0
tensorflow-estimator      2.0.1
termcolor                 1.1.0
testpath                  0.3.1
thriftpy                  0.3.9
timeout-decorator         0.4.1
toolz                     0.9.0
tqdm                      4.32.2
traitlets                 4.3.2
urllib3                   1.24.3
wcwidth                   0.2.5
webencodings              0.5.1
Werkzeug                  1.0.1
wheel                     0.35.1
wrapt                     1.11.1
xgboost                   0.82
xlrd                      1.2.0
XlsxWriter                1.0.7
zipp                      1.2.0
```

- 默认提供Python 3.7.9环境配置

```
spark.hadoop.odps.cupid.resources = public.python-3.7.9-ucs4.tar.gz
spark.pyspark.python = ./public.python-3.7.9-ucs4.tar.gz/python-3.7.9-ucs4/bin/python3
```

第三方库列表如下。

```
Package                   Version
------------------------- -----------
appnope                   0.1.0
```

```
asn1crypto                      0.23.0
astroid                         1.6.1
attrs                           20.3.0
autopep8                        1.3.4
backcall                        0.2.0
backports.functools-lru-cache 1.5
backports.weakref               1.0rc1
beautifulsoup4                  4.6.0
bidict                          0.17.3
bleach                          2.1.2
boto                            2.49.0
boto3                           1.9.147
botocore                        1.12.147
bs4                             0.0.1
bz2file                         0.98
cached-property                 1.5.2
cachetools                      3.1.1
category-encoders               2.2.2
certifi                         2019.11.28
cffi                            1.11.2
chardet                         3.0.4
click                           6.7
click-plugins                   1.1.1
cligj                           0.7.0
cloudpickle                     0.5.3
cryptography                    2.6.1
cssutils                        1.0.2
cycler                          0.10.0
Cython                          0.29.21
dask                            0.18.1
DBUtils                         1.2
decorator                       4.2.1
docutils                        0.16
entrypoints                     0.2.3
fake-useragent                  0.1.11
Fiona                           1.8.17
future                          0.16.0
gensim                          3.8.3
geopandas                       0.8.0
getpass3                        1.2
h5py                            3.1.0
happybase                       1.1.0
html5lib                        1.0.1
idna                            2.10
imbalanced-learn                0.4.3
imblearn                        0.0
importlib-metadata              2.0.0
iniconfig                       1.1.1
ipykernel                       5.3.4
ipython                         7.19.0
ipython-genutils                0.2.0
isort                           4.3.4
itchat                          1.3.10
itsdangerous                    0.24
jedi                            0.11.1
jieba                           0.42.1
Jinja2                          2.10
jmespath                        0.10.0
jsonschema                      2.6.0
jupyter-client                  6.1.7
jupyter-core                    4.6.3
kafka-python                    1.4.6
kazoo                           2.5.0
kiwisolver                      1.3.1
lazy-object-proxy               1.3.1
libarchive-c                    2.8
lightgbm                        2.3.1
lml                             0.0.2
lxml                            4.2.1
Mako                            1.0.10
MarkupSafe                      1.0
matplotlib                      3.3.3
mccabe                          0.6.1
missingno                       0.4.2
mistune                         0.8.3
```

```
mock                    2.0.0
munch                   2.5.0
nbconvert               5.3.1
nbformat                4.4.0
networkx                2.1
nose                    1.3.7
numpy                   1.19.4
packaging               20.4
pandas                  1.1.4
pandocfilters           1.4.2
parso                   0.1.1
patsy                   0.5.1
pbr                     3.1.1
pexpect                 4.4.0
phpserialize            1.3
pickleshare             0.7.4
Pillow                  6.2.0
pip                     20.2.4
plotly                  4.12.0
pluggy                  0.13.1
ply                     3.11
prompt-toolkit          2.0.1
protobuf                3.6.1
psutil                  5.4.3
psycopg2                2.8.6
ptyprocess              0.5.2
py                      1.9.0
py4j                    0.10.6
pycodestyle             2.3.1
pycosat                 0.6.3
pycparser               2.18
pydot                   1.4.1
Pygments                2.2.0
pykafka                 2.8.0
pylint                  1.8.2
pymongo                 3.11.0
PyMySQL                 0.10.1
pynliner                0.8.0
pyodps                  0.9.3.1
pyOpenSSL               17.5.0
pyparsing               2.2.0
pypng                   0.0.20
pyproj                  3.0.0.post1
PyQRCode                1.2.1
pytest                  6.1.2
python-dateutil         2.8.1
pytz                    2020.4
PyWavelets              0.5.2
PyYAML                  3.12
pyzmq                   17.0.0
qtconsole               4.3.1
redis                   3.2.1
requests                2.25.0
retrying                1.3.3
rope                    0.10.7
ruamel.yaml             0.16.12
ruamel.yaml.clib        0.2.2
s3transfer              0.2.0
scikit-image            0.14.0
scikit-learn            0.20.3
scipy                   1.5.4
seaborn                 0.11.0
Send2Trash              1.5.0
setuptools              41.0.0
Shapely                 1.7.1
simplegeneric           0.8.1
six                     1.15.0
sklearn2                0.0.13
smart-open              1.8.1
SQLAlchemy              1.3.20
statsmodels             0.12.1
tabulate                0.8.7
testpath                0.3.1
thriftpy                0.3.9
timeout-decorator       0.4.1
```

```
toml                          0.10.2
toolz                         0.9.0
tornado                       6.1
tqdm                          4.32.2
traitlets                     4.3.2
urllib3                       1.24.3
wcwidth                       0.2.5
webencodings                  0.5.1
wheel                         0.35.1
wrapt                         1.11.1
xgboost                       1.2.1
xlrd                          1.2.0
XlsxWriter                    1.0.7
zipp                          3.4.0
```

- 上传单个WHEEL包

  如果依赖较为简单，则可以只上传单个WHEEL包，通常需要选用manylinux版本。使用方式如下：

  i. 将WHEEL包重命名为ZIP包，例如将pymysql的WHEEL包重命名为pymysql.zip。

  ii. 将重命名后的ZIP包上传，文件类型为ARCHIVE。

  iii. 在DataWorks Spark节点引用，文件类型为ARCHIVE。

  iv. 在代码中修改环境变量后即可导入。

  ```
  sys.path.append('pymysql')
  import pymysql
  ```

- 利用脚本一键打包

  若需要的额外依赖较多，上传单个WHEEL包会导致重复操作量倍增。您可以下载脚本，只需提供一个编辑好的requirements文件，就能够直接生成完整的Python环境用于PySpark使用，具体如下。

  - **使用**

    ```
    $ chmod +x generate_env_pyspark.sh
    $ generate_env_pyspark.sh -h
    Usage:
    generate_env_pyspark.sh [-p] [-r] [-t] [-c] [-h]
    Description:
    -p ARG, the version of python, currently supports python 2.7, 3.5, 3.6 and 3.7 versions.
    -r ARG, the local path of your python requirements.
    -t ARG, the output directory of the gz compressed package.
    -c, clean mode, we will only package python according to your requirements, without other pre-provided dependencies.
    -h, display help of this script.
    ```

  - **示例**

    ```
    # 带有预装依赖的打包方式
    $ generate_env_pyspark.sh -p 3.7 -r your_path_to_requirements -t your_output_directory
    # 不带预装依赖的打包方式（clean mode）
    generate_env_pyspark.sh -p 3.7 -r your_path_to_requirements -t your_output_directory -c
    ```

  - **说明**

    - 脚本适用于Mac或Linux环境，需要预先安装Docker，安装指导请参见Docker帮助文档。
    - 目前仅支持python 2.7、3.5、3.6和3.7版本，如果对Python版本不敏感，推荐使用Python 3.7。
    - `-c` 选项表示是否开启clean mode。clean mode无法使用预装依赖，但输出的Python包更小。各版本的依赖请参见Python 2.7预装依赖、Python 3.5预装依赖、Python 3.6预装依赖、Python 3.7预装依赖。
    - 当前MaxCompute对上传资源的大小有500 MB的限制，因此如果大部分预装依赖用不到，推荐使用 `-c` 选项打包。

- **Spark中使用**

  generate_env_pyspark.sh脚本的输出为在指定目录下（ `-t` 选项）生成指定Python版本（ `-p` 选项）的GZ包。以Python3.7为例，将生成py37.tar.gz，后续再将此包上传为ARCHIVE资源。您可以通过MaxCompute客户端上传，也可以使用odps-sdk上传。各种资源操作，请参见资源操作。以MaxCompute客户端为例，使用方式如下。

  a. 在MaxCompute客户端中执行如下命令添加资源。

  ```
  add archive /your/path/to/py37.tar.gz -f;
  ```

  b. 在Spark配置中增加如下两个参数。

  ```
  spark.hadoop.odps.cupid.resources = your_project.py37.tar.gz
  spark.pyspark.python = your_project.py37.tar.gz/bin/python
  ```

  若上述两个参数不生效，还需在Spark作业中增加如下两项配置。例如使用zeppelin调试Pyspark时，notebook中的Python环境配置。

  ```
  spark.yarn.appMasterEnv.PYTHONPATH = ./your_project.py37.tar.gz/bin/python
  spark.executorEnv.PYTHONPATH = ./your_project.py37.tar.gz/bin/python
  ```

- 利用Docker容器打包Python环境

  该方式适用于如下场景：

  - 需要引入的依赖包含so文件时，无法通过上述ZIP文件的方式使用，无法进行 `pip install` 安装。
  - 对除2.7、3.5、3.6、3.7以外的Python版本有特殊需求。

  针对以上特殊情况，同时保证打包环境与线上环境一致（Mac打出来的Python环境与线上环境存在兼容性问题）。以Python3.7为例，基于Docker的打包步骤如下。

  i. 在安装Docker环境的宿主机新建一个Dockerfile文件。

  - Python 3示例参考如下。

  ```
  FROM centos:7.6.1810
  RUN set -ex \
      # 预安装所需组件
      && yum install -y wget tar libffi-devel zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-dev
  el tk-devel gcc make initscripts zip\
      && wget https://www.python.org/ftp/python/3.7.0/Python-3.7.0.tgz \
      && tar -zxvf Python-3.7.0.tgz \
      && cd Python-3.7.0 \
      && ./configure prefix=/usr/local/python3 \
      && make \
      && make install \
      && make clean \
      && rm -rf /Python-3.7.0* \
      && yum install -y epel-release \
      && yum install -y python-pip
  # 设置默认为python3
  RUN set -ex \
      # 备份旧版本python
      && mv /usr/bin/python /usr/bin/python27 \
      && mv /usr/bin/pip /usr/bin/pip-python27 \
      # 配置默认为python3
      && ln -s /usr/local/python3/bin/python3.7 /usr/bin/python \
      && ln -s /usr/local/python3/bin/pip3 /usr/bin/pip
  # 修复因修改python版本导致yum失效问题
  RUN set -ex \
      && sed -i "s#/usr/bin/python#/usr/bin/python27#" /usr/bin/yum \
      && sed -i "s#/usr/bin/python#/usr/bin/python27#" /usr/libexec/urlgrabber-ext-down \
      && yum install -y deltarpm
  # 更新pip版本
  RUN pip install --upgrade pip
  ```

- Python 2示例参考如下。

```
FROM centos:7.6.1810
RUN set -ex \
    # 预安装所需组件
    && yum install -y wget tar libffi-devel zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-dev
el tk-devel gcc make initscripts zip\
    && wget https://www.python.org/ftp/python/2.7.18/Python-2.7.18.tgz \
    && tar -zxvf Python-2.7.18.tgz \
    && cd Python-2.7.18 \
    && ./configure prefix=/usr/local/python2 \
    && make \
    && make install \
    && make clean \
    && rm -rf /Python-2.7.18*
# 设置默认为python
RUN set -ex \
    && mv /usr/bin/python /usr/bin/python27 \
    && ln -s /usr/local/python2/bin/python /usr/bin/python
RUN set -ex \
    && wget https://bootstrap.pypa.io/get-pip.py \
    && python get-pip.py
RUN set -ex \
    && rm -rf /usr/bin/pip \
    && ln -s /usr/local/python2/bin/pip /usr/bin/pip
# 修复因修改python版本导致yum失效问题
RUN set -ex \
    && sed -i "s#/usr/bin/python#/usr/bin/python27#" /usr/bin/yum \
    && sed -i "s#/usr/bin/python#/usr/bin/python27#" /usr/libexec/urlgrabber-ext-down \
    && yum install -y deltarpm
# 更新pip版本
RUN pip install --upgrade pip
```

ii. 构建镜像并运行容器。

```
# 在Dockerfile文件的目录下运行如下命令。
docker build -t python-centos:3.7
docker run -itd --name python3.7 python-centos:3.7
```

iii. 进入容器安装所需的Python依赖库。

```
docker attach python3.7
pip install [所需依赖库]
```

iv. 打包Python环境。

```
cd /usr/local/
zip -r python3.7.zip python3/
```

v. 拷贝容器中的Python环境到宿主机。

```
# 退出容器
ctrl+P+Q
# 在宿主机运行命令。
docker cp python3.7:/usr/local/python3.7.zip
```

vi. 上传Python3.7.zip包为MaxCompute资源。DataWorks最大只能上传50 MB的包，如果大于50 MB可以通过MaxCompute客户端上传，文件类型为ARCHIVE。上传资源操作，请参见添加资源。

```
add archive /path/to/python3.7.zip -f;
```

vii. 提交作业时只需要在spark-default.conf或DataWorks配置项中添加以下配置即可。

```
spark.hadoop.odps.cupid.resources=[project名称].python3.7.zip
spark.pyspark.python=./[project名].python3.7.zip/python3/bin/python3.7
```

> **说明** 通过Docker容器打包，如果遇到so包找不到的情况，则需要手动将so包放到Python环境中。一般so包在容器中都能找到，并在Spark作业中添加以下环境变量。
>
> ```
> spark.executorEnv.LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./[project名].python3.7.zip/python3/[创建的so包目录]
> spark.yarn.appMasterEnv.LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./[project名].python3.7.zip/python3/[创建的so包目录]
> ```

- 引用用户自定义的Python包

  通常情况下，用户需要使用自定义的Python文件，可以打包提交，这样避免了上传多个PY文件，步骤如下：

i. 将用户代码打包为ZIP包，需要在目录下自定义个一个空白的__init__.py。

ii. 将用户代码ZIP包以MaxCompute资源形式上传，并重命名，该资源在工作目录中将会被解压。

iii. 配置参数 `spark.executorEnv.PYTHONPATH=` 。

iv. 完成上述步骤，主Python文件就可以导入该目录下的Python文件。

8.6. Spark访问VPC实例

本文为您介绍如何使用Spark on MaxCompute访问阿里云VPC内的用户实例。

直接访问VPC

Spark on MaxCompute可以访问阿里云VPC内的云服务器ECS（Elastic Compute Service）、云数据库HBase（Hadoop Database）和云关系型数据库RDS（Relational Database Service）等实例，同时还可以访问自定义私有域名。

访问VPC实例时，在*spark-defaults.conf*或者*DataWorks*的配置文件中添加 `spark.hadoop.odps.cupid.vpc.domain.list` 参数，表明需要访问的一个或多个实例的网络情况。该参数值为JSON格式，配置时需要删除参数中多行文本之间的空格和换行符，合并JSON文本为一行。

访问不同实例时， `spark.hadoop.odps.cupid.vpc.domain.list` 参数取值请参见下文示例。您需要将示例中的RegionID、VPCID、实例域名和端口等替换为实际使用场景下的值。区域的RegionID请参见项目空间操作。

> 注意
> - 需要添加100.104.0.0/16网段至所访问的服务的白名单。
> - 如果RegionID为*cn-shanghai*或者*cn-beijing*，需要设置 `spark.hadoop.odps.cupid.smartnat.enable=true` 。
> - Spark on MaxCompute只支持访问本Region下某一个VPC的服务，不支持同时访问多个VPC的服务。
> - 不需要设置 `spark.hadoop.odps.cupid.eni.enable=true` 。

ENI专线访问VPC

相比直接访问VPC方式，ENI专线方式更加稳定，性能更好，且具备外网访问能力。

使用该方式时，您需要注意：

- ENI专线可以打通一个VPC，如果用户作业需要同时访问多个VPC，则可以将已经通过ENI专线打通的VPC与其他VPC之间再做打通即可。
- Spark作业运行所在MaxCompute项目的阿里云账号的UID与目标VPC所属的阿里云账号的UID必须一致，否则作业运行时会提示报错信息： `You are not allowed to use this vpc - vpc owner and project owner must be the same person` 。

操作流程如下：

1. 您需要将目标VPC的如下信息提供给MaxCompute技术支持团队：
   - VPC所属地域。例如北京。
   - User ID：MaxCompute项目所属阿里云账号的UID，同时也是目标VPC所属阿里云账号的UID。
   - VPC ID：即用户需要通过ENI专线打通的目标VPC ID。
   - VSwitchId：即目标VPC绑定的交换机ID，可以在VPC管理控制台界面查看，如果有多个从中选取一个即可。
   - SecurityGroupId：即目标VPC的安全组ID。用户需要在目标VPC下新建一个安全组，用于对MaxCompute Spark访问VPC服务时进行访问控制。安全组的创建操作，请参见创建安全组。
2. 对ENI进行授权。

   授权的目的在于允许MaxCompute在VPC内创建ENI网卡，以实现MaxCompute到用户VPC的连通。用户只要在阿里云账号登录态下单击授权即可。
3. 等待MaxCompute技术支持团队为您开通专线。
4. 配置安全组规则。

   在完成ENI专线开通后，用户还需要在待访问的服务中增加相关安全规则，授权代表MaxCompute的安全组，即上述第1步中提供的安全组能访问哪些服务的具体端口。例如9200、31000等。

   例如，用户需要访问阿里云RDS，则需要在RDS中增加规则，允许第1步中创建的安全组访问。如果用户需要访问的服务无法添加安全组，只能添加IP，那么需要将第1步中所使用的vSwitch网段都添加进来。
5. 配置用户作业。

   运行Spark作业，需要增加如下两个配置，即可使用ENI专线连通目标VPC内的服务。

   ```
   spark.hadoop.odps.cupid.eni.enable = true
   spark.hadoop.odps.cupid.eni.info = regionid:vpc id
   ```

   原先Spark作业中的VPC如下操作和配置不再需要：

   ```
   spark.hadoop.odps.cupid.vpc.domain.list
   spark.hadoop.odps.cupid.smartnat.enable
   spark.hadoop.odps.cupid.pvtz.rolearn    #（访问自定义域名）
   spark.hadoop.odps.cupid.vpc.usepvtz     #（访问自定义域名）
   ```

访问MongoDB示例

访问MongoDB时，`spark.hadoop.odps.cupid.vpc.domain.list`参数取值如下。该MongoDB有主备2个实例。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "vpcId":"vpc-2zeaeq21mb1dmkqh0****",
      "zones":[
        {
          "urls":[
            {
              "domain":"dds-2ze3230cfea08****.mongodb.rds.aliyuncs.com",
              "port": 3717
            },
            {
              "domain":"dds-2ze3230cfea08****.mongodb.rds.aliyuncs.com",
              "port":3717
            }
          ]
        }
      ]
    }
  ]
}
```

合并示例中JSON文本为一行的结果如下。

```
{"regionId":"cn-beijing","vpcs":[{"vpcId":"vpc-2zeaeq21mb1dmkqh0****","zones":[{"urls":[{"domain":"dds-2ze3230cfea08****.mongodb.
rds.aliyuncs.com","port": 3717},{"domain":"dds-2ze3230cfea08****.mongodb.rds.aliyuncs.com","port":3717}]}]}]}
```

访问RDS示例

访问RDS时，`spark.hadoop.odps.cupid.vpc.domain.list`参数取值如下。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "vpcId":"vpc-2zeaeq21mb1dmkqh0****",
      "zones":[
        {
          "urls":[
            {
              "domain":"rm-2zem49k73c54z****.mysql.rds.aliyuncs.com",
              "port": 3306
            }
          ]
        }
      ]
    }
  ]
}
```

合并示例中JSON文本为一行的结果如下。

```
{"regionId":"cn-beijing","vpcs":[{"vpcId":"vpc-2zeaeq21mb1dmkqh0****","zones":[{"urls":[{"domain":"rm-2zem49k73c54z****.mysql.rds
.aliyuncs.com","port": 3306}]}]}]}
```

访问HBase示例

访问HBase时，`spark.hadoop.odps.cupid.vpc.domain.list`参数取值如下。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "vpcId":"vpc-2zeaeq21mb1dmkqh0exox",
      "zones":[
        {
          "urls":[
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            }
          ]
        }
      ]
    }
  ]
}
```


合并示例中JSON文本为一行的结果如下。


```
{"regionId":"cn-beijing","vpcs":[{"vpcId":"vpc-2zeaeq21mb1dmkqh0exox","zones":[{"urls":[{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":2181},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":16000},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":16020},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":2181},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":16000},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":16020},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":2181},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":16000},{"domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com","port":16020},{"domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com","port":16020},{"domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com","port":16020},{"domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com","port":16020}]}]}]}
```

访问Redis示例

当访问Redis时，`spark.hadoop.odps.cupid.vpc.domain.list`参数取值如下。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "vpcId":"vpc-2zeaeq21mb1dmkqh0****",
      "zones":[
        {
          "urls":[
            {
              "domain":"r-2zebda0d3c05****.redis.rds.aliyuncs.com",
              "port":3717
            }
          ]
        }
      ]
    }
  ]
}
```

合并示例中JSON文本为一行的结果如下。

```
{"regionId":"cn-beijing","vpcs":[{"vpcId":"vpc-2zeaeq21mb1dmkqh0****","zones":[{"urls":[{"domain":"r-2zebda0d3c05****.redis.rds.aliyuncs.com","port":3717}]}]}]}
```

访问LogHub示例

访问LogHub时，`spark.hadoop.odps.cupid.vpc.domain.list`参数取值如下。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "zones":[
        {
          "urls":[
            {
              "domain":"cn-beijing-intranet.log.aliyuncs.com",
              "port":80
            }
          ]
        }
      ]
    }
  ]
}
```

合并示例中JSON文本为一行的结果如下。

```
{"regionId":"cn-beijing","vpcs":[{"zones":[{"urls":[{"domain":"cn-beijing-intranet.log.aliyuncs.com","port":80}]}]}]}
```

domain请使用LogHub Endpoint的经典网络或VPC网络服务入口，各Region对应的Endpoint请参见服务入口。

访问DataHub示例

访问DataHub时，`spark.hadoop.odps.cupid.vpc.domain.list`参数取值如下。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "zones":[
        {
          "urls":[
            {
              "domain":"dh-cn-beijing.aliyun-inc.com",
              "port":80
            }
          ]
        }
      ]
    }
  ]
}
```

合并示例中JSON文本为一行的结果如下。

```
{"regionId":"cn-beijing","vpcs":[{"zones":[{"urls":[{"domain":"dh-cn-beijing.aliyun-inc.com","port":80}]}]}]}
```

domain请使用DataHub Endpoint的经典网络下的ECS Endpoint。Region对应的Endpoint请参见域名列表。

访问自定义域名示例

假设您在VPC内自定义了域名 `example.aliyundoc.com` ，Spark通过域名和端口 `example.aliyundoc.com:80` 发起访问。您在使用前需要完成以下配置：

1. 在PrivateZone中将Zone关联至VPC。
2. 单击一键授权，为MaxCompute赋予PrivateZone的只读权限。
3. 在Spark节点配置里新增如下2个参数。

   ```
   spark.hadoop.odps.cupid.pvtz.rolearn=acs:ram::xxxxxxxxxx:role/aliyunodpsdefaultrole
   spark.hadoop.odps.cupid.vpc.usepvtz=true
   ```

   `spark.hadoop.odps.cupid.pvtz.rolearn` 为用户的ARN信息，从RAM控制台可以获取。
4. 在Spark的配置文件中新增 `spark.hadoop.odps.cupid.vpc.domain.list` 参数，取值如下。

   ```
   {
     "regionId":"cn-beijing",
     "vpcs":[
       {
         "vpcId":"vpc-2zeaeq21mb1dmkqh0****",
         "zones":[
           {
             "urls":[
               {
                 "domain":"example.aliyundoc.com",
                 "port":80
               }
             ],
             "zoneId":"9b7ce89c6a6090e114e0f7c415ed****"
           }
         ]
       }
     ]
   }
   ```

   合并示例中JSON文本为一行的结果如下。

   ```
   {"regionId":"cn-beijing","vpcs":[{"vpcId":"vpc-2zeaeq21mb1dmkqh0****","zones":[{"urls":[{"domain":"example.aliyundoc.com","po
   rt":80}],"zoneId":"9b7ce89c6a6090e114e0f7c415ed****"}]}]}
   ```

访问文件存储HDFS示例

文件存储HDFS产品的使用，请参见开通文件存储HDFS版服务。

- 在Spark中使用文件存储HDFS需要新增*hdfs-site.xml*，内容如下所示。

```
<?xml version="1.0"?>
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>dfs://DfsMountpointDomainName:10290</value>
    </property>
    <property>
        <name>fs.dfs.impl</name>
        <value>com.alibaba.dfs.DistributedFileSystem</value>
    </property>
    <property>
        <name>fs.AbstractFileSystem.dfs.impl</name>
        <value>com.alibaba.dfs.DFS</value>
    </property>
</configuration>
```

- 在Spark配置文件中新增 `spark.hadoop.odps.cupid.vpc.domain.list` 参数，取值如下。

```
{
    "regionId": "cn-shanghai",
    "vpcs": [{
        "vpcId": "vpc-xxxxxx",
        "zones": [{
            "urls": [{
                "domain": "DfsMountpointDomainName",
                "port": 10290
            }]
        }]
    }]
}
```

合并示例中JSON文本为一行的结果如下。

```
{"regionId": "cn-shanghai","vpcs": [{"vpcId": "vpc-xxxxxx","zones": [{"urls": [{"domain": "DfsMountpointDomainName","port": 10290}]}]}]}
```

8.7. Spark访问OSS

本文为您介绍使用Spark访问OSS时需要的相关配置。

OSS Endpoint配置

调试时请使用OSS服务所在地域的外网Endpoint，提交集群需替换为VPC内网Endpoint。详情请参见访问域名和数据中心。

OSS访问方式配置

- 以AccessKey ID和AccessKey Secret方式访问OSS。

```
spark.hadoop.fs.oss.accessKeyId = xxxxxx
spark.hadoop.fs.oss.accessKeySecret = xxxxxx
spark.hadoop.fs.oss.endpoint = oss-xxxxxx-internal.aliyuncs.com
```

- 以StsToken的方式访问OSS。

  以AccessKey ID和AccessKey Secret方式访问OSS，需要明文将AccessKey ID和AccessKey Secret写在配置中，存在一定的安全风险。因此建议您以StsToken的方式访问OSS。

  i. 单击一键授权，将当前云账号的OSS资源通过StsToken的方式授权给MaxCompute项目直接访问。

  > **说明** 当MaxCompute的Project Owner为OSS云账号时，才可以执行一键授权。

  ii. 获取roleArn。

  a. 登录RAM控制台。

  b. 在左侧导航栏上，单击**RAM角色管理**。

  c. 在**RAM角色管理**页面，搜索AliyunODPSDefaultRole。

  d. 单击**AliyunODPSDefaultRole**，在**基本信息**区域获取**ARN**。格式为 `acs:ram::xxxxxxxxxxxxxx:role/aliyunodpsdefaultrole` 。

  iii. 在Spark配置中添加如下内容即可访问OSS资源。

```
# 此配置表明Spark是通过StsToken去访问OSS资源。
spark.hadoop.fs.oss.credentials.provider=org.apache.hadoop.fs.aliyun.oss.AliyunStsTokenCredentialsProvider
# 此配置是一键授权后产生的一个roleArn。
spark.hadoop.fs.oss.ststoken.roleArn=acs:ram::xxxxxxxxxxxxxx:role/aliyunodpsdefaultrole
# 此配置是OSS资源对应的VPC访问Endpoint。
spark.hadoop.fs.oss.endpoint=oss-cn-hangzhou-internal.aliyuncs.com
```

网络白名单配置

默认情况下无需配置网络白名单即可访问OSS。

特殊情况下（例如用户的OSS Bucket为大流量类型），如果无法访问OSS，需要配置如下信息。

```
spark.hadoop.odps.cupid.trusted.services.access.list=[your_bucket_name].oss-xxxxx-internal.aliyuncs.com
```

> 说明 该配置在yarn-cluster模式使用，且该配置项必须放在配置文件或命令行提交参数中。

8.8. 作业诊断

本文为您介绍如何通过作业日志检查作业是否已正常提交并执行。MaxCompute为Spark作业提供Logview工具以及Spark Web-UI，帮助您诊断作业。

背景信息

通过Spark-submit方式（DataWorks执行Spark任务时也会产生相应日志）提交作业。

```
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class  SparkPi /tmp/spark-2.x-demo/target/AliSpark-2.x-quickstart-1.0-SNAPSHOT-shaded.jar
```

作业提交成功后，MaxCompute会创建一个Instance，并在日志中打印Instance的Logview和Tracking URL。

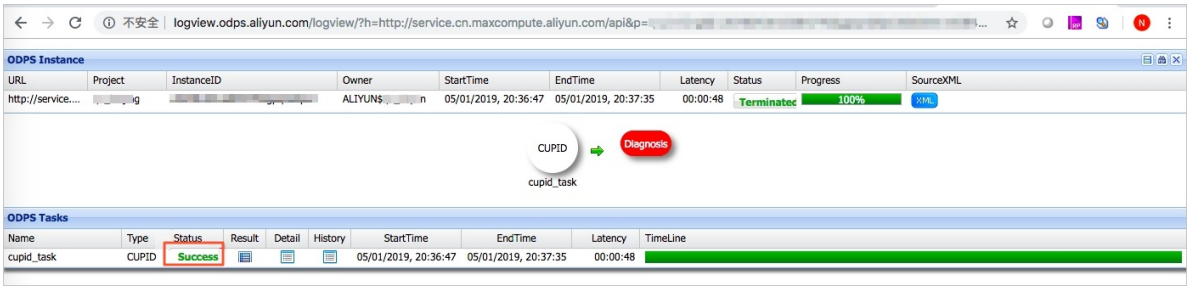
```
19/01/05 20:36:47 INFO YarnClientImplUtil: logview url: http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/api&p=qn_beijing&i=xxx&token=xxx
成功标准：<看到以下输出，可能会有其他日志一并输出>
19/01/05 20:37:34 INFO Client:
   client token: N/A
   diagnostics: N/A
   ApplicationMaster host: 11.220.xxx.xxx
   ApplicationMaster RPC port: 30002
   queue: queue
   start time: 1546691807945
   final status: SUCCEEDED
   tracking URL: http://jobview.odps.aliyun.com/proxyview/jobview/?h=http://service.cn.maxcompute.aliyun-inc.com/api&p=project_name&i=xxx&t=spark&id=application_xxx&metaname=xxx&token=xxx
```

使用Logview工具诊断作业

以 `logview.odps.aliyun.com` 开头的URL，统称为Logview。Logview是MaxCompute自研的分布式作业追踪工具。该工具的功能如下：

- 获取作业的状态。
- 获取作业各节点的起停调度信息。
- 获取作业各节点的标准输入输出日志。Spark结果输出建议打印到StdOut，Spark的log4j日志则默认输出到StdErr。
- Logview的时效性为3～5天。当本地磁盘已满时，StdOut和StdErr会被清理掉。

1. 在浏览器中打开日志输出的Logview，查看CUPID类型作业的运行情况。



2. 在Logview页面，执行如下操作。
   i. 单击**Detail**，查看作业的详情。**master-0**代表Spark Driver所在节点。
   ii. 单击**master-0**并选择All Tab，查看Spark Driver节点的具体信息。

iii. 单击**StdOut**，查看节点的结果输出。

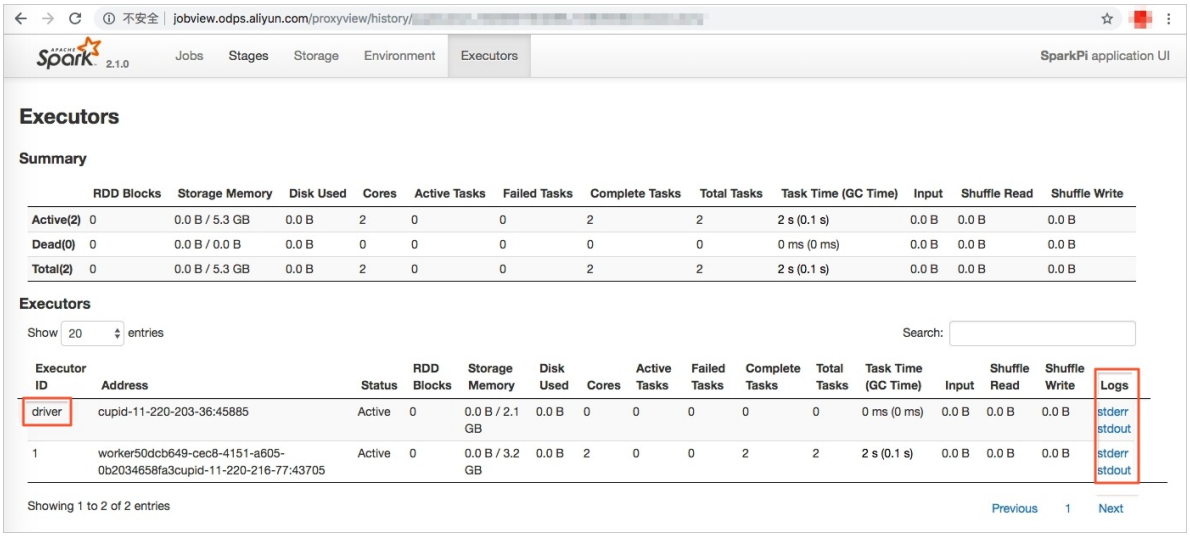
iv. 单击**StdErr**，查看节点的log4j日志。

使用Spark-Web-UI诊断作业

日志输出的Tracking URL，表示您的作业已经提交到MaxCompute集群。Tracking URL是Spark Web-UI和HistoryServer的URL。该工具的功能如下：

- 获取原生Spark-Web-UI的所有信息。
- 作业运行时，此页面显示作业运行的实时信息。
- 作业运行结束后，把事件从驱动传递到HistoryServer有1～3分钟的延迟。作业运行结束，立刻打开Tracking URL时，可能会出现一个页面，显示`Application application_1560240626712_2078769635 not found.`错误。请稍后重试即正常。

1. 在浏览器中打开日志输出的Tracking URL，查看Spark作业的运行情况。



2. 在Tracking URL页面，执行如下操作。

i. 单击**Environment**，确认设置的Spark参数是否全部正确。

ii. 单击**Executors**，查看是否有Dead节点，以及Spark Driver的StdOut和StdErr。

iii. 单击**Stdout**，查看节点的结果输出。

iv. 单击**Stderr**，查看节点的log4j日志。

8.9. Spark常见问题

本文为您介绍使用Spark过程中的常见问题。

问题类别	常见问题
开发Spark	• 如何自检项目工程？ • 在DataWorks上运行ODPS Spark节点的步骤是什么？ • MaxCompute Spark如何在本地进行调试？ • 如何通过Spark访问VPC环境内的服务？ • 如何把JAR包当成资源来引用？ • 如何通过Spark传入参数？ • 如何将Spark流式读取的DataHub数据写入MaxCompute？ • 如何将开源Spark代码迁移至MaxCompute Spark？ • 如何通过Spark处理MaxCompute中的表数据？

问题类别	常见问题
作业报错	• 运行Spark作业时，报错User signature dose not match，如何解决？ • 运行Spark作业时，报错You have NO privilege，如何解决？ • 运行Spark作业时，报错Access Denied，如何解决？ • 运行Spark作业时，报错No space left on device，如何解决？ • 运行Spark作业时，报错Table or view not found，如何解决？ • 运行Spark作业时，报错Shutdown hook called before final status was reported，如何解决？ • 运行Spark作业时，发生JAR包版本冲突类错误，如何解决？ • 运行Spark作业时，报错ClassNotFound，如何解决？ • 运行Spark作业时，报错The task is not in release range，如何解决？ • 运行Spark作业时，报错java.io.UTFDataFormatException，如何解决？ • 运行Spark作业时，打印的中文乱码，如何解决？ • Spark调用外网第三方任务时报错，如何解决？

如何自检项目工程？

建议您检查如下内容：

- 检查*pom.xml*。

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope> // spark-xxxx_${scala.binary.version} 依赖scope必须是provided。
</dependency>
```

- 检查主类spark.master。

```
val spark = SparkSession
      .builder()
      .appName("SparkPi")
      .config("spark.master", "local[4]") // 如果是以yarn-cluster方式提交，代码中如果有local[N]的配置，将会报错。
      .getOrCreate()
```

- 检查主类Scala代码。

```
object SparkPi { // 必须是object，如果在IDEA创建文件的时候写为class，main函数是无法加载的。
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("SparkPi")
      .getOrCreate()
```

- 检查主类代码配置。

```
val spark = SparkSession
      .builder()
      .appName("SparkPi")
      .config("key1", "value1")
      .config("key2", "value2")
      .config("key3", "value3")
      ...  // 如果执行local测试时，将MaxCompute配置在hard-code代码里，部分配置是无法生效的。
      .getOrCreate()
```

> **说明** 建议您在使用yarn-cluster方式提交任务时，将配置项都写在spark-defaults.conf中。

在DataWorks上运行ODPS Spark节点的步骤是什么？

1. 在本地Python环境中编辑Spark代码并打包。本地Python环境版本要求为Python 2.7。
2. 上传资源包至DataWorks。详情请参见创建MaxCompute资源。
3. 在DataWorks上创建ODPS Spark节点。详情请参见创建ODPS Spark节点。
4. 编写代码并运行节点，在DataWorks控制台上即可查看运行结果。

MaxCompute Spark如何在本地进行调试？

您可以通过Intellij IDEA在本地进行调试。详情请参见搭建开发环境。

如何通过Spark访问VPC环境内的服务？

更多通过Spark访问VPC环境内的服务信息，请参见Spark访问VPC实例。

如何把JAR包当成资源来引用？

您可以通过参数 `spark.hadoop.odps.cupid.resources` 指定需要引用的资源。资源可以多个项目共享，建议您设置相关权限确保数据安全。示例如下。

```
spark.hadoop.odps.cupid.resources = projectname.xx0.jar,projectname.xx1.jar
```

如何通过Spark传入参数？

传参详情请参见Spark on Dataworks。

如何将Spark流式读取的DataHub数据写入MaxCompute？

示例代码请参见DataHub。

如何将开源Spark代码迁移至MaxCompute Spark？

您可以根据作业场景选择的迁移方案如下：

- 作业无需访问MaxCompute表和OSS。

  您可以直接运行已有JAR包，详情请参见搭建开发环境。对于Spark或Hadoop的依赖必须设置为provided。
- 作业需要访问MaxCompute表。

  配置相关依赖后重新打包即可，详情请参见搭建开发环境。
- 访问OSS所需要的包，打通网络请参见Spark访问VPC实例。

  配置相关依赖后重新打包即可，详情请参见搭建开发环境。

如何通过Spark处理MaxCompute中的表数据？

MaxCompute Spark支持Local、Cluster和DataWorks运行模式。三种模式的配置不同，详情请参见运行模式。

运行Spark作业时，报错User signature dose not match，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  Stack:
  com.aliyun.odps.OdpsException: ODPS-0410042:
  Invalid signature value - User signature dose not match
  ```
- 产生原因

  身份验证未通过，AccessKey ID或AccessKey Secret有误。
- 解决措施

  请检查*spark-defaults.conf*提供的AccessKey ID、AccessKey Secret和阿里云官网控制台用户信息管理中的**AccessKey ID**、**AccessKey Secret**是否一致，如果不一致，请修改一致。

运行Spark作业时，报错You have NO privilege，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  Stack:
  com.aliyun.odps.OdpsException: ODPS-0420095:
  Access Denied - Authorization Failed [4019], You have NO privilege 'odps:CreateResource' on {acs:odps:*:projects/*}
  ```
- 产生原因

  权限不足，需要申请权限。
- 解决措施

  需要由项目所有者授予Resource的Read和Create权限。更多授权信息，请参见权限列表。

运行Spark作业时，报错Access Denied，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  Exception in thread "main" org.apache.hadoop.yarn.exceptions.YarnException: com.aliyun.odps.OdpsException: ODPS-0420095: Acces
  s Denied - The task is not in release range: CUPID
  ```
- 产生原因
  - 原因一：*Spark-defaults.conf*中配置的AccessKey ID、AccessKey Secret不正确。

- 原因二：项目所在的地域未提供MaxCompute Spark服务。

- 解决措施
  - 原因一的解决措施：检查*Spark-defaults.conf*配置信息，修改为正确的AccessKey ID、AccessKey Secret。更多信息，请参见搭建开发环境。
  - 原因二的解决措施：确认项目所在的地域是否已经提供了MaxCompute Spark服务或加入钉钉群21969532（MaxCompute Spark支持群）咨询。

运行Spark作业时，报错No space left on device，如何解决？

Spark使用网盘进行本地存储。Shuffle数据和BlockManager溢出的数据均存储在网盘上。网盘的大小通过spark.hadoop.odps.cupid.disk.driver.device_size参数控制，默认为20 GB，最大为100 GB。如果调整到100 GB仍然报此错误，需要分析具体原因。常见原因为数据倾斜，在Shuffle或者Cache过程中数据集中分布在某些Block。此时可以缩小单个Executor的核数（spark.executor.cores），增加Executor的数量（spark.executor.instances）。

运行Spark作业时，报错Table or view not found，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  Table or view not found:xxx
  ```

- 产生原因
  - 原因一：表或视图不存在。
  - 原因二：打开了Hive的catalog配置。

- 解决措施
  - 原因一的解决措施：请创建表。
  - 原因二的解决措施：去掉catalog配置。报错示例如下，需要去掉`enableHiveSupport()`。

    ```
    spark = SparkSession.builder.appName(app_name).enableHiveSupport().getOrCreate()
    ```

运行Spark作业时，报错Shutdown hook called before final status was reported，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  App Status: SUCCEEDED, diagnostics: Shutdown hook called before final status was reported.
  ```

- 产生原因

  提交到集群的user main并没有通过AM（ApplicationMaster）申请集群资源。例如，用户没有新建SparkContext或用户在代码中设置spark.master为local。

运行Spark作业时，发生JAR包版本冲突类错误，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  User class threw exception: java.lang.NoSuchMethodError
  ```

- 产生原因

  JAR包版本冲突或类错误。

- 解决措施

  i. 在*$SPARK_HOME/jars*路径下找出异常类所在的JAR。

  ii. 执行如下命令定位第三方库的坐标以及版本。

  ```
  grep <异常类类名> $SPARK_HOME/jars/*.jar
  ```

  iii. 在Spark作业根目录下，执行如下命令查看整个工程的所有依赖。

  ```
  mvn dependency:tree
  ```

  iv. 找到对应的依赖后，执行如下命令排除冲突包。

  ```
  maven dependency exclusions
  ```

  v. 重新编译并提交代码。

运行Spark作业时，报错ClassNotFound，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  java.lang.ClassNotFoundException: xxxx.xxx.xxxxx
  ```

- 产生原因

  类不存在或依赖配置错误。

- 解决措施

  i. 执行如下命令查看您提交的JAR包中是否存在该类定义。

  ```
  jar -tf <作业JAR包> | grep <类名称>
  ```

  ii. 检查*pom.xml*文件中的依赖是否正确。

  iii. 使用Shade方式提交JAR包。

运行Spark作业时，报错The task is not in release range，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  The task is not in release range: CUPID
  ```

- 产生原因

  项目所在地域未开通MaxCompute Spark服务。

- 解决措施

  请您提工单，开启该地域的MaxCompute Spark服务。

运行Spark作业时，报错java.io.UTFDataFormatException，如何解决？

- 问题现象

  运行Spark作业时，返回报错如下。

  ```
  java.io.UTFDataFormatException: encoded string too long: 2818545 bytes
  ```

- 解决措施

  调整*spark-defaults.conf*中spark.hadoop.odps.cupid.disk.driver.device_size参数的值。默认为20 GB，最大支持100 GB。

运行Spark作业时，打印的中文乱码，如何解决？

您需要添加如下配置。

```
"--conf" "spark.executor.extraJavaOptions=-Dfile.encoding=UTF-8 -Dsun.jnu.encoding=UTF-8"
"--conf" "spark.driver.extraJavaOptions=-Dfile.encoding=UTF-8 -Dsun.jnu.encoding=UTF-8"
```

Spark调用外网第三方任务时报错，如何解决？

Spark不能调用外网第三方任务，网络不通。

您可以在VPC中搭建Nginx反向代理，通过代理访问外网。Spark支持直接访问VPC，详情请参见Spark访问VPC实例。

9.查询加速（MCQA）

9.1. 概述

本文为您介绍MaxCompute查询加速MCQA（MaxCompute Query Acceleration）功能，并帮助您了解该功能的系统架构、关键特性、应用场景和使用限制。

功能介绍

MaxCompute的MCQA功能可以对中、小数据量查询作业进行加速优化，将执行时间为分钟级的查询作业缩减至秒级，同时完全兼容原MaxCompute的查询功能。

MCQA支持将主流BI工具或SQL客户端连接至MaxCompute项目，开展即席查询（Ad Hoc）或商业智能（BI）分析。

MCQA使用独立的资源池，不占用配额组，可以自动识别查询作业，缓解排队压力，优化使用体验。

加速后的查询作业计费方式和SQL作业的按量计费方式保持一致，详情请参见MCQA（查询加速SQL）按量计费。

MaxCompute为使用包年包月资源的SQL作业提供了查询加速（MCQA）免费体验功能，请参见免费体验。

MaxCompute还支持将MCQA（MaxCompute Query Acceleration）查询作业的运行结果写入临时缓存中。当用户后续执行相同的查询作业时，MaxCompute会优先返回缓存中的结果，加快执行速度。更多查询结果缓存信息，请参见增强：查询结果缓存机制。

> **说明** 查询作业回退为普通SQL作业后，按原SQL作业计费模式计费。

关键特性

MCQA功能的关键特性如下：

- 低延时

  采用高效、低延时的资源调度策略，并且使用单独的资源池。

- 自动识别

  支持自动识别查询作业大小，优先快速返回查询结果或通过批处理返回结果，为分析不同规模或复杂度的查询作业提供保障。

- 语法兼容

  SQL语法与原MaxCompute SQL语法完全一致。

- 支持选择查询方式

  支持客户端选择MCQA或批处理方式执行查询作业，您也可以强制指定使用MCQA适应对时延敏感的场景。

应用场景

MCQA功能的应用场景如下。

场景	说明	场景特点
即席查询（Ad Hoc）	您可以通过MCQA优化中小规模数据集（百GB规模内）的查询性能，直接对MaxCompute表开展低时延的查询操作，以便快速完成数据开发及数据分析。	您可以根据实际需求，灵活选择查询条件，快速获取查询结果并调整查询逻辑。对查询时延的要求在几十秒内。使用者通常为掌握SQL技能的数据开发或数据分析师，希望使用熟悉的客户端工具开展查询分析。
商业智能（BI）	利用MaxCompute搭建企业级数据仓库时，ETL会将数据加工处理为面向业务可消费的聚合数据。借助MCQA的低延时、弹性并发、数据缓存等特性，结合MaxCompute表分区、分桶等优化设计，可以低成本满足多并发、快速响应的报告生成、统计分析及固定报表分析需求。	查询的数据对象通常为聚合后的结果数据，适用于数据量较小、多维查询、固定查询、高频查询场景。查询延时要求高，秒级返回（例如大部分查询不超过5秒，不同查询作业由于数据规模和查询复杂度不同，查询时间有较大差异）。
海量数据明细查询分析	MCQA可以自动识别查询作业特征，既能快速响应，处理小规模作业，同时还可以自动匹配大规模作业资源需求，满足分析人员分析不同规模和复杂度的查询作业的需求。	需要探索的历史数据量大、真正需要的有效数据量不大、查询延时要求适中。使用者通常为业务分析人员，往往需要从明细数据中探寻业务规律，发现业务机会，验证业务假设。

使用限制

仅支持SELECT语句。如果您提交了MCQA不支持的语句或功能，MaxCompute客户端（0.35.1及以上版本）会自动回退到普通离线模式执行，ODPS-JDBC（3.2.3及以上版本）和ODPS-SDK（0.35.1及以上版本）可以通过配置自动回退到普通离线模式执行，其他工具暂不支持回退到普通离线模式执行。

MCQA功能的详细使用限制如下。

限制项	说明

限制项	说明
功能	• MCQA功能支持标准版、按量计费的MaxCompute服务。 • MCQA功能支持包年包月计费模式，提供免费体验功能，请参见包年包月规格下使用查询加速（邀测）和免费体验。 • MCQA功能不支持开发者版的MaxCompute服务，需要升级为标准版。
查询	• 单次执行的作业支持并发的Worker数上限为2000个。 • 您通过客户端提交的MCQA作业默认30s超时，通过DataWorks临时查询提交的MCQA作业默认20s超时，超时后MCQA作业将默认回退为普通查询作业。 • 仅存储格式为ALIORC的表的数据才可能被缓存至内存中进行加速。 • 不支持查询外部表。
查询并发	单个MaxCompute项目的MCQA作业最大并发数为120。

9.2. 包年包月规格下使用查询加速（邀测）

MaxCompute产品规格使用的是包年包月，可以通过创建查询加速独享资源配额，实现包年包月产品规格下对应项目的查询加速（MCQA）支持。本文为您介绍如何使用查询加速对MaxCompute包年包月规格实例下的项目进行加速。

背景信息

当前MaxCompute包年包月规格下使用查询加速功能还在公测中，如果您需要使用的话，请提交新功能测试申请，请参见新功能测试申请。

使用限制

- 单次执行的作业并支持并发的Worker数上限为2000个，超出系统会将MCQA作业默认回退为普通查询作业。
- 配制交互式资源类型的配额组，用以查询加速作业的预留CU最小配额需和预留CU最大配额值一致，否则不生效。
- 交互式资源类型预留CU最小配额需与预留CU最大配额相等，且需大于等于50CU生效，如果小于50CU作业不支持提交到交互式资源Quota执行。配制了交互式资源Quota后，所有Project仅支持可识别为加速的作业提前到交互式资源配额，不再享有免费体验功能。
- 其他限制请参见使用限制。

使用查询加速

使用查询加速功能加速包年包月MaxCompute实例的项目步骤如下。

1. 进入MaxCompute控制台**管家**页面，详情请参见进入MaxCompute管家。
2. 在左侧导航栏单击**配额**。
3. 设置分时。
   i. 在配额管理功能界面右上角单击**设置分时**。
   ii. 在**设置分时**对话框设置三个分时时间段。
   三个分时时间段设置为：
   - 00:00:00 - 07:00:00
   - 07:00:00 - 18:00:00
   - 18:00:00 - 23:59:59

   如下图所示。



   iii. 单击**开启分时**，在**确定要开启分时吗？**对话框单击**确定**，完成分时设置。
4. 新建交互式资源类型配额组。
   交互式资源类型不支持弹性，预留CU最小配额需和预留CU最大配额一致。
   i. 在配额管理功能界面右上角单击**新建配额组**。

ii. 在**新建配额组**对话框配置如下参数。

参数	说明
配额组名称	自定义输入配额组名称。
资源类型	配额组的类型，有如下取值。 ■ 批处理。 ■ 交互式。
预留CU配额	包年包月里的预留计算资源。详情请参见计算费用（包年包月）。可调整每个分时段的预留CU最小配额和预留CU最大配额。
标签	自定义输入标签名称。标签名称尽量不要重名，否则作业会随机调度到其中一个配额组，而不是均匀分布在多个配额组。



iii. 单击**执行**，完成新建交互式资源类型配额组。

5. 使用查询加速

当您完成包年包月产品规格下交互式资源配额组创建后，您提交的作业即可被识别为被加速的作业，会提交到新建的交互式配额组执行。

回退策略

如果因使用限制发生查询加速作业回退为普通查询作业，包年包月规格下配额回退为当前Project绑定的配额资源（Quota）。

9.3. 增强：查询结果缓存机制

MaxCompute支持将MCQA（MaxCompute Query Acceleration）查询作业的运行结果写入临时缓存中。当用户后续执行相同的查询作业时，MaxCompute会优先返回缓存中的结果，加快执行速度。本文为您介绍MCQA查询结果缓存机制、限制、计费规则并指导您如何验证查询作业是否使用了缓存机制。

缓存机制

对于每个MCQA查询作业，MaxCompute会在内部创建临时数据集来缓存查询结果。临时数据集的所有者即运行查询作业生成了缓存结果的用户。临时数据集对用户不可见，不支持查看临时数据集内容。MaxCompute会自动为运行查询作业的用户授予对临时数据集的访问权限。

MaxCompute会缓存查询结果大约6个小时。出现如下情况时，MaxCompute会删除缓存结果：

- MaxCompute项目的资源使用率较高时，MaxCompute会提前删除缓存结果。
- 缓存结果所引用的表或视图变更后，缓存结果立即失效，MaxCompure会删除失效的缓存结果。
- 缓存结果已过期。

缓存限制

MCQA查询结果缓存的使用限制如下：

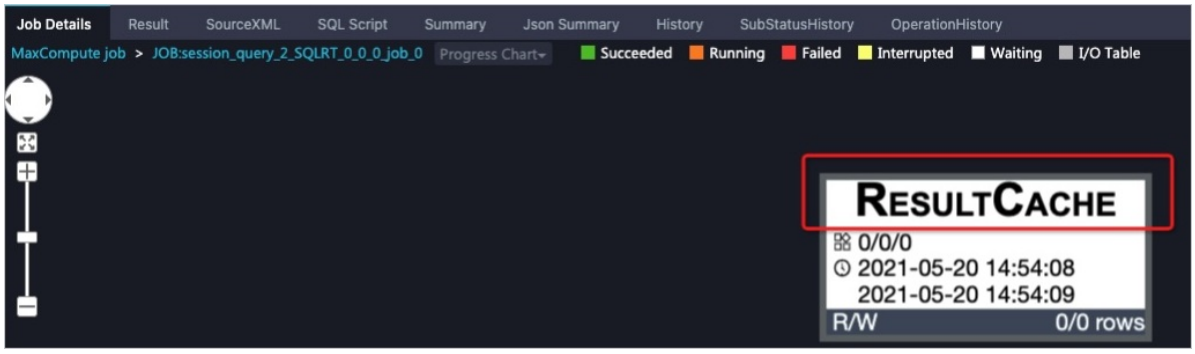
- 如果您希望从缓存的查询结果中检索数据，执行的查询作业以及查询的上下文配置必须与原始查询作业完全相同。当您运行重复的查询作业时，MaxCompute会重复使用缓存结果。
- 缓存的查询结果中所引用的表或视图变更后，缓存失效，再次执行相同的查询作业，不会检索到缓存数据。
- 临时数据集的缓存大小限制为10 GB。

缓存计费规则

缓存的查询结果不会产生任何存储及计算费用，可以有效降低资源使用费用。

缓存验证

您可以获取查询作业的Logview信息，并在**Job Details**页签，查看到查询作业运行结果已写入缓存中，如下图示所示。更多获取Logview信息，请参见使用Logview 2.0查看Job运行信息。



9.4. 免费体验

MaxCompute为使用包年包月资源的SQL作业提供了查询加速（MCQA）免费体验功能。本文为您介绍查询加速免费体验使用说明及使用限制。

免费体验说明

如果您已购买包年包月套餐，公测期间，无需进行额外操作，即可免费体验查询加速功能。该过程不生成费用账单，不会产生费用。

免费体验功能支持为使用包年包月资源的实例下SQL扫描量在10 GB以内的查询作业提供免费查询加速服务。该查询加速资源不占用您已购买的计算资源（CU），由查询加速公共资源池提供。

> 注意 MaxCompute对免费体验不提供任何SLO（服务等级目标）保障，如果您有更高负载要求，请关注后续提供的付费服务。

免费体验限制

免费体验功能针对使用包年包月资源的实例，以MaxCompute项目为单位进行限制：

- 工具限制：
  - 支持通过MaxCompute控制台的查询编辑器或DataWorks提交的临时查询作业，且SQL扫描量在10 GB以内的作业会被免费加速。不限制作业并发数及日累计免费加速作业数。
  - 支持对通过MaxCompute客户端、MaxCompute Studio客户端、JDBC连接、SDK连接提交的查询作业免费加速。单个MaxCompute项目支持的最大作业并发数为5，日免费加速作业数累计为500个。

    > 说明 如果作业数超过500个，执行SQL任务时会报错：
    >
    > ```
    > Error:com.aliyun.odps.OdpsException: ODPS-1800001: Session exception - Failed to submit sub-query in session because:Prepaid project run out of free query quota.
    > ```
    >
    > 此种情况您需修改启用MCQA的配置参数，详情请参见常见问题。

- 查询限制：
  - 如果您运行的查询作业的SQL扫描量大于查询加速免费限制上限，则作业会自动回退为普通查询作业。
  - 通过MaxCompute客户端、MaxCompute Studio客户端、JDBC连接、SDK连接提交的查询作业，如果作业并发数超出限制上限，则作业自动回退为普通查询作业。
  - 通过MaxCompute客户端、MaxCompute Studio客户端、JDBC连接、SDK连接提交的查询作业，如果日累计免费加速作业数超出限制上限，则作业自动回退为普通查询作业。

更高负载支持说明

如果您有更高负载的功能使用需求，MaxCompute后续会为使用包年包月资源的项目提供支持高负载的付费服务，后续支持情况请及时关注公告。

9.5. 使用说明

查询加速MCQA（MaxCompute Query Acceleration）是MaxCompute内建的功能，使用原生的MaxCompute SQL语言，并且支持所有的MaxCompute内建函数以及权限系统。本文为您介绍如何使用MCQA功能。

背景信息

MCQA功能支持的接入方式如下：

- MaxCompute客户端：无缝接入，使用说明请参见MaxCompute客户端。
- DataWorks临时查询或数据开发：默认无缝接入，使用说明请参见DataWorks临时查询或数据开发。
- JDBC：无缝接入，使用说明请参见JDBC。
- SDK：需要配置Pom依赖，使用说明请参见基于Java SDK启用MCQA功能。
- MaxCompute Studio：需要配置SQL执行模式，使用说明请参见基于MaxCompute Studio启用MCQA功能
- PyODPS：需要调用查询加速执行方法，使用说明请参见基于PyODPS启用MCQA功能。

- SQLAlchemy：基于PyODPS无缝接入，使用说明请参见基于PyODPS，使用SQLAlchemy或其他支持SQLAlchemy接口的第三方工具实现查询加速。

MaxCompute客户端

您可以通过MaxCompute客户端开启MCQA功能，操作步骤如下：

1. 下载最新版客户端（odpscmd）。

   > **说明** 客户端需要为0.35.1及以上版本。

2. 安装并配置客户端，详情请参见安装并配置MaxCompute客户端。
3. 修改客户端安装目录conf下的配置文件odps_config.ini，在配置文件最后一行增加如下命令行。

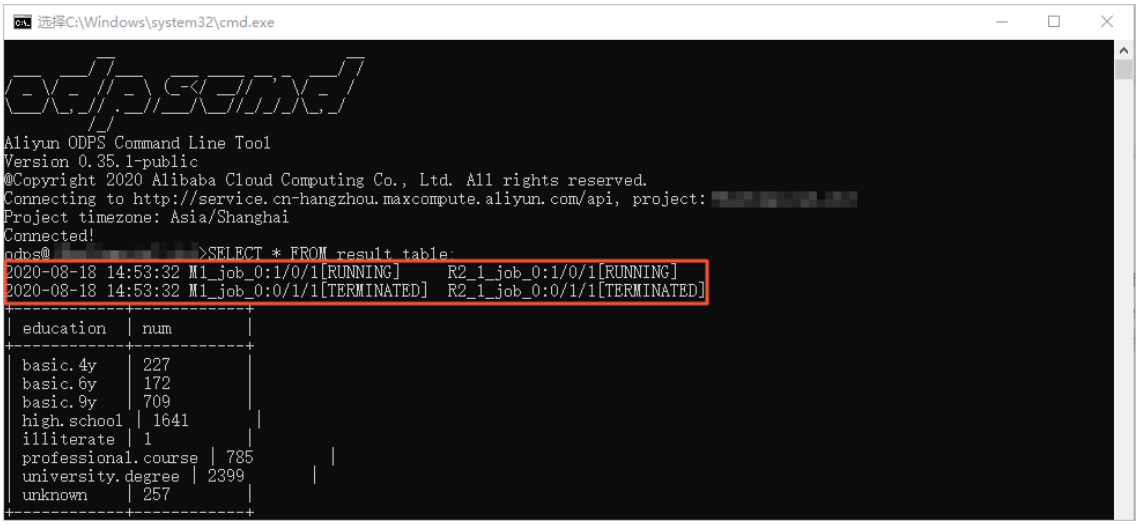
   ```
   enable_interactive_mode=true
   ```

4. 运行客户端安装目录bin下的MaxCompute客户端（Linux系统下运行./bin/odpscmd，Windows下运行./bin/odpscmd.bat）。出现如下信息，表示运行成功。

5. 执行查询作业验证MCQA功能已开启。

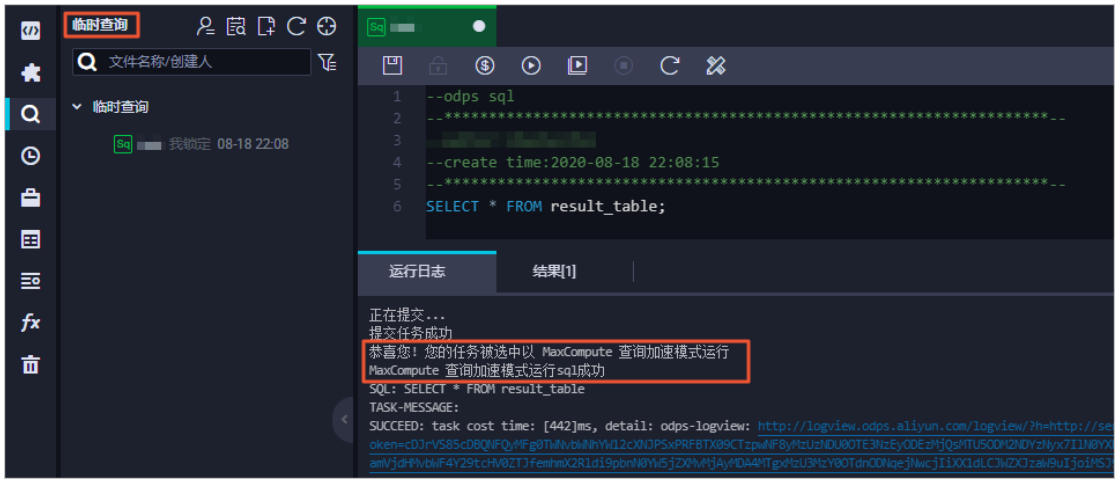
   执行查询作业后，客户端界面返回结果如果包含如下信息表明MCQA功能已开启。

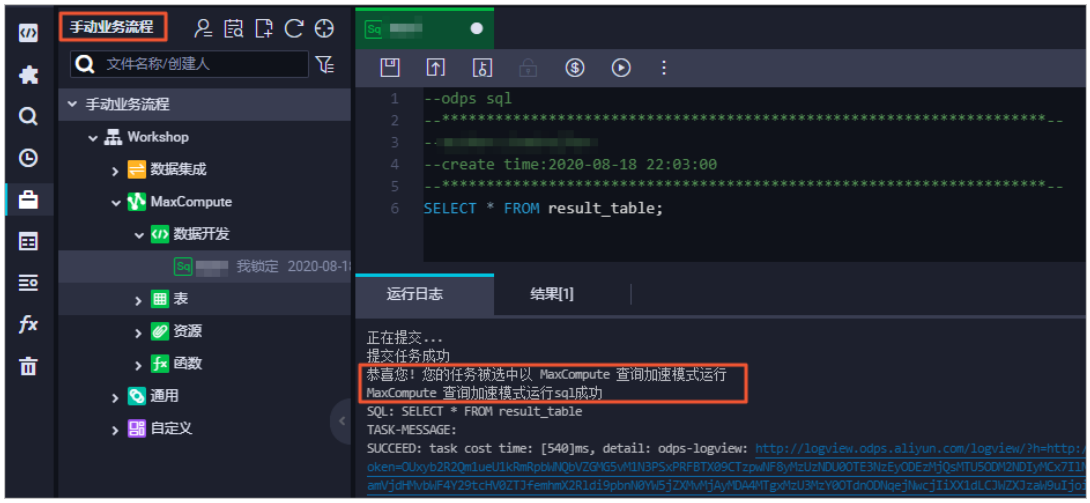
DataWorks临时查询或数据开发

DataWorks的**临时查询**及**手动业务流程**模块默认开启MCQA功能，您无需手动开启。如果您需要关闭MCQA功能，请提工单处理。

在**临时查询**模块执行查询作业，如果返回结果包含如下信息表明MCQA功能已开启。**临时查询**详情请参见创建临时查询。



在**手动业务流程**模块执行查询作业，返回结果包含如下信息表明MCQA功能已开启。**手动业务流程**详情请参见管理手动业务流程。



JDBC

JDBC使用场景如下：

- 使用JDBC连接MaxCompute，详情请参见JDBC使用说明。如果您需要在该场景下开启MCQA功能，需要修改相关配置，详情请参见配置JDBC启用MCQA功能。
- 使用JDBC连接Tableau，对MaxCompute中的数据进行可视化分析，详情请参见配置JDBC使用Tableau。如果您需要在该场景下开启MCQA功能，需要修改相关配置，详情请参见基于JDBC配置Tableau启用MCQA功能。
- 使用JDBC连接SQL Workbench/J，使用SQL Workbench/J对MaxCompute中的数据执行SQL语句，详情请参见配置JDBC使用SQL Workbench/J。如果您需要在该场景下开启MCQA功能，需要修改相关配置，详情请参见基于JDBC配置SQLWorkBench启用MCQA功能。

配置JDBC启用MCQA功能

使用JDBC连接MaxCompute时，您可以通过执行如下操作开启MCQA功能。使用JDBC连接MaxCompute的操作详情请参见JDBC使用说明。

1. 下载支持MCQA功能的JDBC JAR包或可编译的源代码。
2. 通过Maven方式配置Pom依赖。

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-jdbc</artifactId>
  <version>3.2.8</version>
  <classifier>jar-with-dependencies</classifier>
</dependency>
```

> **说明** 版本需要为3.2.0及以上版本。

3. 基于源代码创建Java程序，适配实际信息。详情请参见ODPS JDBC。

   需要适配的信息如下。

```
String accessId = "your_access_id";
String accessKey = "your_access_key";
String conn = "jdbc:odps:http://service.<regionid>.maxcompute.aliyun.com/api?project=<your_project_name>"&accessId&accessKey&
charset=UTF-8&interactiveMode=true&autoSelectLimit=1000000000";
Statement stmt = conn.createStatement();
--your_access_id、your_access_key为您的云账号信息，your_project_name为需要使用MCQA功能的项目名称。
Connection conn = DriverManager.getConnection(conn, accessId, accessKey);
Statement stmt = conn.createStatement();
String tableName = "testOdpsDriverTable";
stmt.execute("drop table if exists " + tableName);
stmt.execute("create table " + tableName + " (key int, value string)");
```

在JDBC中启用MCQA功能的方式与常规方式完全一致，只需要在连接串（`String conn`）或代码中（二选一即可）配置如下信息：

- 连接串（`String conn`）：增加`interactiveMode=true`属性，启用MCQA。数据量超过100万限制时，增加`autoSelectLimit=1000000000"`属性。
- 代码：增加`interactiveMode=true`语句。数据量超过100万限制时，增加`autoSelectLimit=1000000000"`语句。

4. （可选）您可以在连接串中配置如下参数，完善处理逻辑。
   - `enableOdpsLogger`：用于打印日志。未配置SLF4J时，建议您配置此参数为True。
   - `fallbackForUnknownError`：默认为False，设置为True时，表示发生未知错误时回退到离线模式。
   - `fallbackForResourceNotEnough`：默认为False，设置为True时，表示发生资源不足问题时回退到离线模式。
   - `fallbackForUpgrading`：默认为False，设置为True时，表示升级期间回退到离线模式。
   - `fallbackForRunningTimeout`：默认为False，设置为True时，表示执行超时时回退到离线模式。
   - `fallbackForUnsupportedFeature`：默认为False，设置为True时，表示遇到MCQA不支持的场景时回退到离线模式。
   - `alwaysFallback`：默认为False，设置为True时，表示在以上几种场景下全部回退到离线模式，仅在JDBC 3.2.3及以上版本才支持。

基于JDBC配置Tableau启用MCQA功能

服务器增加`interactiveMode=true`属性，用于开启MCQA功能。建议您同步增加`enableOdpsLogger=true`属性，用于打印日志。配置操作详情请参见配置JDBC使用Tableau。

完整的**服务器**配置示例如下。

```
http://service.cn-beijing.maxcompute.aliyun.com/api?project=****_beijing&interactiveMode=true&enableOdpsLogger=true&autoSelectLim
it=1000000000"
```

如果只对项目空间中的部分表进行Tableau操作，您可以在**服务器**参数中增加`table_list=table_name1, table_name2`属性选择需要的表，表之间用英文逗号（,）分隔。如果表过多，会导致Tableau打开缓慢，强烈建议使用此方式只载入需要的表。示例如下。

```
http://service.cn-beijing.maxcompute.aliyun.com/api?project=****_beijing&interactiveMode=true&enableOdpsLogger=true&autoSelectLim
it=1000000000"&table_list=orders,customers
```

对于有大量分区的表不建议把所有分区的数据都设置成数据源，可以筛选需要的分区，或通过自定义SQL获取需要的数据。

基于JDBC配置SQLWorkBench启用MCQA功能

完成JDBC驱动配置后，在Profile配置界面修改已填写的JDBC URL，支持SQLWorkBench使用MCQA功能。Profile配置操作详情请参见配置JDBC使用SQL Workbench/J。

需要配置的URL格式为`jdbc:odps:<MaxCompute_endpoint>?project=<MaxCompute_project_name>&accessId=<AccessKey ID>&accessKey=<AccessKey Secret>&charset=UTF-8&interactiveMode=true&autoSelectLimit=1000000000"`，参数说明如下：

- maxcompute_endpoint：MaxCompute服务所在区域的Endpoint。详情请参见Endpoint。
- maxcompute_project_name：MaxCompute项目空间名称。
- AccessKey ID：有访问指定项目空间权限的AccessKey ID。
- AccessKey Secret：AccessKey ID对应的AccessKey Secret。
- charset=UTF-8：字符集编码格式。
- interactiveMode：MCQA功能开关，`true`表示开启MCQA功能。
- autoSelectLimit：数据量超过100万限制时，需要配置此参数。

基于Java SDK启用MCQA功能

Java SDK详情请参见Java SDK介绍。您需要通过Maven配置Pom依赖，配置示例如下。

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-core</artifactId>
  <version>0.35.5-public</version>
</dependency>
```

> **说明** 版本需要为0.35.1及以上版本。

创建Java程序，命令示例如下。

```
import com.aliyun.odps.Odps;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.ResultSet;
import com.aliyun.odps.sqa.*;
import java.io.IOException;
import java.util.*;
public class SQLExecutorExample {
    public static void SimpleExample() {
        // 设置账号和项目信息。
        Account account = new AliyunAccount("<your_access_id>", "<your_access_key>");
        Odps odps = new Odps(account);
        odps.setDefaultProject("<your_project_name>");
        odps.setEndpoint("http://service.<regionid>.maxcompute.aliyun.com/api");
        // 准备构建SQLExecutor。
        SQLExecutorBuilder builder = SQLExecutorBuilder.builder();
        SQLExecutor sqlExecutor = null;
        try {
            // run in offline mode or run in interactive mode
            if (false) {
                // 创建一个默认执行离线SQL的Executor。
                sqlExecutor = builder.odps(odps).executeMode(ExecuteMode.OFFLINE).build();
            } else {
                // 创建一个默认执行查询加速SQL的Executor，并且在查询加速模式失败后，自动回退到离线查询。
                sqlExecutor = builder.odps(odps).executeMode(ExecuteMode.INTERACTIVE).fallbackPolicy(FallbackPolicy.alwaysFallbac
kPolicy()).build();
            }
            // 如果需要的话可以传入查询的特殊设置。
            Map<String, String> queryHint = new HashMap<>();
            queryHint.put("odps.sql.mapper.split.size", "128");
            // 提交一个查询作业，支持传入Hint。
            sqlExecutor.run("select count(1) from test_table;", queryHint);
            // 列举一些支持的常用获取信息的接口。
            // UUID
            System.out.println("ExecutorId:" + sqlExecutor.getId());
            // 当前查询作业的logview。
            System.out.println("Logview:" + sqlExecutor.getLogView());
            // 当前查询作业的Instance对象(Interactive模式多个查询作业可能为同一个Instance)。
            System.out.println("InstanceId:" + sqlExecutor.getInstance().getId());
            // 当前查询作业的阶段进度(Console的进度条)。
            System.out.println("QueryStageProgress:" + sqlExecutor.getProgress());
            // 当前查询作业的执行状态变化日志，例如回退信息。
            System.out.println("QueryExecutionLog:" + sqlExecutor.getExecutionLog());
            // 提供两种获取结果的接口。
            if(false) {
                // 直接获取全部查询作业结果，同步接口，可能会占用本线程直到查询成功或失败。
                // 一次性读取全部结果数据到内存中，当数据量较大时不建议使用，可能会有内存问题。
                List<Record> records = sqlExecutor.getResult();
                printRecords(records);
            } else {
                // 获取查询结果的迭代器ResultSet，同步接口，可能会占用本线程直到查询成功或失败。
                // 获取大量结果数据时推荐使用，分次读取查询结果。
                ResultSet resultSet = sqlExecutor.getResultSet();
                while (resultSet.hasNext()) {
                    printRecord(resultSet.next());
                }
            }
            // run another query
            sqlExecutor.run("select * from test_table;", new HashMap<>());
            if(false) {
                // 直接获取全部查询结果，同步接口，可能会占用本线程直到查询成功或失败。
                // 一次性读取全部结果数据到内存中，当数据量较大时不建议使用，可能会有内存问题。
                List<Record> records = sqlExecutor.getResult();
                printRecords(records);
            } else {
                // 获取查询结果的迭代器ResultSet，同步接口，可能会占用本线程直到查询成功或失败。
                // 获取大量结果数据时推荐使用，分次读取查询结果。
```

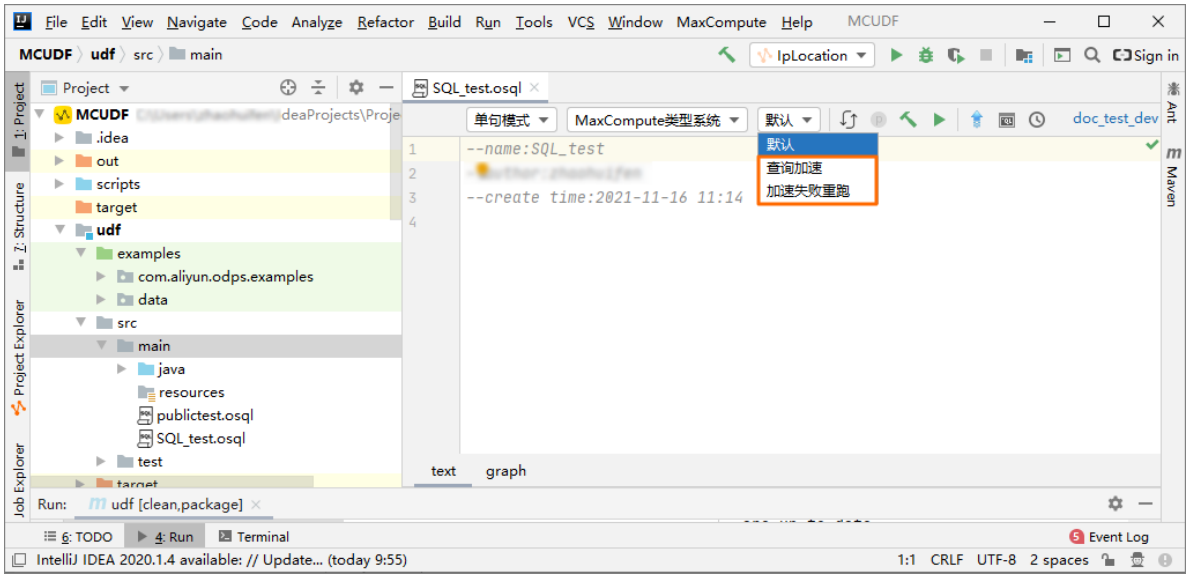
```
                ResultSet resultSet = sqlExecutor.getResultSet();
                while (resultSet.hasNext()) {
                    printRecord(resultSet.next());
                }
            }
        } catch (OdpsException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (sqlExecutor != null) {
                // 关闭Executor释放相关资源。
                sqlExecutor.close();
            }
        }
    }
    // SQLExecutor can be reused by pool mode
    public static void ExampleWithPool() {
        // 设置账号和项目信息。
        Account account = new AliyunAccount("your_access_id", "your_access_key");
        Odps odps = new Odps(account);
        odps.setDefaultProject("your_project_name");
        odps.setEndpoint("http://service.<regionid>.maxcompute.aliyun.com/api");
        // 通过连接池方式执行查询。
        SQLExecutorPool sqlExecutorPool = null;
        SQLExecutor sqlExecutor = null;
        try {
            // 准备连接池，设置连接池大小和默认执行模式。
            SQLExecutorPoolBuilder builder = SQLExecutorPoolBuilder.builder();
            builder.odps(odps)
                    .initPoolSize(1) // init pool executor number
                    .maxPoolSize(5)  // max executors in pool
                    .executeMode(ExecuteMode.INTERACTIVE); // run in interactive mode
            sqlExecutorPool = builder.build();
            // 从连接池中获取一个Executor，如果不够将会在Max限制内新增Executor。
            sqlExecutor = sqlExecutorPool.getExecutor();
            // Executor具体用法和上一示例一致。
            sqlExecutor.run("select count(1) from test_table;", new HashMap<>());
            System.out.println("InstanceId:" + sqlExecutor.getId());
            System.out.println("Logview:" + sqlExecutor.getLogView());
            List<Record> records = sqlExecutor.getResult();
            printRecords(records);
        } catch (OdpsException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            sqlExecutor.close();
        }
        sqlExecutorPool.close();
    }
    private static void printRecord(Record record) {
        for (int k = 0; k < record.getColumnCount(); k++) {
            if (k != 0) {
                System.out.print("\t");
            }
            if (record.getColumns()[k].getType().equals(OdpsType.STRING)) {
                System.out.print(record.getString(k));
            } else if (record.getColumns()[k].getType().equals(OdpsType.BIGINT)) {
                System.out.print(record.getBigint(k));
            } else {
                System.out.print(record.get(k));
            }
        }
    }
    private static void printRecords(List<Record> records) {
        for (Record record : records) {
            printRecord(record);
            System.out.println();
        }
    }
    public static void main(String args[]) {
        SimpleExample();
        ExampleWithPool();
    }
```

```
}
```

基于MaxCompute Studio启用MCQA功能

V3.5.0或以上版本的MaxCompute Studio插件支持MCQA功能，推荐您安装最新版本的MaxCompute Studio插件。更多安装MaxCompute Studio插件的操作，请参见安装MaxCompute Studio。

您在MaxCompute Studio的SQL编辑器中，选择带查询加速功能的SQL执行模式**查询加速**或**加速失败重跑**后，执行查询语句即可启用查询加速功能。如下图所示。



- **查询加速**：使用查询加速功能执行SQL查询语句。
- **加速失败重跑**：使用查询加速功能执行SQL查询语句失败后，回退到离线模式（即默认模式）执行SQL查询语句。

基于PyODPS启用MCQA功能

0.10.7或以上版本的PyODPS支持MCQA功能。您可以在运行SQL时使用`run_sql_interactive()`方法启用MCQA功能。命令示例如下。

```
odps = ODPS(...)
instance = odps.run_sql_interactive('select count(*) from test_table')
# logview
print(instance.get_logview_address())
# 可选：如果使用Tunnel下载结果，则不需要客户端等待。
# 等待运行完毕，也可以使用wait_for_completion，区别在于wait_for_success将在任务失败时抛出异常。
instance.wait_for_success()
# 下载结果
# 显示ODPS返回的警告
print(instance.get_warnings())
# 用于屏显，若任务失败则返回错误信息
print(instance.get_printable_result())
# 从frontend下载结果
for each_record in instance.open_reader():
    pass # each_record 是 odps.Record 类型
# 使用tunnel下载结果
for each_record in instance.open_reader(tunnel=True):
    pass
```

`run_sql_interactive()`方法可携带的参数如下：

- `service_name`：可选。用于指定查询加速会话的名称。
- `service_startup_timeout`：可选。用于指定等待Session Attach的超时时间。
- `force_reattach`：可选。`run_sql_interactive()`默认会自动复用Session。用于强制重新Attach Session。

基于PyODPS，使用SQLAlchemy或其他支持SQLAlchemy接口的第三方工具实现查询加速

PyODPS集成了SQLAlchemy，可以使用SQLAlchemy查询MaxCompute数据。您需要在连接串中指定如下参数实现查询加速：

- `interactive_mode=true`：必填。查询加速功能总开关。
- `reuse_odps=true`：可选。打开强制复用连接，对于部分第三方工具（例如Apache SuperSet），打开此选项可提高性能。

您可以在连接串中配置`fallback_policy=<policy1>,<policy2>,...`参数，完善处理逻辑。与JDBC的配置项类似，控制加速失败的回退行为。

- `generic`：默认为False，设置为True时，表示发生未知错误时回退到离线模式。

- `noresource`：默认为False，设置为True时，表示发生资源不足问题时回退到离线模式。
- `upgrading`：默认为False，设置为True时，表示升级期间回退到离线模式。
- `timeout`：默认为False，设置为True时，表示执行超时时回退到离线模式。
- `unsupported`：默认为False，设置为True时，表示遇到MCQA不支持的场景时回退到离线模式。
- `default`：等同于同时指定unsupported、upgrading、noresource和timeout。如果连接串中未指定`fallback_policy`，则此项为默认值。
- `all`：默认为False，设置为True时，表示在以上几种场景下全部回退到离线模式，仅在JDBC 3.2.3及以上版本才支持。

例如，打开查询加速，打开强制复用链接，在查询加速功能尚未支持、升级中和资源不足时回退到离线模式的连接串如下。

```
odps://<access_id>:<access_key>@<project>/?endpoint=<endpoint>&interactive_mode=true&reuse_odps=true&fallback_policy=unsupported,
upgrading,noresource
```

常见问题

- 问题现象一：使用JDBC链接MaxCompute，执行包年包月资源的SQL任务时报错（ODPS-1800001），详细报错信息如下。

  ```
  Error:com.aliyun.odps.OdpsException: ODPS-1800001: Session exception - Failed to submit sub-query in session because:Prepaid p
  roject run out of free query quota.
  ```

  - 可能原因：

    您当前使用了查询加速（MCQA）功能，当前查询加速功能正在公测中，如果您已购买包年包月套餐，公测期间，无需进行额外操作，即可免费体验查询加速功能。免费体验查询加速时，单个MaxCompute项目支持的最大作业并发数为5，日免费加速作业数累计为500个。

    如果作业数超过500个时，会出现上述报错。

    > **说明** 更多关于免费体验查询加速功能的介绍请参见免费体验。

  - 解决方案：

    您需要在配置JDBC启用MCQA功能的配置过程中，设置`alwaysFallback`参数值为true，设置完成后，没有超过500个作业数时，能正常使用MCQA进行查询加速，超过500个的作业会退回至离线模式。

    配置的详细操作及参数解释请参见配置JDBC启用MCQA功能。

- 问题现象二：基于PyODPS发送请求并获取结果的时长比DataWorks长
  - 可能原因：
    - 使用了`wait_for_xxx`方法，延长了时间。
    - 轮询间隔时间长。
  - 解决方案：
    - 请求本身运行很快的情况下，不使用`wait_for_xxx`，发出请求后直接使用Tunnel下载结果。
    - 调低轮询间隔：`instance.wait_for_success(interval=0.1)`。命令示例如下。

      ```
      from odps import ODPS, errors
      max_retry_times = 3
      def run_sql(odps, stmt):
          retry = 0
          while retry < max_retry_times:
              try:
                  inst = odps.run_sql_interactive(stmt)
                  print(inst.get_logview_address())
                  inst.wait_for_success(interval=0.1)
                  records = []
                  for each_record in inst.open_reader(tunnel=True):
                      records.append(each_record)
                  return records
              except errors.ODPSError as e:
                  retry = retry + 1
                  print("Error: " + str(e) + " retry: " + str(retry) + "/" + str(max_retry_times))
                  if retry >= max_retry_times:
                      raise e
      odps = ODPS(...)
      run_sql(odps, 'select 1')
      ```

10.外部表

10.1. 概述

MaxCompute作为阿里云大数据平台的核心计算组件，拥有强大的计算能力。MaxCompute能够调度大量的节点执行并行计算，同时对分布式计算中的Failover、重试等功能都有体系化的处理、管理机制。

背景信息

MaxCompute SQL作为分布式数据处理的主要入口，可实现EB级别离线数据的快速处理和存储。随着大数据业务的不断扩展，新的数据使用场景在不断产生，MaxCompute计算框架也在不断的演化。MaxCompute原来主要面对内部特殊格式数据的强大计算能力，正一步步的开放给不同的外部数据。

现阶段MaxCompute SQL处理的主要是以**cfile**列格式存储在内部MaxCompute表格中的结构化数据。对于MaxCompute表外的各种用户数据（包括文本以及各种非结构化的数据），您需要通过不同工具导入MaxCompute表再进行计算。以OSS为例，如果您需要在MaxCompute中处理OSS上的数据，通常有以下两种方式：

- 通过OSS SDK或者其他工具从OSS下载数据，然后再通过MaxCompute Tunnel将数据导入表里。
- 编写UDF，在UDF里直接调用OSS SDK访问OSS数据。

但这两种方法都有不足之处：

- 第一种方法需要在MaxCompute系统外部做一次中转，如果OSS数据量太大，还需要考虑如何并发来加速，无法充分利用MaxCompute大规模计算的能力。
- 第二种方法通常需要申请UDF网络访问权限，还需要开发者自己控制作业并发数，解决数据如何分片的问题。

因此，MaxCompute提供了外部表来解决上述问题。外部表的功能，旨在提供除MaxCompute内部表格以外的其他数据的处理能力。通过一条简单的DDL语句，即可在MaxCompute上创建一张外部表，建立MaxCompute表与外部数据源的关联，提供各种数据的接入和输出能力。创建好的外部表可以像普通的MaxCompute表一样使用（大部分场景），充分利用MaxCompute SQL的强大计算功能。

> 说明
> - 使用外部表功能时，外部表的数据不会复制一份存在MaxCompute上并产生存储费用。
> - 外部表支持全量搜索。
> - Tunnel功能及Tunnel SDK当前不支持外部表操作。您可以通过Tunnel直接上传数据到MaxCompute内部表，或者是通过OSS Python SDK上传到OSS后，在MaxCompute使用外部表做映射。
> - 您可以通过DataWorks配合MaxCompute对外部表进行可视化的创建、搜索、查询、配置、加工和分析。详情请参见外部表。

外部表示例

您可以通过以下示例，深入了解通过MaxCompute外部表功能处理各种非结构化数据的方法：

- 访问OSS和TableStore（OTS）非结构化数据，请参见 访问OSS非结构化数据和访问OTS非结构化数据。
- 外部表访问OSS的账号，在RAM中自定义授权MaxCompute访问OSS的权限，请参见STS模式授权。
- MaxCompute的非结构化框架支持通过INSERT方式将MaxCompute的数据直接输出到OSS，请参见将数据写入OSS。
- 处理各种开源格式数据，请参见支持开源格式数据。

10.2. STS模式授权

本文为您介绍如何通过一键式授权或在RAM中自定义角色的方式授权MaxCompute访问OSS和OTS（Tablestore）的权限。MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）实现对数据的安全访问。

OSS的STS模式授权

MaxCompute外部表直接访问OSS的数据，需要事先将OSS的数据相关权限赋予MaxCompute的访问账号，您可以通过以下方式授权：

- 方式一（推荐）：当MaxCompute和OSS的Owner是同一个账号时，登录阿里云账号后，单击此处完成一键授权。
- 方式二：当MaxCompute和OSS的Owner不是同一个账号时，您可以自定义角色并授权。
  i. 创建RAM角色。

使用OSS账号登录RAM控制台，通过RAM控制台中的RAM角色创建角色。例如角色oss-admin。



> **说明** 更多创建角色操作信息，请参见创建可信实体为阿里云账号的RAM角色。

ii. 修改RAM角色策略内容。

在RAM控制台中的RAM角色管理页面，单击新建的RAM角色名称后，在**信任策略管理**页签修改信任策略。当MaxCompute和OSS的Owner不是同一个账号，设置如下所示，支持跨账号实现MaxCompute访问OSS。

```
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号ID@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

> **说明**
>
> `MaxCompute的Owner云账号ID` 为访问OSS服务的账号。
>
> 更多修改角色策略内容操作，请参见修改RAM角色的信任策略。

iii. 新建权限策略。

在RAM控制台中的权限策略管理页面，新建权限策略，例如AliyunODPSRolePolicy。策略内容如下，您也可以自定义其他权限。

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "oss:ListBuckets",
   "oss:GetObject",
   "oss:ListObjects",
   "oss:PutObject",
   "oss:DeleteObject",
   "oss:AbortMultipartUpload",
   "oss:ListParts"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
```

iv. 将新建的权限策略AliyunODPSRolePolicy授权给新建的RAM角色。

更多授权操作信息，请参见为RAM角色授权。

OTS的STS模式授权

MaxCompute外部表直接访问OTS的数据，需要事先将OTS的数据相关权限赋予MaxCompute的访问账号，您可以通过以下方式授权：

- 方式一（推荐）：当MaxCompute和OTS的Owner是同一个账号时，登录阿里云账号后，单击此处完成一键授权。
- 方式二：当MaxCompute和OTS的Owner不是同一个账号时，您可以自定义角色并授权。
  i. 创建RAM角色。

  使用OTS账号登录RAM控制台，通过RAM控制台中的RAM角色创建角色。例如角色oss-adminots。

  更多创建角色操作信息，请参见创建可信实体为阿里云账号的RAM角色。

  ii. 修改RAM角色策略内容。

  在RAM控制台中的RAM角色管理页面，单击新建的RAM角色名称后，在**信任策略管理**页签修改信任策略。当MaxCompute和OTS的Owner不是同一个账号时，设置如下所示，支持跨账号实现MaxCompute访问OTS。

```
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号的UID@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

  `MaxCompute的Owner云账号ID` 为访问OTS服务的账号。

  iii. 新建权限策略。

  在RAM控制台中的权限策略管理页面，新建权限策略，例如AliyunODPSRolePolicy。策略内容如下，您也可以自定义其他权限。

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "ots:ListTable",
   "ots:DescribeTable",
   "ots:GetRow",
   "ots:PutRow",
   "ots:UpdateRow",
   "ots:DeleteRow",
   "ots:GetRange",
   "ots:BatchGetRow",
   "ots:BatchWriteRow",
   "ots:ComputeSplitPointsBySize"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
```

  iv. 将新建的权限策略AliyunODPSRolePolicy授权给新建的RAM角色。

  更多授权操作信息，请参见为RAM角色授权。

10.3. 网络开通流程

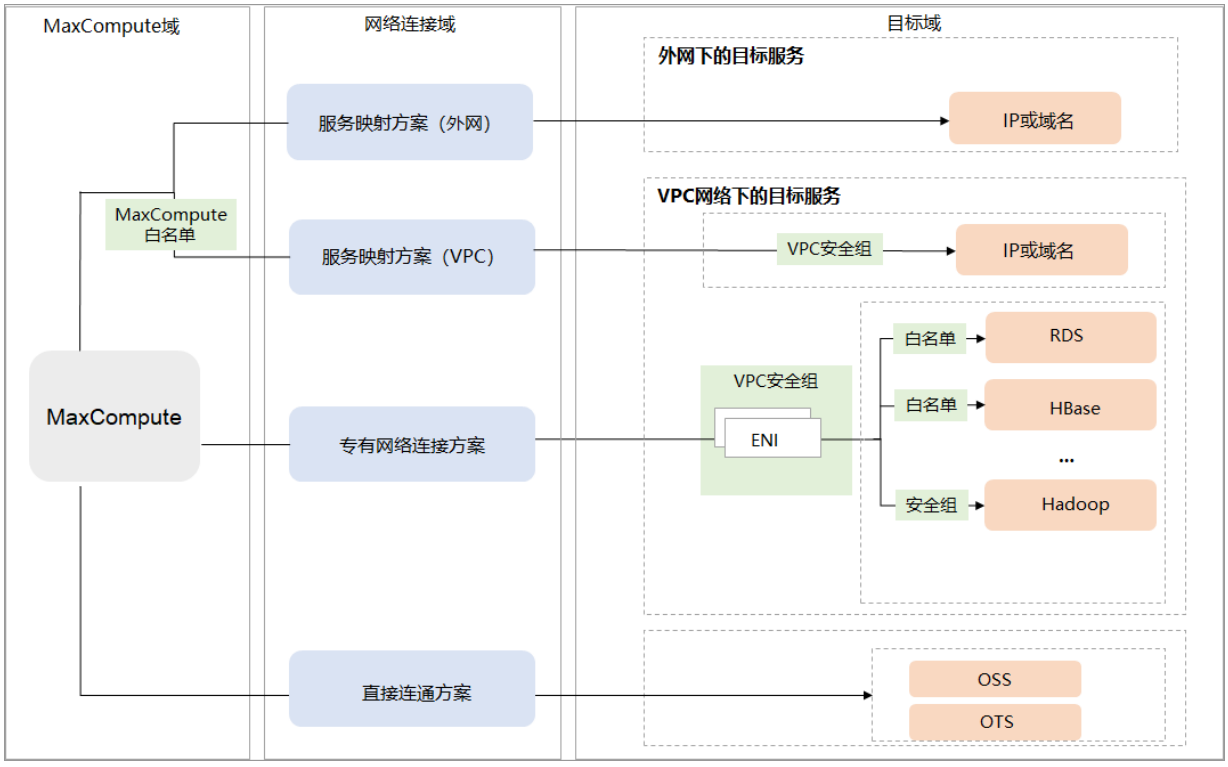
当您通过MaxCompute访问位于外网或VPC网络中的目标服务时，默认情况下，MaxCompute与目标服务之间的网络不可达。您需要开通网络连接以便访问外网或VPC中的目标服务（例如指定IP或域名、RDS、HBase集群、Hadoop集群等）。本文为您介绍MaxCompute与目标服务间的网络结构及支持的网络开通方案。

背景信息

当您涉及在如下场景中访问位于外网或VPC网络中的目标服务时，需要申请开通MaxCompute和目标服务间的网络连接。

- UDF：用户创建了自定义函数UDF，且需要调用UDF访问位于外网或VPC网络中的目标服务。
- 外部表：用户创建了外部表，且需要基于外部表访问位于外网或VPC网络中的目标服务。
- 湖仓一体：用户基于MaxCompute构建了湖仓一体能力，且需要访问位于VPC网络中的目标服务。

MaxCompute与目标服务间的网络结构及支持的网络开通方案如下。



MaxCompute访问目标服务有如下三种方案：

- 服务映射方案

  根据目标服务所处网络类型分为如下两种：

  - 服务映射方案（外网）

    适用于通过UDF或外部表访问处于外网中的目标IP或域名的场景。您需要提工单申请，如果目标IP或域名不存在安全限制，审核通过后即可访问目标IP或域名。

    > **说明** 当待访问的外网IP或域名存在安全限制时，请联系外网对应责任人根据相应的安全限制进行处理，确保MaxCompute访问外网IP或域名不受限制。

  - 服务映射方案（VPC）

    适用于MaxCompute与VPC间的网络已连接，需要通过UDF或外部表访问处于VPC网络中的单个IP或域名的场景。您只需要将MaxCompute项目所属地域的IP网段添加至VPC的安全组，并将目标IP或域名所属VPC实例添加至MaxCompute项目，完成双向授权后，即可访问目标IP或域名。

- 专有网络连接方案

  适用于通过外部表、UDF或基于湖仓一体架构访问处于VPC网络下的RDS、HBase集群、Hadoop集群场景。您需要通过VPC网络管理控制台进行授权以及配置安全组，在MaxCompute控制台创建MaxCompute与VPC网络之间的连接，配置RDS、HBase集群、Hadoop集群等目标服务安全组，以此来建立MaxCompute与目标服务间的网络通路。

- 直接连通方案

  适用于通过UDF或外部表访问阿里云OSS、OTS（Tablestore）服务的场景。OSS、OTS服务与MaxCompute连通无需申请开通专有网络。

  > 说明
  > - 如果您创建了OSS或OTS外部表，可以通过OSS、OTS的内网域名访问。
  > - 如果您调用UDF访问OSS、OTS，只能通过OSS、OTS的外网域名访问。您在访问之前，需要在MaxCompute添加外网访问OSS、OTS的IP白名单。详情请参见添加IP白名单。更多OSS域名信息，请参见访问域名和数据中心。更多OTS域名信息，请参见服务地址。

前提条件

当您使用服务映射方案（外网）或专有网络连接方案申请开通MaxCompute与目标服务间的网络连接前，请确认已满足如下条件：

- 已创建MaxCompute项目。

  如果已有MaxCompute项目，直接使用即可，不必再创建新项目。在湖仓一体场景中，建议设置MaxCompute项目的数据类型为Hive兼容类型。更多创建MaxCompute项目信息，请参见创建MaxCompute项目。

- 如果需要访问位于VPC网络中的目标服务，请提前获取VPC所有者账号。同时还需要获取访问MaxComoute项目的阿里云账号，以及目标服务环境或

集群的管理员账号。

使用限制

- 在湖仓一体场景中，MaxCompute项目必须由VPC实例所有者账号创建。
- 通过MaxCompute访问位于VPC网络中的RDS、HBase集群、Hadoop集群时，MaxCompute项目必须与RDS、HBase集群、Hadoop集群所使用的VPC网络处于相同地域。

开通地域

MaxCompute仅支持在如下地域通过服务映射方案或专有网络连接方案开通网络连接。

方案类型	支持地域	支持连通的目标
服务映射方案（外网服务映射方案）	中国站所有地域	外网IP或域名
服务映射方案（VPC服务映射方案）	• 华北2（北京） • 华东2（上海）	VPC IP或域名
专有网络连接方案	• 华北2（北京） • 华东2（上海） • 华北3（张家口） • 华东1（杭州） • 华南1（深圳）	• VPC IP或域名 • RDS • HBase集群 • Hadoop集群

服务映射方案（外网）

MaxCompute访问外网IP或域名需要按照如下步骤进行操作：

1. **提交申请**

   提工单，申请配置外网的IP或域名、端口的白名单。

   工单中需要填写目标IP或域名、端口，如果需要访问多个域名或端口，请使用英文逗号（,）分隔。例如访问阿里云域名，需要提供的网络配置信息为 `www.aliyun.com:80` ；访问高德服务，需要提供的网络配置信息为 `restapi.amap.com:443,restapi.amap.com:80` 。

   MaxCompute技术支持团队接收到提交的申请信息后，会进行审核并完成配置。确认工单处理完毕后，即可继续执行后续步骤。

2. **测试网络连通性**

   登录MaxCompute客户端，配置 `odps.internet.access.list` 属性并执行调用UDF的SQL语句，访问目标IP或域名。命令格式如下。

   ```
   --设置网络连接申请表单中配置的外网IP或域名、端口。即下方SQL语句要访问的外网IP或域名。
   set odps.internet.access.list=<ip_address:port | realm_name:port>;
   --执行SQL语句调用UDF。
   select <UDF_name>("<http://ip_address | realm_name>");
   ```

   - ip_address:port | realm_name:port：必填。目标外网IP或域名、端口。
   - UDF_name：访问外网IP或域名的UDF。

     假设UDF代码示例如下。

     ```
     package com.aliyun.odps.test.udf;
     import com.aliyun.odps.udf.UDF;
     import java.io.BufferedReader;
     import java.io.IOException;
     import java.io.InputStreamReader;
     import java.net.URL;
     public class <UDF_name> extends UDF {
       public String evaluate(String urlStr) throws IOException {
         URL url = new URL(urlStr);
         StringBuilder sb = new StringBuilder();
         try (BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream()))) {
           String line;
           while ((line = reader.readLine()) != null) {
             sb.append(line).append('\n');
           }
         }
         return sb.toString();
       }
     }
     ```

     基于UDF示例代码创建的内建函数名称为url_fetch，网络连接审核通过后，执行命令示例如下。

     ```
     set odps.internet.access.list=www.aliyun.com:80;
     select url_fetch("http://www.aliyun.com");
     ```

服务映射方案（VPC）

MaxCompute访问VPC网络中的服务需要按照如下步骤进行操作：

1. **配置白名单**

   i. 为目标VPC实例安全组配置MaxCompute IP网段。

   您需要在VPC实例的安全组中添加待开通网络连接地域对应的MaxCompute IP网段，允许MaxCompute网段的IP访问VPC网络中的服务，IP网段信息如下表所示。更多配置VPC安全组操作，请参见VPC安全组。

地域	MaxCompute IP网段
华东2（上海）	100.104.49.64/26、100.104.212.192/26、100.104.244.0/26、100.104.94.0/26
华北2（北京）	100.104.218.0/26、100.104.120.0/26、100.104.156.192/26、100.104.149.0/26、100.104.49.64/26、100.104.212.192/26、100.104.244.0/26、100.104.94.0/26

   ii. 登录MaxCompute客户端，将目标VPC实例添加至MaxCompute白名单，允许VPC网络中的服务访问MaxCompute。命令格式如下。

   ```
   --将目标VPC实例添加至MaxCompute白名单。
   setproject odps.security.outbound.destination=<RegionID>_<VPC实例ID>[*];
   ```

   - RegionID：必填。VPC网络所属的地域ID。获取方法，请参见获取RegionID及VPC ID。
   - VPC实例ID：必填。VPC实例ID。您可以登录专有网络管理控制台，在**实例ID/名称**列获取目标VPC实例ID。
   - [*]：必填。通配符，表示将VPC网络下的所有IP和端口都加入MaxCompute白名单。`[]`不可省略。

   例如，需要连接的VPC实例ID为`vpc-bp1e4p7feyvwt103j****`，所在地域为`华东2（上海）`，将VPC内的所有IP和端口都加入MaxCompute白名单，命令示例如下。

   ```
   setproject odps.security.outbound.destination=cn-shanghai_vpc-bp1e4p7feyvwt103j****[*];
   ```

2. **测试网络连通性**

   通过MaxCompute客户端设置如下属性并提交调用UDF的SQL语句，访问VPC网络中的目标IP。

   例如需要访问的VPC内的网络IP和端口为`192.0.2.0:80`，命令示例如下。

   ```
   --指定要访问的VPC ID。
   set odps.vpc.id=vpc-bp1e4p7feyvwt103j****;
   --指定要访问的VPC网络IP和端口号。
   set odps.vpc.access.ips=192.0.2.0:80;
   --调用UDF访问目标IP和端口号。
   select url_fetch("http://192.0.2.0:80");
   ```

专有网络连接方案

通过专有网络连接方案开通网络连接的操作步骤如下：

1. **授权**

   使用VPC实例所有者账号，在登录阿里云状态下单击授权，单击**同意授权**完成授权。

   授权权限包括允许MaxCompute在VPC安全组内创建并绑定弹性网卡ENI。授权后，MaxCompute会自动在VPC网络内创建弹性网卡ENI。

2. **配置安全组**

   在VPC实例下，创建独立安全组用于控制MaxCompute对VPC网络内的各种资源的访问权限。

   i. 登录专有网络管理控制台，单击目标VPC实例ID后，单击**资源管理**页签。



ii. 在**资源管理**页签的**专有网络资源**区域，将鼠标悬停至安全组数值上方后，单击**添加**，为MaxCompute创建独立安全组并记录安全组ID。

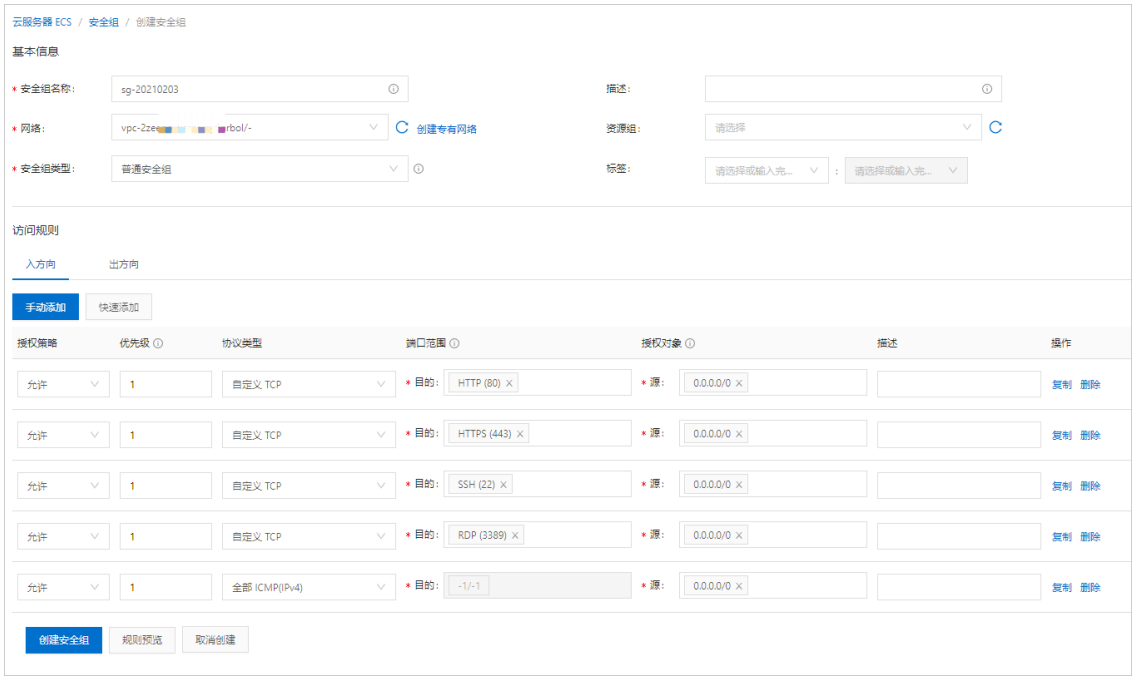
新建安全组为普通安全组，需要选择与连通服务相同的VPC网络。更多创建安全组操作，请参见创建安全组。

- **添加安全组**



- **配置安全组**

您可以按照默认配置添加安全组。



> **说明**
>
> - 安全组为普通安全组，不是企业安全组。普通安全组的出口是默认开启的。企业安全组的出口是默认关闭的，会导致访问不了VPC内的任何服务。
> - MaxCompute在后续的创建网络通路过程中会默认自动根据带宽需求创建2个ENI，不收取费用，由MaxCompute创建的ENI将位于这个安全组内。在HBase场景，如果用户的HBase无法对某个安全组开放网络权限，您可以将MaxCompute创建的ENI IP添加至白名单即可。由于ENI IP可能会发生变化，推荐您添加VPC实例对应交换机的IP网段至白名单。
>
> 您可以在ECS管理控制台的左侧导航栏，单击**弹性网卡**，获取ENI IP。

3. **创建网络连接**

阿里云账号（主账号）或拥有tenant的Super_Administrator角色的RAM用户（子账号）可以在MaxCompute控制台创建MaxCompute与VPC网络之间的连接。具体操作如下所示：

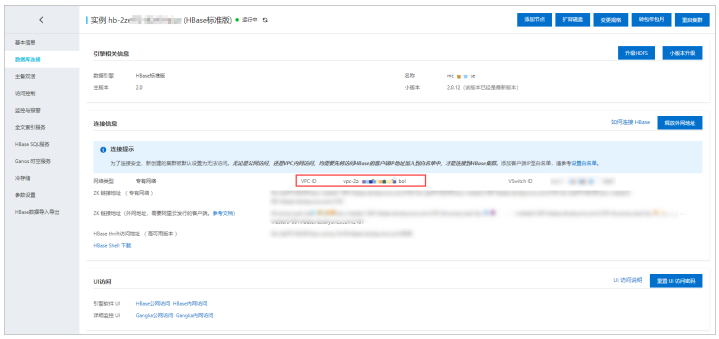
> **说明** tenant的Super_Administrator角色可以在MaxCompute控制台的**用户管理**页签授权。仅主账号或已经拥有tenant的Super_Administrator角色的子账号可以操作授权。详情请参见添加用户（账号级别）。

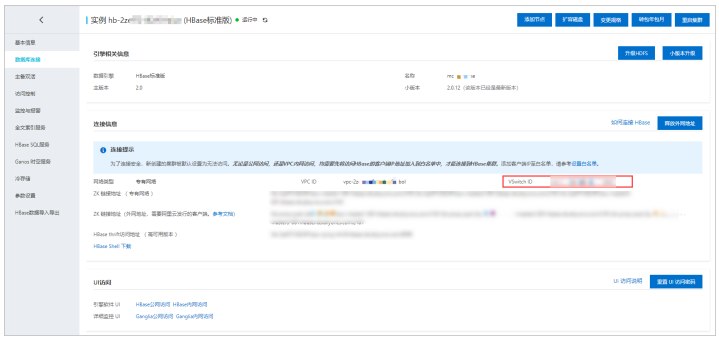
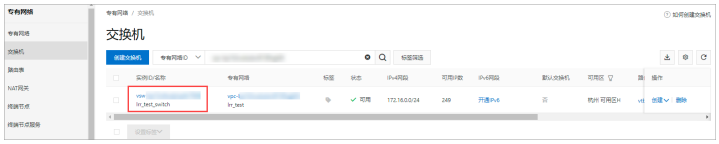
i. 登录MaxCompute控制台。

ii. 在MaxCompute控制台的**网络资源**页签，单击**新增网络连接**。

iii. 在**新增网络链接**对话框，按下表填写参数后单击**确定**。

参数名称	说明

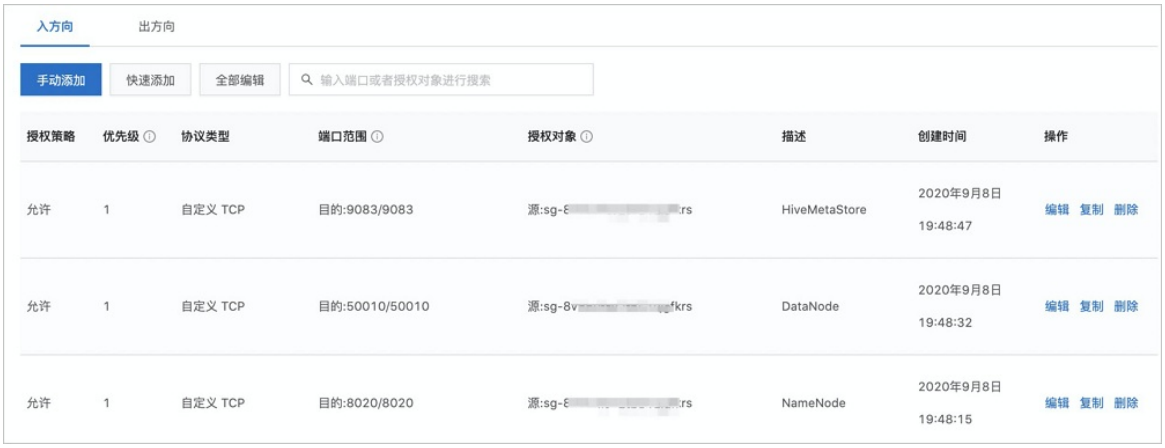
参数名称	说明
链接名称	自定义网络链接名称。格式如下： ■ 字母开头。 ■ 只能包含字母、下划线（_）和数字。 ■ 长度在1-63个字符。
类型	网络连接类型。默认为**直通链接（passthrough）**。 **说明** 直通链接对应的即是专有网络连接方案。
区域	MaxCompute支持的通过专有网络连接方案开通网络连接的地域。详情请参见开通地域。
vpcid	VPC实例ID。 获取方式如下： ■ 连接HBase、Hadoop集群时：您可以在对应控制台的网络连接信息处获取该信息。 例如，当您连接阿里云HBase集群时，您可以登录HBase管理控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可在右侧**连接信息**处获取VPC ID。  ■ 其他情况：登录专有网络管理控制台，在**专有网络**页面即可获取VPC ID。 

参数名称	说明
vSwitchID	VPC网络绑定的交换机ID。 获取方式如下： ■ 连接HBase、Hadoop集群时：您可以在对应控制台的网络连接信息处获取该信息。 例如，当您连接阿里云HBase集群时，您可以登录HBase管理控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可在右侧**连接信息**处获取交换机ID。  ■ 其他情况：登录专有网络管理控制台，在左侧导航栏单击**交换机**，即可在右侧通过VPC实例ID获取到交换机ID。 
securityGroup	步骤2中记录的安全组ID。

4. 配置目标服务的安全组。
   - 配置Hadoop集群安全组。

     为Hadoop集群的安全组配置如下信息，确保MaxCompute可以访问Hadoop集群。安全组配置内容如下：
     - 配置Hadoop集群所在安全组的入方向访问规则。
     - 授权对象为ENI所在的安全组，即步骤2中创建的安全组。
     - HiveMetaStore端口：9083。
     - HDFS NameNode端口：8020。
     - HDFS DataNode端口：50010。

     例如，当您连接在阿里云E-MapReduce上创建的Hadoop集群时，您需要配置的安全组规则如下图所示。详细配置操作，请参见添加安全组规则。

   - 配置HBase集群安全组。

     将为MaxCompute创建的安全组或ENI IP加入HBase集群的安全组或IP白名单中。

     例如，当您连接阿里云HBase集群时，可以登录HBase管理控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**访问控制**，即可在**安全组**或**白名单设置**页签添加安全组或IP白名单。更多添加安全组或IP白名单操作，请参见设置白名单和安全组。



说明 如果不允许添加安全组，您可以在**白名单设置**页签添加MaxCompute创建的ENI IP。当MaxCompute配置变更时，ENI IP可能会发生变化，推荐您添加交换机ID的IP网段添加至白名单。

- 配置RDS安全组。

  将为MaxCompute创建的安全组或ENI IP加入RDS安全组或IP白名单中。

  例如，当您连接阿里云RDS时，可以登录RDS管理控制台，在**实例列表**页面单击目标实例名称后，在左侧导航栏单击**数据安全性**，即可在**安全组**或**白名单设置**页签添加安全组或IP白名单。更多添加安全组或IP白名单操作，请参见设置安全组或设置IP白名单。



说明 当MaxCompute配置变更时，ENI IP可能会发生变化，推荐您添加交换机ID的IP网段添加至白名单。

5. **测试网络连通性**

   参照湖仓一体操作，测试网络连通性。详细操作请参见MaxCompute湖仓一体。

10.4. OSS外部表

10.4.1. 创建OSS外部表

MaxCompute支持您在项目中创建OSS（Object Storage Service）外部表，与存储服务OSS上的目录建立映射关系，您可以通过OSS外部表访问OSS目录下的数据文件中的非结构化数据，或将MaxCompute项目中的数据写入OSS目录。本文为您介绍创建OSS外部表的语法、参数信息并提供示例。

背景信息

对象存储服务OSS是一种海量、安全、低成本、高可靠的云存储服务，适合存放任意类型的数据文件。当您需要使用MaxCompute读取存储在OSS目录中的数据或需要将MaxCompute项目中的数据写入OSS目录时，可以在MaxCompute项目中创建OSS外部表建立与目录的映射关系。

OSS外部表包含分区表和非分区表两种类型，实际需要创建哪种类型的表，主要取决于OSS中数据文件的存储路径格式。当数据文件以分区路径方式存储时，需要创建分区表；否则创建非分区表。更多读取分区数据信息，请参见读取以分区方式存储的OSS数据。

前提条件

在创建OSS外部表前，请确认执行操作的账号已满足如下条件：

- 已授予访问OSS的权限。

更多授权信息，请参见OSS的STS模式授权。

- 已具备在MaxCompute项目中创建表（CreateTable）的权限。

  更多表操作权限信息，请参见权限列表。

注意事项

在使用OSS外部表时，您需要注意：

- 对于不同格式的数据文件，创建OSS外部表语句仅个别参数设置有出入，请您仔细阅读创建OSS外部表语法及参数说明，确保创建符合实际业务需求的外部表，否则读取OSS数据或将数据写入OSS操作会执行失败。
- OSS外部表只是记录与OSS目录的映射关系。当删除OSS外部表时，不会删除映射的OSS目录下的数据文件。
- 如果OSS数据文件类型为归档文件，需要先解冻文件。更多解冻操作，请参见解冻文件。

操作入口

MaxCompute支持您在如下平台创建OSS外部表。

创建方式	平台
基于MaxCompute SQL创建OSS外部表	MaxCompute客户端
	使用控制台（查询编辑器）连接
	DataWorks控制台-ODPS SQL节点
	MaxCompute Studio-SQL程序
以可视化方式创建OSS外部表	MaxCompute Studio-SQL程序
	DataWorks控制台-外部表

创建OSS外部表语法

创建OSS外部表的场景及相应语法格式如下。详细语法参数及属性列表信息，请参见参考：语法参数说明、参考：with serdeproperties属性列表和参考：tblproperties属性列表。

场景	语法格式	支持读取或写入OSS的数据文件格式
通过内置文本数据解析器创建外部表	create external table [if not exists] <mc_oss_extable_name> (<col_name> <data_type>, ...) [partitioned by (<col_name> <data_type>, ...)] stored by '<StorageHandler>' with serdeproperties (['<property_name>'='<property_value>',...]) location '<oss_location>';	• CSV • 以GZIP方式压缩的CSV • TSV • 以GZIP方式压缩的TSV

场景	语法格式	支持读取或写入OSS的数据文件格式
通过内置开源数据解析器创建外部表	create external table [if not exists] <mc_oss_extable_name> (<col_name> <data_type>, ...) [partitioned by (<col_name> <data_type>, ...)] [row format serde '<serde_class>' [with serdeproperties (['<property_name>'='<property_value>',...])] stored as <file_format> location '<oss_location>' [using '<resource_name>'] [tblproperties ('<tbproperty_name>'='<tbproperty_value>',...)];	• PARQUET • 以SNAPPY、GZIP或LZO方式压缩的PARQUET • TEXTFILE（JSON、TEXT） • 以SNAPPY、LZO、BZ2、GZ、DEFLATE方式压缩的TEXTFILE • ORC • 以SNAPPY、ZLIB方式压缩的ORC • AVRO • SEQUENCEFILE
通过自定义解析器创建外部表	create external table [if not exists] <mc_oss_extable_name> (<col_name> <date_type>, ...) [partitioned by (<col_name> <data_type>, ...)] stored by '<StorageHandler>' with serdeproperties (['<property_name>'='<property_value>',...]) location '<oss_location>' using '<jar_name>';	除上述格式外的数据文件。

补全OSS外部表分区数据语法

当您创建的OSS外部表为分区表时，需要额外执行引入分区数据的操作。MaxCompute会根据您创建OSS外部表时指定的目录，自动补全OSS外部表的分区数据。语法格式如下。

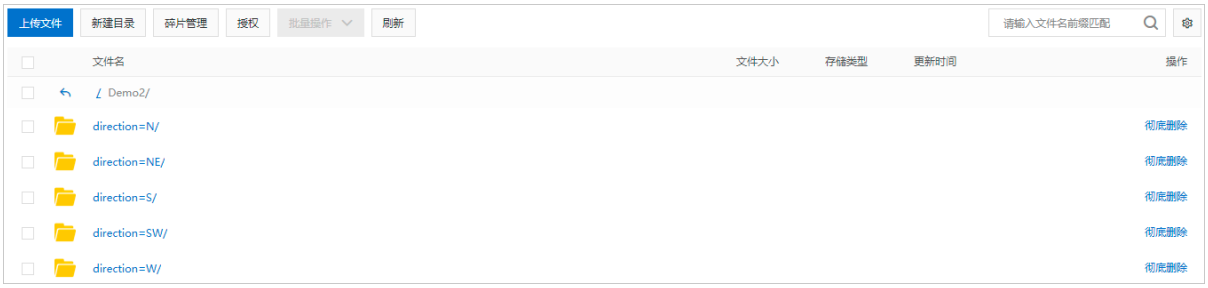
- （推荐）格式一：执行如下命令后，MaxCompute会自动解析OSS目录结构，识别分区，为OSS外部表添加分区信息。

```
msck repair table <mc_oss_extable_name> add partitions;
```

- 格式二：手动执行如下命令为OSS外部表添加分区信息。

```
alter table <mc_oss_extable_name> add partition (<col_name>=<col_value>) [add partition (<col_name>=<col_value>) ...];
```

col_name和col_value的值需要与分区数据文件所在目录名称对齐。假设，分区数据文件所在的OSS目录结构如下图，col_name对应 `direction` ，col_value对应 `N、NE、S、SW、W` 。一个 `add partition` 对应一个子目录，多个OSS子目录需要使用多个 `add partition` 。



示例：数据准备

为便于理解，为您提供示例数据如下：

- oss_endpoint：`oss-cn-hangzhou-internal.aliyuncs.com` ，即华东1（杭州）。
- Bucket名称：`oss-mc-test` 。
- 目录名称：`Demo1/` 、`Demo2/` 、`Demo3/` 和 `SampleData/` 。

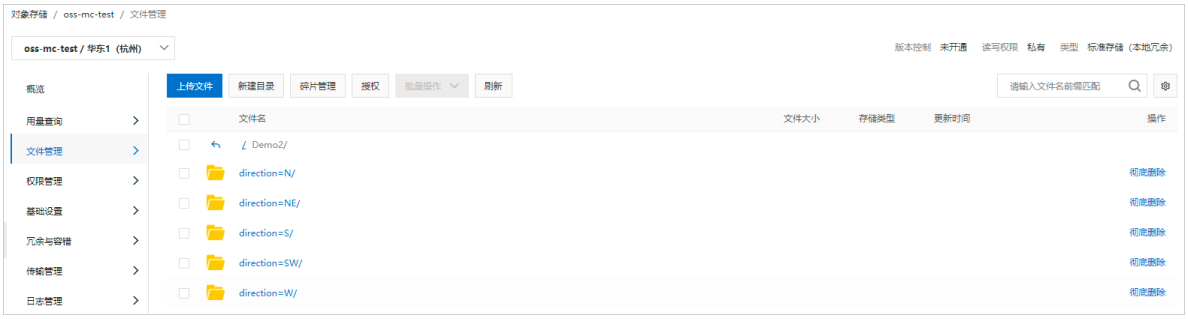
  上述四个目录下分别上传以下数据文件：

- `Demo1/` 目录下上传的文件为vehicle.csv，用于和通过内置文本数据解析器创建的非分区表建立映射关系。文件中包含的数据信息如下。

```
1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/15/2014 0:00,S
1,7,53,1,46.81006,-92.08174,9/15/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/15/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/15/2014 0:00,NE
1,10,31,1,46.81006,-92.08174,9/15/2014 0:00,N
```

- `Demo2/` 目录下包含五个子目录 `direction=N/` 、 `direction=NE/` 、 `direction=S/` 、 `direction=SW/` 和 `direction=W/` ，分别上传的文件为vehicle1.csv、vehicle2.csv、vehicle3.csv、vehicle4.csv和vehicle5.csv，用于和通过内置文本数据解析器创建的分区表建立映射关系。文件中包含的数据信息如下。

```
--vehicle1.csv
1,7,53,1,46.81006,-92.08174,9/15/2014 0:00
1,10,31,1,46.81006,-92.08174,9/15/2014 0:00
--vehicle2.csv
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00
1,9,4,1,46.81006,-92.08174,9/15/2014 0:00
--vehicle3.csv
1,6,9,1,46.81006,-92.08174,9/15/2014 0:00
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00
1,6,9,1,46.81006,-92.08174,9/15/2014 0:00
--vehicle4.csv
1,8,63,1,46.81006,-92.08174,9/15/2014 0:00
--vehicle5.csv
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00
```



- `Demo3/` 目录下上传的文件为vehicle.csv.gz，压缩包内文件为vehicle.csv，与 `Demo1/` 目录下的文件内容相同，用于和携带压缩属性的OSS外部表建立映射关系。



- `SampleData/` 目录下上传的文件为vehicle6.csv，用于和通过开源数据解析器创建的OSS外部表建立映射关系。文件中包含的数据信息如下。

```
1|1|51|1|46.81006|-92.08174|9/14/2014 0:00|S
1|2|13|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|3|48|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|4|30|1|46.81006|-92.08174|9/14/2014 0:00|W
1|5|47|1|46.81006|-92.08174|9/14/2014 0:00|S
1|6|9|1|46.81006|-92.08174|9/14/2014 0:00|S
1|7|53|1|46.81006|-92.08174|9/14/2014 0:00|N
1|8|63|1|46.81006|-92.08174|9/14/2014 0:00|SW
1|9|4|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|10|31|1|46.81006|-92.08174|9/14/2014 0:00|N
```

示例：通过内置文本数据解析器创建OSS外部表-非分区表

与示例：数据准备中的 `Demo1/` 目录建立映射关系。创建OSS外部表命令示例如下。

```
create external table if not exists mc_oss_csv_external1
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties (
 'odps.properties.rolearn'='acs:ram::xxxxxx:role/aliyunodpsdefaultrole'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo1/';
```

您可以执行 `desc extended mc_oss_csv_external1;` 命令查看创建好的OSS外部表结构信息。

示例：通过内置文本数据解析器创建OSS外部表-分区表

与示例：数据准备中的 `Demo2/` 目录建立映射关系。创建OSS外部表并引入分区数据命令示例如下。

```
create external table if not exists mc_oss_csv_external2
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string
)
partitioned by (
direction string
)
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties (
 'odps.properties.rolearn'='acs:ram::xxxxxx:role/aliyunodpsdefaultrole'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo2/';
--引入分区数据。
msck repair table mc_oss_csv_external2 add partitions;
--等效于如下语句。
alter table mc_oss_csv_external2 add partition (direction = 'N') partition (direction = 'NE') partition (direction = 'S') partiti
on (direction = 'SW') partition (direction = 'W');
```

您可以执行 `desc extended mc_oss_csv_external2;` 命令查看创建好的外部表结构信息。

示例：通过内置文本数据解析器创建OSS外部表-压缩数据

与示例：数据准备中的 `Demo3/` 目录建立映射关系。创建OSS外部表命令示例如下。

```
create external table if not exists mc_oss_csv_external3
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties (
 'odps.properties.rolearn'='acs:ram::xxxxxx:role/aliyunodpsdefaultrole',
 'odps.text.option.gzip.input.enabled'='true',
 'odps.text.option.gzip.output.enabled'='true'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo3/';
```

您可以执行 `desc extended mc_oss_csv_external3;` 命令查看创建好的外部表结构信息。

示例：通过内置开源数据解析器创建OSS外部表

- 映射PARQUET数据文件

```
create external table [if not exists] <mc_oss_extable_name>
(
<col_name> <data_type>,
...
)
[partitioned by (<col_name> <data_type>, ...)]
stored as parquet
location '<oss_location>';
select ... from <mc_oss_extable_name> ...;
```

- 映射TEXTFILE数据文件
  - **关联TEXT数据建表示例**

```
create external table [if not exists] <mc_oss_extable_name>
(
<col_name> <data_type>,
...
)
[partitioned by (<col_name> <data_type>, ...)]
row format serde 'org.apache.hive.hcatalog.data.JsonSerDe'
stored as textfile
location '<oss_location>';
select ... from <mc_oss_extable_name> ...;
```

建表时不支持自定义 `row format` 。 `row format` 默认值如下。

```
FIELDS TERMINATED BY : '\001'
ESCAPED BY : '\'
COLLECTION ITEMS TERMINATED BY : '\002'
MAP KEYS TERMINATED BY : '\003'
LINES TERMINATED BY : '\n'
NULL DEFINED AS : '\N'
```

  - **关联CSV数据建表示例**

```
--关闭native的text reader。
set odps.ext.hive.lazy.simple.serde.native=false;
--创建OSS外部表。
create external table <mc_oss_extable_name>
(
<col_name> <data_type>,
...
)
row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
with serdeproperties (
"separatorChar" = ",",
"quoteChar"= '"',
"escapeChar"= "\\"
)
stored as textfile
location '<oss_location>'
tblproperties (
"skip.header.line.count"="1",
"skip.footer.line.count"="1"
);
select ... from <mc_oss_extable_name> ...;
```

> **说明** OpenCSVSerde只支持STRING类型。执行DML语句时，您需要在SQL语句前增加 `set odps.sql.hive.compatible=true;` 命令，与SQL语句一起提交执行。

- **关联JSON数据建表示例**

```
create external table [if not exists] <mc_oss_extable_name>
(
<col_name> <data_type>,
...
)
[partitioned by (<col_name> <data_type>, ...)]
row format serde 'org.apache.hive.hcatalog.data.JsonSerDe'
stored as textfile
location '<oss_location>';
select ... from <mc_oss_extable_name> ...;
```

- 映射ORC数据文件

```
create external table [if not exists] <mc_oss_extable_name>
(
<col_name> <data_type>,
...
)
[partitioned by (<col_name> <data_type>, ...)]
stored as orc
location '<oss_location>';
select ... from <mc_oss_extable_name> ...;
```

- 映射AVRO数据文件

```
create external table [if not exists] <mc_oss_extable_name>
(
<col_name> <data_type>,
...
)
[partitioned by (<col_name> <data_type>, ...)]
stored as avro
location '<oss_location>';
select ... from <mc_oss_extable_name> ...;
```

- 映射SEQUENCEFILE数据文件

```
create external table [if not exists] <mc_oss_extable_name>
(
<col_name> <data_type>,
...
)
[partitioned by (<col_name> <data_type>, ...)]
stored as sequencefile
location '<oss_location>';
select ... from <mc_oss_extable_name> ...;
```

示例：通过自定义解析器创建OSS外部表

与示例：数据准备中的 `SampleData/` 目录建立映射关系。

操作流程如下：

1. 使用MaxCompute Studio开发TextExtractor.java、TextOutputer.java、SplitReader.java和TextStorageHandler.java四个Java程序。
   更多开发Java程序信息，请参见开发UDF。
2. 通过MaxCompute Studio的一键式打包功能，将TextStorageHandler.java打包上传为MaxCompute资源。
   假设，此处打包的资源名称为 `javatest-1.0-SNAPSHOT.jar` 。更多打包上传信息，请参见打包、上传及注册。
3. 执行如下命令创建OSS外部表。

```
create external table if not exists ambulance_data_txt_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
stored by 'com.aliyun.odps.udf.example.text.TextStorageHandler'
  with serdeproperties (
'delimiter'='|',
'odps.properties.rolearn'='acs:ram::xxxxxxxxxxxx:role/aliyunodpsdefaultrole'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/SampleData/'
using 'javatest-1.0-SNAPSHOT.jar';
```

其中delimeter是OSS数据的分隔符名称。

您可以执行 `desc extended ambulance_data_txt_external;` 命令查看创建好的外部表结构信息。

> **说明** 对于自定义解析器，默认不会对数据分片，防止解析器出现正确性问题。如果您确认可以处理分片，需要使用如下命令允许进行数据分片，即启动多个mapper。
>
> ```
> set odps.sql.unstructured.data.single.file.split.enabled=true;
> ```

参考：语法参数说明

上述语法中各参数的含义如下。

参数名称	可选/必填	说明
mc_oss_extable_name	必填	待创建的OSS外部表的名称。 表名大小写不敏感，在查询外部表时，无需区分大小写，且不支持强制转换大小写。
col_name	必填	OSS外部表的列名称。 在读取OSS数据场景，创建的OSS外部表结构必须与OSS数据文件结构保持一致，否则无法成功读取OSS数据。
data_type	必填	OSS外部表的列数据类型。 在读取OSS数据场景，创建的OSS外部表各列数据类型必须与OSS数据文件各列数据类型保持一致，否则无法成功读取OSS数据。
partitioned by (<col_name> <data_type>, ...)	条件必填	当OSS中的数据文件是以分区路径方式存储时，需要携带该参数，创建分区表。 • col_name：分区列名称。 • data_type：分区列数据类型。

StorageHandler	条件必填	当您使用内置文本数据解析器或自定义解析器创建OSS外部表时，需要携带该参数。MaxCompute解析器主要分为如下两类：
		• 内置文本数据解析器：
		  ○ `com.aliyun.odps.CsvStorageHandler`：读写CSV或以GZIP方式压缩的CSV数据文件。
		  ○ `com.aliyun.odps.TsvStorageHandler`：读写TSV或以GZIP方式压缩的TSV数据文件。
		• 自定义解析器：您通过编写MaxCompute UDF自定义的解析器。更多编写MaxCompute UDF信息，请参见开发UDF。

'<property_name>'='<property_value>'	必填	OSS外部表扩展属性。详细属性列表，请参见参考：with serdeproperties属性列表。

参数名称	可选/必填	说明

oss_location	必填	数据文件所在OSS路径。格式为 `oss://<oss_endpoint>/<Bucket名称>/<OSS目录名称>/` 。MaxCompute默认会读取该路径下的所有数据文件。
		• oss_endpoint：OSS访问域名信息。建议您使用OSS提供的内网域名，否则将产生OSS流量费用。更多OSS内网域名信息，请参见访问域名和数据中心。
		说明 建议数据文件存放的OSS地域与MaxCompute项目所在地域保持一致。由于MaxCompute只在部分地域部署，跨地域的数据连通性可能存在问题。
		• Bucket名称：OSS存储空间名称，即Bucket名称。更多查看存储空间名称信息，请参见列举存储空间。
		• 目录名称：OSS目录名称。目录后不需要指定文件名。
		--**正确用法。**
		`oss://oss-cn-shanghai-internal.aliyuncs.com/oss-mc-test/Demo1/`
		--**错误用法。**
		`http://oss-cn-shanghai-internal.aliyuncs.com/oss-mc-test/Demo1/`
		-- **不支持HTTP连接。**
		`https://oss-cn-shanghai-internal.aliyuncs.com/oss-mc-test/Demo1/`
		-- **不支持HTTPS连接。**
		`oss://oss-cn-shanghai-internal.aliyuncs.com/Demo1`
		-- **连接地址错误。**
		`oss://oss-cn-shanghai-internal.aliyuncs.com/oss-mc-test/Demo1/vehicle.csv` -- **不需要指定文件名。**

jar_name	条件必填	当您使用自定义解析器创建OSS外部表时，需要携带该参数。指定自定义解析器代码对应的JAR包。该JAR包需要添加为MaxCompute项目资源。 更多添加资源信息，请参见添加资源。
serde_class	条件可选	指定MaxCompute内置的开源数据解析器。如果数据文件格式为TEXTFILE，必须携带该参数，其他场景可以不配置。 MaxCompute支持的开源数据格式对应的serde_class如下： • PARQUET：`org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe` • TEXTFILE：`org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe`、`org.apache.hive.hcatalog.data.JsonSerDe`、`org.apache.hadoop.hive.serde2.OpenCSVSerde` • ORC：`org.apache.hadoop.hive.ql.io.orc.OrcSerde` • AVRO：`org.apache.hadoop.hive.serde2.avro.AvroSerDe` • SEQUENCEFILE：`org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe` • RCFILE：`org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe` • ORCFILE：`org.apache.hadoop.hive.ql.io.orc.OrcSerde`
file_format	条件必填	当OSS数据文件为开源格式时，需要携带该参数。指定OSS数据文件的格式。 **说明** 单个文件大小不能超过3 GB，如果文件过大，建议拆分。
resource_name	条件可选	当您使用自定义的serde class时，需要指定依赖的资源。资源中包含了自定义的serde class。 serde class相关的JAR包需要添加为MaxCompute项目资源。 更多添加资源信息，请参见添加资源。
'<tbproperty_name>'='<tbproperty_value>'	条件可选	OSS外部表扩展属性。详细属性列表，请参见参考：tblproperties属性列表。

参考：with serdeproperties属性列表

property_name	使用场景	说明	property_value	默认值
odps.properties.rolearn	使用STS模式授权时，请添加该属性。	指定RAM中AliyunODPSDefaultRole的ARN信息。	您可以通过RAM控制台中的角色详情获取。	无

property_name	使用场景	说明	property_value	默认值
delimiter	需要明确CSV或TSV数据文件的列分隔符时，请添加该属性。	MaxCompute基于设置的分隔符，正确读取OSS数据文件中的各列数据。	单个字符	英文逗号（,）
odps.text.option.gzip.input.enabled	当需要读取以GZIP方式压缩的CSV或TSV文件数据时，请添加该属性。	**CSV、TSV压缩属性**。配置该参数值为True时，MaxCompute才可以正常读取压缩文件，否则会读取失败。	• True • False	False
odps.text.option.gzip.output.enabled	当需要将数据以GZIP压缩方式写入OSS时，请添加该属性。	**CSV、TSV压缩属性**。配置该参数值为True时，MaxCompute才能将数据以GZIP压缩方式写入OSS，否则不压缩。	• True • False	False
odps.text.option.header.lines.count	当OSS数据文件为CSV或TSV，且需要忽略OSS数据文件中的前N行时，请添加该属性。	MaxCompute读取OSS数据文件时，会忽略指定的行数。	非负整数	0
odps.text.option.null.indicator	当OSS数据文件为CSV或TSV，且需要定义OSS数据文件中NULL的解析规则时，请添加该属性。	通过该参数配置的字符串会被解析为SQL中的NULL。例如 `\N` 指代NULL，则 `a,\N,b` 会解析为 `a, NULL, b` 。	字符串	空字符串
odps.text.option.ignore.empty.lines	当OSS数据文件为CSV或TSV，且需要定义OSS数据文件中空行的处理规则时，请添加该属性。	配置该参数值为True时，MaxCompute会忽略数据文件中的空行，否则会读取空行。	• True • False	True
odps.text.option.encoding	当OSS数据文件为CSV或TSV，且OSS数据文件编码规则非默认编码规则时，请添加该属性。	确保此处配置的编码规则与OSS数据文件编码规则保持一致，否则MaxCompute无法成功读取数据。	• UTF-8 • UTF-16 • US-ASCII • GBK	UTF-8
odps.text.option.delimiter	当需要明确CSV或TSV数据文件的列分隔符时，请添加该属性。	确保此处配置的列分隔符可以正确读取OSS数据文件的每一列，否则MaxCompute读取的数据会出现错位问题。	单个字符	英文逗号（,）
odps.text.option.use.quote	当CSV或TSV数据文件中的字段包含换行（CRLF）、双引号或英文逗号时，请添加该属性。	当CSV某个字段中包含换行、双引号（需要在 `"` 前再加 `"` 转义）或英文逗号时，整个字段必须用双引号（""）括起来作为列分隔符。该参数指定是否识别CSV的列分隔符 `"` 。	• True • False	False
mcfed.parquet.compression	当需要将PARQUET数据以压缩方式写入OSS时，请添加该属性。	**PARQUET压缩属性**。PARQUET数据默认不压缩。如果指定了压缩方式，请在创建OSS外部表语句前添加 `set odps.sql.hive.compatible=true;` 命令，与SQL语句一起提交。	• SNAPPY • GZIP	无
parquet.file.cache.size	在处理PARQUET数据场景，如果需要提升读OSS数据文件性能，请添加该属性。	指定读OSS数据文件时，可缓存的数据量，单位为KB。	1024	无
parquet.io.buffer.size	在处理PARQUET数据场景，如果需要提升读OSS数据文件性能，请添加该属性。	指定OSS数据文件大小超过1024 KB时，可缓存的数据量，单位为KB。	4096	无
separatorChar	当需要明确以TEXTFILE格式保存的CSV数据的列分隔符时，请添加该属性。	指定CSV数据列分隔符。	单个字符串	英文逗号（,）
quoteChar	当以TEXTFILE格式保存的CSV数据中的字段包含换行、双引号或英文逗号时，请添加该属性。	指定CSV数据的引用符。	单个字符串	无
escapeChar	当需要明确以TEXTFILE格式保存的CSV数据的转义规则时，请添加该属性。	指定CSV数据的转义符。	单个字符串	无

参考：tblproperties属性列表

property_name	使用场景	说明	property_value	默认值
skip.header.line.count	当需要忽略以TEXTFILE格式保存的CSV文件中的前N行数据时，请添加该属性。	MaxCompute读取OSS数据时，会忽略从首行开始指定行数的数据。	非负整数	无
skip.footer.line.count	当需要忽略以TEXTFILE格式保存的CSV文件中的后N行数据时，请添加该属性。	MaxCompute读取OSS数据时，会忽略从尾行开始指定行数的数据。	非负整数	无
mcfed.orc.compress	当需要将ORC数据以压缩方式写入OSS时，请添加该属性。	**ORC压缩属性**。指定ORC数据的压缩方式。如果指定了压缩方式，请在创建OSS外部表语句前添加 `set odps.sql.hive.compatible=true;` 命令，与SQL语句一起提交。	• SNAPPY • ZLIB	无
mcfed.mapreduce.output.fileoutputformat.compress	当需要将TEXTFILE数据文件以压缩方式写入OSS时，请添加该属性。	**TEXTFILE压缩属性**。配置该参数值为True时，MaxCompute才可以将TEXTFILE数据文件以压缩方式写入OSS，否则不压缩。如果指定了压缩方式，请在创建OSS外部表语句前添加 `set odps.sql.hive.compatible=true;` 命令，与SQL语句一起提交。	• True • False	False
mcfed.mapreduce.output.fileoutputformat.compress.codec	当需要将TEXTFILE数据文件以压缩方式写入OSS时，请添加该属性。	**TEXTFILE压缩属性**。设置TEXTFILE数据文件的压缩方式。如果指定了压缩方式，请在创建OSS外部表语句前添加 `set odps.sql.hive.compatible=true;` 命令，与SQL语句一起提交。	• com.hadoop.compression.lzo.LzoCodec • com.hadoop.compression.lzo.LzopCodec • org.apache.hadoop.io.compress.SnappyCodec • com.aliyun.odps.io.compress.SnappyRawCodec	无
io.compression.codecs	当OSS数据文件为Raw-Snappy格式时，请添加该属性。	配置该参数值为True时，MaxCompute才可以正常读取压缩数据，否则MaxCompute无法成功读取数据。	com.aliyun.odps.io.compress.SnappyRawCodec	无

10.4.2. 读取OSS数据

创建OSS外部表后，您可以通过外部表读取存储在OSS目录中的数据文件。

背景信息

完成OSS外部表创建后，您可以根据需要选择如下方式之一对OSS外部表进行操作：

- （推荐）方式一：将OSS的开源格式数据导入MaxCompute内部表，然后再读取OSS的数据。

  如果直接使用OSS外部表，每次读取数据都涉及OSS的I/O操作，且MaxCompute本身针对内部存储设计的许多高性能优化都无法应用，因此性能上会有所损失。如果您需要对数据进行反复计算或对计算的高效性比较敏感，推荐先在MaxCompute项目中创建一个与OSS外部表Schema相同的内部表，然后将OSS上的数据导入MaxCompute内部表，在内部表上执行复杂查询语句，以便获得更高的计算性能。

  命令示例如下。

  ```
  create table <table_internal> like <mc_oss_extable_name>;
  insert overwrite table <table_internal> select * from <mc_oss_extable_name>;
  ```

- 方式二：直接读取OSS数据，与MaxCompute内部表的操作相同。

  OSS外部表会被当作一个普通的内部表一样使用，不同之处在于，MaxCompute是直接从OSS读取对应的数据。

注意事项

在使用外部表读取OSS数据时，需要注意：

- 您需要创建好OSS外部表后，才可以通过外部表读取OSS数据。更多MaxCompute支持的OSS数据文件类型及创建OSS外部表命令信息，请参见创建OSS外部表语法。
- SQL语句中涉及到复杂数据类型时，需要在SQL语句前添加 `set odps.sql.type.system.odps2=true;` 命令，和SQL语句一起提交执行。更多数据类型信息，请参见数据类型版本说明。
- 对于映射开源数据的OSS外部表，您需要在Session级别设置 `set odps.sql.hive.compatible=true;` 后再读取OSS数据，否则会报错。

示例：读取OSS数据-非分区表（内置文本数据解析器）

基于示例：通过内置文本数据解析器创建OSS外部表-非分区表中创建的OSS外部表mc_oss_csv_external1读取OSS数据。

使用MaxCompute客户端，从OSS读取数据。命令示例如下。

```
select recordId, patientId, direction from mc_oss_csv_external1 where patientId > 25;
```

返回结果如下。

```
+------------+------------+------------+
| recordid   | patientid  | direction  |
+------------+------------+------------+
| 1          | 51         | S          |
| 3          | 48         | NE         |
| 4          | 30         | W          |
| 5          | 47         | S          |
| 7          | 53         | N          |
| 8          | 63         | SW         |
| 10         | 31         | N          |
+------------+------------+------------+
```

示例：读取OSS数据-分区表（内置文本数据解析器）

基于示例：通过内置文本数据解析器创建OSS外部表-分区表中创建的OSS外部表mc_oss_csv_external2读取OSS数据。

使用MaxCompute客户端，从OSS读取数据。命令示例如下。

```
select recordId, patientId, direction from mc_oss_csv_external2 where direction = 'NE';
```

返回结果如下。

```
+------------+------------+------------+
| recordid   | patientid  | direction  |
+------------+------------+------------+
| 2          | 13         | NE         |
| 3          | 48         | NE         |
| 9          | 4          | NE         |
+------------+------------+------------+
```

示例：读取OSS压缩数据（内置文本数据解析器）

基于示例：通过内置文本数据解析器创建OSS外部表-压缩数据中创建的OSS外部表mc_oss_csv_external3读取OSS数据。

使用MaxCompute客户端，从OSS读取数据。命令示例如下。

```
select recordId, patientId, direction from mc_oss_csv_external3 where patientId > 25;
```

> 说明 如果OSS压缩数据为开源数据格式，需要在SQL语句前添加 `set odps.sql.hive.compatible=true;` 命令，与SQL语句一起提交执行。

返回结果如下。

```
+------------+------------+------------+
| recordid   | patientid  | direction  |
+------------+------------+------------+
| 1          | 51         | S          |
| 3          | 48         | NE         |
| 4          | 30         | W          |
| 5          | 47         | S          |
| 7          | 53         | N          |
| 8          | 63         | SW         |
| 10         | 31         | N          |
+------------+------------+------------+
```

示例：读取OSS文本数据（自定义解析器）

基于示例：通过自定义解析器创建OSS外部表中创建的OSS外部表ambulance_data_txt_external读取OSS数据。

使用MaxCompute客户端，从OSS读取数据。命令示例如下。

```
select recordId, patientId, direction from ambulance_data_txt_external where patientId > 25;
```

返回结果如下。

```
+----------+-----------+-----------+
| recordid | patientid | direction |
+----------+-----------+-----------+
| 1        | 51        | S         |
| 3        | 48        | NE        |
| 4        | 30        | W         |
| 5        | 47        | S         |
| 7        | 53        | N         |
| 8        | 63        | SW        |
| 10       | 31        | N         |
+----------+-----------+-----------+
```

10.4.3. 读取以分区方式存储的OSS数据

MaxCompute支持创建OSS外部表为分区表，访问OSS上以分区方式存储的数据，通过该方式可降低读取数据量并提升数据处理效率。本文为您介绍MaxCompute支持的OSS标准分区路径格式和自定义分区路径格式。

背景信息

创建OSS外部表后，MaxCompute会全量扫描OSS目录下的所有数据，包括子目录下的数据文件。但当数据量比较大时，对全目录扫描会产生不必要的I/O消耗以及数据处理时间。通常，解决该问题有如下两种方法：

- （推荐）方式一：在OSS上将数据以标准分区路径或自定义分区路径方式存储。通过MaxCompute新建OSS外部表时，您需要在建表语句指定分区及oss_location信息。推荐您使用标准分区路径。更多分区路径信息，请参见标准分区路径格式和自定义分区路径格式。
- 方式二：规划多个数据存放路径。创建多个OSS外部表来读取各个路径下的数据。即每个OSS外部表指向OSS数据的一个子集。该方式操作比较繁琐，数据管理效果不佳。不推荐使用该方式。

标准分区路径格式

标准分区路径格式如下。

```
oss://<oss_endpoint>/<Bucket名称>/<目录名称>/<partitionKey1=value1>/<partitionKey2=value2>/...
```

> **说明** 因为OSS数据是离线准备的，即通过OSS控制台或其他OSS工具上传到OSS中，所以数据的路径格式取决于上传时的格式。

参数名称	可选/必填	说明
oss_endpoint	必填	OSS访问域名信息。建议您使用OSS提供的内网域名，否则将产生OSS流量费用。更多OSS内网域名信息，请参见访问域名和数据中心。 **说明** 建议数据文件存放的OSS地域与MaxCompute项目所在地域保持一致。由于MaxCompute只在部分地域部署，跨地域的数据连通性可能存在问题。
Bucket名称	必填	OSS存储空间名称，即Bucket名称。查看存储空间名称操作，请参见列举存储空间。
目录名称	必填	指定OSS目录名称。目录后不需要指定文件名。
partitionKey	必填	OSS外部表的分区列名。
value	必填	OSS外部表的分区列值。

例如，某公司会将每天产生的Log文件以CSV格式存放在OSS上，并通过MaxCompute以天为周期处理数据。则存储OSS数据的标准分区路径应设置如下。

```
oss://oss-odps-test/log_data/year=2016/month=06/day=01/logfile
oss://oss-odps-test/log_data/year=2016/month=06/day=02/logfile
oss://oss-odps-test/log_data/year=2016/month=07/day=10/logfile
oss://oss-odps-test/log_data/year=2016/month=08/day=08/logfile
...
```

基于上述分区存储路径，创建OSS外部表、引入并分析分区数据的命令示例如下。

```
--创建OSS外部表。
create external table log_table_external (
    click STRING,
    ip STRING,
    url STRING
  )
partitioned by (
    year STRING,
    month STRING,
    day STRING
  )
stored by 'com.aliyun.odps.CsvStorageHandler'
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/log_data/';
--引入OSS分区数据。
alter table log_table_external add partition (year = '2016', month = '06', day = '01');
alter table log_table_external add partition (year = '2016', month = '06', day = '02');
alter table log_table_external add partition (year = '2016', month = '07', day = '10');
alter table log_table_external add partition (year = '2016', month = '08', day = '08');
...
--分析数据。只访问log_data/year=2016/month=06/day=01子目录下的文件（logfile），不会对整个log_data目录做全量数据扫描。
select count(distinct(ip)) from log_table_external where year = '2016' and month = '06' and day = '01';
```

完整的读取标准分区路径数据示例，请参见示例：通过内置文本数据解析器创建OSS外部表-分区表。

自定义分区路径格式

如果OSS上的数据实际上已按照分区方式存储，只是路径不是标准分区路径格式时，MaxCompute支持将不同的子目录绑定至不同的分区。

自定义分区路径格式示例如下，只有分区列值，无分区列名。

```
oss://oss-odps-test/log_data_customized/2016/06/01/logfile
oss://oss-odps-test/log_data_customized/2016/06/02/logfile
oss://oss-odps-test/log_data_customized/2016/07/10/logfile
oss://oss-odps-test/log_data_customized/2016/08/08/logfile
...
```

创建OSS外部表后，通过 `alter table ... add partition ... location ...` 命令指定子目录，将不同的子目录绑定到不同的分区，命令示例如下。

```
alter table log_table_external add partition (year = '2016', month = '06', day = '01')
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-odps-test/log_data_customized/2016/06/01/';
alter table log_table_external add partition (year = '2016', month = '06', day = '02')
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-odps-test/log_data_customized/2016/06/02/';
alter table log_table_external add partition (year = '2016', month = '07', day = '10')
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-odps-test/log_data_customized/2016/07/10/';
alter table log_table_external add partition (year = '2016', month = '08', day = '08')
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-odps-test/log_data_customized/2016/08/08/';
```

基于该方式，自定义分区路径后，即使数据存放不符合推荐的标准分区路径格式，您也可以正确地对子目录数据进行访问。

10.4.4. 将数据写入OSS

MaxCompute支持您通过INSERT方式将MaxCompute项目中的数据通过映射的OSS外部表直接输出到OSS目录。本文为您介绍向OSS写入数据的方法并提供示例。

背景信息

与读取OSS数据类似，MaxCompute支持将内部表数据或处理外部表得到的数据写入OSS：

- 通过内置文本或开源数据解析器将数据写入OSS

  当需要将数据以CSV、TSV、开源格式或MaxCompute支持的压缩格式写入OSS时，您可以通过基于MaxCompute内置文本数据解析器或开源数据解析器创建的OSS外部表，执行INSERT操作将数据写入OSS。

  实现示例如下：

  - 示例：通过内置文本数据解析器将数据写入OSS-非分区路径
  - 示例：通过内置文本数据解析器将数据写入OSS-分区路径
  - 示例：通过内置文本数据解析器将数据以压缩方式写入OSS
  - 示例：通过内置开源数据解析器将数据写入OSS

- 通过自定义解析器将数据写入OSS

  当需要将数据以自定义格式文件写入OSS时，您可以通过基于自定义解析器创建的OSS外部表，执行INSERT操作将数据写入OSS。

实现示例请参见示例：通过自定义解析器将数据写入OSS。

- 通过OSS分片上传功能将数据写入OSS

  当需要将数据以开源格式写入OSS时，您可以通过基于开源数据解析器创建的OSS外部表，及OSS的分片上传功能，执行INSERT操作将数据写入OSS。该功能需要在Session级别或Project级别设置 `odps.sql.unstructured.oss.commit.mode` 属性进行开启或关闭。默认关闭。更多分片上传功能信息，请参见分片上传。

  在使用该功能前，您需要确认MaxCompute项目已启用jobconf2来生成作业执行计划，即 `odps.sql.jobconf.odps2` 属性值为True。

  > **说明** `odps.sql.jobconf.odps2` 属性默认值为True，如果取值非True，请在Session级别执行 `set odps.sql.jobconf.odps2=true;` 命令进行设置。

  `odps.sql.unstructured.oss.commit.mode` 属性设置不同取值时的实现原理如下：

  - 取值为False：MaxCompute写入到OSS外部表的数据，会存储在LOCATION目录下的.odps文件夹中。.odps文件夹中维护了一个.meta文件，用于保证MaxCompute数据的一致性。.odps的内容只有MaxCompute能正确处理，在其他数据处理引擎中可能无法正确解析，会导致报错。
  - 取值为True：MaxCompute使用分片上传功能，以 `two-phase commit` 的方式保证数据的一致性，同时也不会有.odps目录以及.meta文件。兼容其他数据处理引擎。

  > **注意** `insert overwrite` 操作场景，在极端情况下如果作业运行失败会出现与预期不一致的问题，表现为旧数据被删除，但是新数据并没有写入。
  >
  > 问题原因：新写入的数据因为极低概率的硬件故障或元数据更新失败等原因没有成功写入目标表，且OSS的删除操作不支持回滚，导致被删除的旧数据无法恢复。
  >
  > 解决方案：
  >
  > - 如果基于旧数据再覆写OSS外部表，例如 `insert overwrite table T select * from table T;` ，需要提前做好OSS数据备份，作业运行失败时，再基于备份的旧数据覆写OSS外部表。
  > - 如果 `insert overwrite` 作业可以重复提交，当作业运行失败时重新提交作业即可。

前提条件

在向OSS写入数据前，请您确认已完成如下操作：

- 已授予访问OSS的权限。

  更多授权信息，请参见OSS的STS模式授权。

- （可选）已准备好OSS存储空间（Bucket）、OSS目录及OSS数据文件。

  > **说明** MaxCompute已支持在OSS侧自动创建目录，对于携带外部表及UDF的SQL语句，您可以通过一条SQL语句执行读写外部表及UDF的操作。原手动创建目录方式仍然支持。

  更多OSS存储空间、OSS目录、上传数据文件操作信息，请参见创建存储空间、创建目录和上传文件。

示例：通过内置文本数据解析器将数据写入OSS-非分区路径

读取示例：通过内置文本数据解析器创建OSS外部表-非分区表中创建的OSS外部表mc_oss_csv_external1的数据，并以CSV格式写入OSS目录 `oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo1/output/` 。

操作流程如下：

1. 登录MaxCompute客户端，执行如下命令创建OSS外部表映射目标OSS目录。

```
create external table if not exists mc_oss_csv_external4
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties (
 'odps.properties.rolearn'='acs:ram::xxxxxx:role/aliyunodpsdefaultrole'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo1/output/';
```

2. 通过MaxCompute客户端对外部表执行 `insert overwrite` 或 `insert into` 命令，将数据写入OSS。

```
insert into table mc_oss_csv_external4 select * from mc_oss_csv_external1;
```

执行成功后，您可以在OSS目录下查看到导出的文件。output文件夹下会产生一个.odps文件夹，包含.meta文件和保存.csv文件的文件夹。例如 `output/.odps/20210330********/R2_1_0_0-0_TableSink1-0-.csv` 。



说明

- .odps文件夹中的.meta文件为MaxCompute额外输出的宏数据文件，用于记录当前文件夹中有效的数据。正常情况下，如果INSERT操作成功，可以认为当前文件夹的所有数据均是有效数据。只有在有作业失败的情况下，需要对该宏数据进行解析。即使是在作业中途失败或被中止的情况下，对于INSERT OVERWRITE操作，再运行一次即可。
- 对于MaxCompute内置的TSV或CSV解析器，处理产生的文件数目与对应SQL的并发度相同。
- 如果 `insert overwrite ... select ... from ...;` 操作在源数据表from_tablename上分配了1000个Mapper，则最后将产生1000个TSV或CSV文件。
- 如果您需要控制生成文件的数目，可以通过MaxCompute的各种灵活语义和配置来实现。如果Outputer在Mapper里，可以通过调整 `odps.stage.mapper.split.size` 的大小来控制Mapper的并发数，从而调整产生的文件数目。如果Outputer在Reducer或Joiner里，也可以分别通过 `odps.stage.reducer.num` 和 `odps.stage.joiner.num` 来调整。

示例：通过内置文本数据解析器将数据写入OSS-分区路径

读取示例：通过内置文本数据解析器创建OSS外部表-分区表中创建的OSS外部表mc_oss_csv_external2的数据，并以CSV格式写入OSS目录 `oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo2/output/` 。

操作流程如下：

1. 登录MaxCompute客户端，执行如下命令创建OSS外部表映射目标OSS目录。

```
create external table if not exists mc_oss_csv_external5
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string
)
partitioned by (
direction string
)
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties (
 'odps.properties.rolearn'='acs:ram::xxxxxx:role/aliyunodpsdefaultrole'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo2/output/';
```

2. 通过MaxCompute客户端对外部表执行 `insert overwrite` 或 `insert into` 命令，将数据写入OSS。

```
insert into table mc_oss_csv_external5 partition (direction) select * from mc_oss_csv_external2;
```

执行成功后，您可以在OSS目录下查看到导出的文件。output文件夹下根据INSERT语句指定的分区值生成对应的分区子目录。分区子目录中会产生一个.odps文件夹，包含.meta文件和保存.csv文件的文件夹。例如 `output/direction=N/.odps/20210330*********/R2_1_0_0-0_TableSink1-0-.csv` 。



> 说明
> - .odps文件夹中的.meta文件为MaxCompute额外输出的宏数据文件，用于记录当前文件夹中有效的数据。正常情况下，如果INSERT操作成功，可以认为当前文件夹的所有数据均是有效数据。只有在有作业失败的情况下，需要对该宏数据进行解析。即使是在作业中途失败或被中止的情况下，对于INSERT OVERWRITE操作，再运行一次即可。
> - 对于MaxCompute内置的TSV或CSV解析器，处理产生的文件数目与对应SQL的并发度相同。
> - 如果 `insert overwrite ... select ... from ...;` 操作在源数据表from_tablename上分配了1000个Mapper，则最后将产生1000个TSV或CSV文件。
> - 如果您需要控制生成文件的数目，可以通过MaxCompute的各种灵活语义和配置来实现。如果Outputer在Mapper里，可以通过调整 `odps.stage.mapper.split.size` 的大小来控制Mapper的并发数，从而调整产生的文件数目。如果Outputer在Reducer或Joiner里，也可以分别通过 `odps.stage.reducer.num` 和 `odps.stage.joiner.num` 来调整。

示例：通过内置文本数据解析器将数据以压缩方式写入OSS

读取示例：通过内置文本数据解析器创建OSS外部表-非分区表中创建的OSS外部表mc_oss_csv_external1的数据，并以GZIP压缩格式写入OSS目录 `oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo3/output/` 。

操作流程如下：

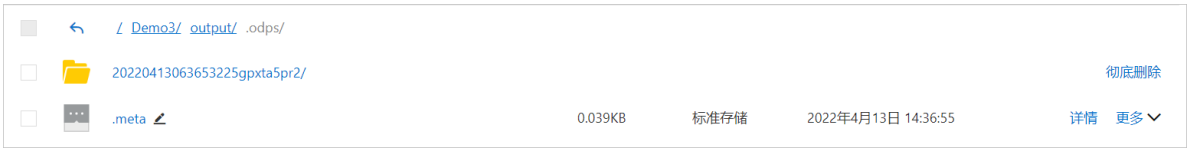
1. 登录MaxCompute客户端，执行如下命令创建OSS外部表映射目标OSS目录。

```
create external table if not exists mc_oss_csv_external6
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
stored by 'com.aliyun.odps.CsvStorageHandler'
with serdeproperties (
 'odps.properties.rolearn'='acs:ram::xxxxxx:role/aliyunodpsdefaultrole',
 'odps.text.option.gzip.input.enabled'='true',
 'odps.text.option.gzip.output.enabled'='true'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo3/output/';
```

2. 通过MaxCompute客户端对外部表执行 `insert overwrite` 或 `insert into` 命令，将数据写入OSS。

```
insert into table mc_oss_csv_external6 select * from mc_oss_csv_external1;
```

执行成功后，您可以在OSS目录下查看到导出的文件。output文件夹下会产生一个.odps文件夹，包含.meta文件和保存.gz文件的文件夹。例如 `output/.odps/20220413*********/M1_0_0-0_TableSink1-0-.csv.gz` 。



> 说明
> - .odps文件夹中的.meta文件为MaxCompute额外输出的宏数据文件，用于记录当前文件夹中有效的数据。正常情况下，如果INSERT操作成功，可以认为当前文件夹的所有数据均是有效数据。只有在有作业失败的情况下，需要对该宏数据进行解析。即使是在作业中途失败或被中止的情况下，对于INSERT OVERWRITE操作，再运行一次即可。
> - 对于MaxCompute内置的TSV或CSV解析器，处理产生的文件数目与对应SQL的并发度相同。
> - 如果 `insert overwrite ... select ... from ...;` 操作在源数据表from_tablename上分配了1000个Mapper，则最后将产生1000个TSV或CSV文件。
> - 如果您需要控制生成文件的数目，可以通过MaxCompute的各种灵活语义和配置来实现。如果Outputer在Mapper里，可以通过调整 `odps.stage.mapper.split.size` 的大小来控制Mapper的并发数，从而调整产生的文件数目。如果Outputer在Reducer或Joiner里，也可以分别通过 `odps.stage.reducer.num` 和 `odps.stage.joiner.num` 来调整。

示例：通过内置开源数据解析器将数据写入OSS

读取示例：通过内置文本数据解析器创建OSS外部表-非分区表中创建的OSS外部表mc_oss_csv_external1的数据，并以ORC格式写入OSS目录 `oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo4/output/` 。

操作流程如下：

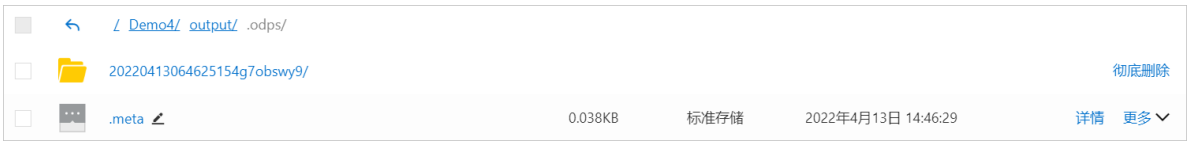
1. 登录MaxCompute客户端，执行如下命令创建OSS外部表映射目标OSS目录。

```
create external table if not exists mc_oss_orc_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
stored as orc
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/Demo4/output/';
```

2. 通过MaxCompute客户端对外部表执行 `insert overwrite` 或 `insert into` 命令，将数据写入OSS。

```
insert into table mc_oss_orc_external select * from mc_oss_csv_external1;
```

执行成功后，您可以在OSS目录下查看到导出的文件。output文件夹下会产生一个.odps文件夹，包含.meta文件和保存ORC文件的文件夹。例如 `output/.odps/20220413*********/M1_0_0-0_TableSink1` 。



说明

- .odps文件夹中的.meta文件为MaxCompute额外输出的宏数据文件，用于记录当前文件夹中有效的数据。正常情况下，如果INSERT操作成功，可以认为当前文件夹的所有数据均是有效数据。只有在有作业失败的情况下，需要对该宏数据进行解析。即使是在作业中途失败或被中止的情况下，对于INSERT OVERWRITE操作，再运行一次即可。
- 对于MaxCompute内置的TSV或CSV解析器，处理产生的文件数目与对应SQL的并发度相同。
- 如果 `insert overwrite ... select ... from ...;` 操作在源数据表from_tablename上分配了1000个Mapper，则最后将产生1000个TSV或CSV文件。
- 如果您需要控制生成文件的数目，可以通过MaxCompute的各种灵活语义和配置来实现。如果Outputer在Mapper里，可以通过调整 `odps.stage.mapper.split.size` 的大小来控制Mapper的并发数，从而调整产生的文件数目。如果Outputer在Reducer或Joiner里，也可以分别通过 `odps.stage.reducer.num` 和 `odps.stage.joiner.num` 来调整。

示例：通过自定义解析器将数据写入OSS

基于示例：通过自定义解析器创建OSS外部表中开发的自定义解析器，为您介绍MaxCompute如何将数据输出到OSS。

操作流程如下：

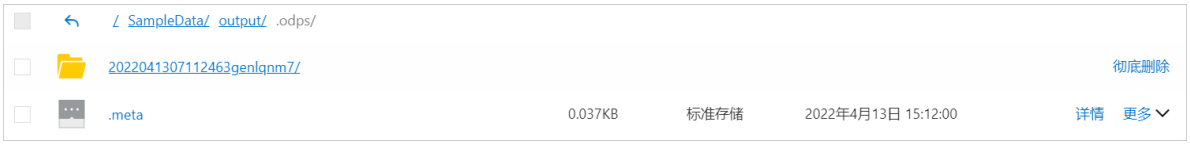
1. 登录MaxCompute客户端，执行如下命令创建OSS外部表映射目标OSS目录。

```
create external table if not exists mc_oss_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
stored by 'com.aliyun.odps.udf.example.text.TextStorageHandler'
  with serdeproperties (
'delimiter'='|',
'odps.properties.rolearn'='acs:ram::xxxxxxxxxxxxx:role/aliyunodpsdefaultrole'
)
location 'oss://oss-cn-hangzhou-internal.aliyuncs.com/oss-mc-test/SampleData/output'
using 'javatest-1.0-SNAPSHOT.jar';
```

2. 通过MaxCompute客户端对外部表执行 `insert overwrite` 或 `insert into` 命令，将数据写入OSS。

```
insert into table mc_oss_external select * from mc_oss_csv_external1;
```

执行成功后，您可以在OSS目录下查看到导出的文件。output文件夹下会产生一个.odps文件夹，包含.meta文件和保存TEXT文件的文件夹。例如 `output/.odps/20220413*********/M1_0_0-0_TableSink1-0` 。



说明

- .odps文件夹中的.meta文件为MaxCompute额外输出的宏数据文件，用于记录当前文件夹中有效的数据。正常情况下，如果INSERT操作成功，可以认为当前文件夹的所有数据均是有效数据。只有在有作业失败的情况下，需要对该宏数据进行解析。即使是在作业中途失败或被中止的情况下，对于INSERT OVERWRITE操作，再运行一次即可。
- 对于MaxCompute内置的TSV或CSV解析器，处理产生的文件数目与对应SQL的并发度相同。
- 如果 `insert overwrite ... select ... from ...;` 操作在源数据表from_tablename上分配了1000个Mapper，则最后将产生1000个TSV或CSV文件。
- 如果您需要控制生成文件的数目，可以通过MaxCompute的各种灵活语义和配置来实现。如果Outputer在Mapper里，可以通过调整 `odps.stage.mapper.split.size` 的大小来控制Mapper的并发数，从而调整产生的文件数目。如果Outputer在Reducer或Joiner里，也可以分别通过 `odps.stage.reducer.num` 和 `odps.stage.joiner.num` 来调整。

10.5. OTS外部表

本文将进一步为您介绍如何将来自Tablestore（OTS）的数据纳入MaxCompute上的计算生态，实现多种数据源之间的无缝连接。

表格存储（Tablestore）是构建在阿里云飞天分布式系统之上的NoSQL数据存储服务，提供海量结构化数据的存储和实时访问，详情请参见Tablestore文档。

您可以通过DataWorks配合MaxCompute对外部表进行可视化的创建、搜索、查询、配置、加工和分析。详情请参见外部表。

MaxCompute与Tablestore是两个独立的大数据计算和存储服务，所以两者之间的网络必须保证连通性。MaxCompute公共云服务访问Tablestore存储时，推荐您使用Tablestore私网地址，即Host名以*ots-internal.aliyuncs.com*作为结尾的地址，例如*tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.com*。

Tablestore与MaxCompute都有其自身的类型系统。在MaxCompute处理Tablestore数据时，两者之间的数据类型对应关系如下所示。

MaxCompute Type	Tablestore Type
STRING	STRING
BIGINT	INTEGER
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
BINARY	BINARY

STS模式授权

MaxCompute计算服务访问Tablestore数据需要有一个安全的授权通道。因此，MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）实现对数据的安全访问。

您可以通过以下两种方式授予权限：

- 当MaxCompute和Tablestore的Owner是同一个账号时，在登录阿里云账号后，需要完成一键授权。详情请参见一键授权。
- 自定义授权
  i. 首先在RAM控制台中授予MaxCompute访问Tablestore的权限。

     登录RAM控制台（若MaxCompute和Tablestore不是同一个账号，此处需由Tablestore账号登录进行授权），创建角色，例如角色AliyunODPSDefaultRole、AliyunODPSRoleForOtherUser。

  ii. 修改策略内容设置。

```
--当MaxCompute和Tablestore的Owner是同一个账号时，进行如下设置。
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
--当MaxCompute和Tablestore的Owner不是同一个账号时，进行如下设置。
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号的UID@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

您可以单击右上角的头像查看云账号的UID。



iii. 编辑该角色的授权策略AliyunODPSRolePolicy。

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "ots:ListTable",
   "ots:DescribeTable",
   "ots:GetRow",
   "ots:PutRow",
   "ots:UpdateRow",
   "ots:DeleteRow",
   "ots:GetRange",
   "ots:BatchGetRow",
   "ots:BatchWriteRow",
   "ots:ComputeSplitPointsBySize"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
--还可自定义其它权限。
```

iv. 将权限AliyunODPSRolePolicy授权给该角色。

创建外部表

MaxCompute通过创建外部表，把对Tablestore表数据的描述引入到MaxCompute的meta系统内部后，即可实现对Tablestore数据的处理。本节通过下述示例为您说明MaxCompute对接Tablestore的一些概念和实现。

建表时，表名和字段名大小写不敏感。创建外部表语句示例如下。

```
DROP TABLE IF EXISTS ots_table_external;
CREATE EXTERNAL TABLE IF NOT EXISTS ots_table_external
(
odps_orderkey bigint,
odps_orderdate string,
odps_custkey bigint,
odps_orderstatus string,
odps_totalprice double,
odps_createdate timestamp
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':o_orderkey,:o_orderdate,o_custkey,o_orderstatus,o_totalprice',
'tablestore.table.name'='ots_tpch_orders',
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole',
'tablestore.timestamp.ticks.unit'='seconds',
'tablestore.column.odps_createdate.timestamp.ticks.unit'='millis'
)
LOCATION 'tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.com';
```

示例说明：

- `com.aliyun.odps.TableStoreStorageHandler` 是MaxCompute内置的处理Tablestore数据的StorageHandler，定义了MaxCompute和Tablestore的交互，相关逻辑由MaxCompute实现。
- `SERDEPROPERITES` 是提供参数选项的接口，在使用TableStoreStorageHandler时，有三个必选项和和两个可选项。
  - 三个必选项为：
    - tablestore.columns.mapping：用于描述MaxCompute将访问的Tablestore表的列，包括主键和属性列。
      - 以冒号（：）开头用于表示Tablestore主键，例如示例中的 `:o_orderkey` 和 `:o_orderdate` ，其它的均为属性列。
      - Tablestore支持1~4个主键，主键类型为STRING、INTEGER和BINARY，其中第一个主键为分区键。
      - 在指定映射时，您必须提供指定Tablestore表的所有主键，无需提供全部的属性列。只需提供需要通过MaxCompute访问的属性列，提供的属性列必须是Tablestore表的列，否则即使外部表可以创建成功，查询时也会报错。
    - tablestore.table.name：需要访问的Tablestore表名。如果指定的Tablestore表名错误（不存在），则会报错，MaxCompute不会主动创建Tablestore表。
    - odps.properties.rolearn：RAM中AliyunODPSDefaultRole的ARN信息。您可以通过RAM控制台中的**RAM角色管理**进行获取。
  - 两个可选项为：
    - tablestore.timestamp.ticks.unit：表级别时间类型设置。用以指定该外部表中所有Integer的字段都处于同一个时间类型。选项值为seconds（秒）、millis（毫秒）、micros（微秒）、nanos（纳秒）。
    - tablestore.column.<col1_name>.timestamp.ticks.unit：列级别时间类型设置。用以指定该外部表中列字段的时间类型，选项值为seconds（秒）、millis（毫秒）、micros（微秒）、nanos（纳秒）。

    > **说明** 这两个选项主要是将Tablestore中字段类型为Integer的字段映射为MaxCompute中的timestamp类型。两个选项同时存在时，列级别优先级比表级别的高。

- LOCATION：用来指定Tablestore的Instance名、Endpoint等具体信息。这里的Tablestore数据的安全访问建立在前文介绍的RAM/STS授权的前提上。

您可以执行如下语句，查看创建好的外部表结构信息。

```
desc extended <table_name>;
```

在返回的信息里，除了包含和内部表一样的基础信息，Extended Info还包含外部表StorageHandler、Location等信息。

查询外部表

创建完外部表后，Tablestore的数据便引入到了MaxCompute生态中，即可通过正常的MaxCompute SQL语法访问Tablestore数据，如下所示。

```
SELECT odps_orderkey, odps_orderdate, SUM(odps_totalprice) AS sum_total
FROM ots_table_external
WHERE odps_orderkey > 5000 AND odps_orderkey < 7000 AND odps_orderdate >= '1996-05-03' AND odps_orderdate < '1997-05-01'
GROUP BY odps_orderkey, odps_orderdate
HAVING sum_total> 400000.0;
```

> **说明** 在查询表或字段时，无需区分大小写，且不支持强制转换大小写。

使用常见的MaxCompute SQL语句访问Tablestore时，所有的操作细节（例如列名的选择）是在MaxCompute内部处理完成的。上述SQL示例中，使用的列名是odps_orderkey、odps_totalprice等，而不是原始Tablestore中的主键名o_orderkey或属性列名o_totalprice。这是因为在创建外部表的DDL语句中，已经完成了对应的Mapping。您也可以在创建外部表时，按需选择保留原始的Tablestore主键/列名。

如果您需要对一份数据做多次计算，相比每次从Tablestore去远程读数据，更高效的方法是先一次性把需要的数据导入到MaxCompute内部成为一个MaxCompute（内部）表，示例如下。

```
CREATE TABLE internal_orders AS
SELECT odps_orderkey, odps_orderdate, odps_custkey, odps_totalprice
FROM ots_table_external
WHERE odps_orderkey > 5000 ;
```

现在internal_orders就是一个MaxCompute表了，也拥有所有MaxCompute内部表的特性，包括高效的压缩列存储数据格式、完整的内部宏数据以及统计信息等。同时因为表存储在MaxCompute内部，所以访问速度会比访问外部的Tablestore更快。这种方法非常适用于需要进行多次计算的热点数据。

MaxCompute导出数据到Tablestore

> **说明** MaxCompute不会主动创建外部的Tablestore表，所以在对Tablestore表进行数据输出之前，必须保证该表已经在Tablestore上完成创建（否则将报错）。

根据上面的操作，您已创建了外部表ots_table_external来打通MaxCompute与Tablestore数据表ots_tpch_orders的链路，同时还有一份存储在MaxCompute内部表internal_orders的数据。现在，如果您需要对internal_orders中的数据进行处理后再写回Tablestore，则可通过对外部表执行 `insert overwrite table` 操作实现，示例如下。

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custkey, 'SHIPPED'), CEIL(odps_totalprice)
FROM internal_orders;
```

> **说明** 如果MaxCompute表内数据本身有一定的顺序，例如已经按照Primary Key做过一次排序，则在写入到OTS表时，会导致压力集中在一个OTS分区上面，无法充分利用分布式写入的特点。因此，当出现这种情况时，建议您通过 `distribute by rand()` 先将数据打散，示例如下。
>
> ```
> INSERT OVERWRITE TABLE ots_table_external
> SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custkey, 'SHIPPED'), CEIL(odps_totalprice)
> FROM (SELECT * FROM internal_orders DISTRIBUTE BY rand()) t;
> ```

对于Tablestore这种KV数据的NoSQL存储介质，MaxCompute的输出将只影响相对应主键所在的行，例如示例中只影响所有odps_orderkey + odps_orderdate这两个主键值对应行上的数据。而且在这些Tablestore行上，也只会更新在创建外部表（ots_table_external）时指定的属性列，而不会修改未在外部表中出现的数据列。

> **说明**
>
> - 将MaxCompute中的数据写入OTS时一次不能超过4MB，否则需要用户剔除掉超大数据再写入。此时可能会产生报错。
>
>   ```
>   ODPS-0010000:System internal error - Output to TableStore failed with exception:
>   TableStore BatchWrite request id XXXXX failed with error code OTSParameterInvalid and message:The total data size of B
>   atchWriteRow request exceeds the limit
>   ```
>
> - 将数据批量写入或分行写入，都算一次操作。详细描述请参考BatchWriteRow。因此如果批量写入数据量太大，也可以分行写入。
> - 将数据批量写入时请注意不要有重复行，否则可能产生如下报错。
>
>   ```
>   ErrorCode: OTSParameterInvalid, ErrorMessage: The input parameter is invalid
>   ```
>
>   详细描述请参见使用BatchWriteRow一次提交100条数据的时候报OTSParameterInvalid错误。

更多详情请参见MaxCompute访问TableStore（OTS）数据。

10.6. Hologres外部表

MaxCompute外部表支持使用数据库JDBC驱动机制访问Hologres数据源数据。本文为您介绍如何在外部表建表语句中指定Hologres数据源、STS认证信息、映射目标表、JDBC驱动信息，来创建Hologres的外部表。

背景信息

Hologres是兼容PostgreSQL协议的实时交互式分析数据仓库，在底层与MaxCompute无缝连接。

您可以使用在MaxCompute上创建Hologres外部表的方式，基于PostgreSQL JDBC驱动及STS认证信息查询Hologres数据源的数据。该方式无冗余存储，无需导入导出数据，可实现快速获取查询结果。

前提条件

创建Hologres外部表前，请确认已经满足如下条件：

- 已准备好Hologres数据库及目标表。

  创建Hologres数据库信息，请参见创建数据库。

  创建Hologres表信息，请参见CREATE TABLE。

  假设已准备好的Hologres实例信息如下：

  - Hologres数据库名称：`mc_test`。
  - Hologres数据库的Schema：`public`。
  - Hologres数据库经典网络连接地址：`hgprecn-cn-oew210ut****-cn-hangzhou-internal.hologres.aliyuncs.com:80`。
  - Hologres表名称：`holo`。表数据如下：

A	B
id	name
1	kate
2	mary
3	bob
4	tom
5	lulu
6	mark
7	haward
8	lilei
9	hanmeimei
10	lily
11	lucy

- 已准备好待创建Hologres外部表的目标MaxCompute项目。

  创建MaxCompute项目信息，请参见创建MaxCompute项目。

- 已安装MaxCompute客户端。

  更多安装MaxCompute客户端操作，请参见安装并配置MaxCompute客户端。

使用限制

Hologres外部表的使用限制如下：

- MaxCompute不支持对创建的Hologres外部表执行更新（UPDATE）、删除（DELETE）操作。
- Hologres的分区表和MaxCompute的分区表没有对应关系。Hologres外部表不支持分区。
- 当您需要向Hologres外部表写入大量数据时，采用并行多进程写入方式，会小概率出现某个写入进程数据重写情况，导致数据重复。
- MaxCompute里建的Hologres外部表，其中DECIMAL数据类型默认小数18位，不能修改，只能建成`decimal(38, 18)`。如果小数位数比较少，可以在MaxCompute中建外部表时数据类型选择String，使用的时候再使用`cast`函数强制转化后使用。
- MaxCompute创建的Hologres外部表，不支持Array，Map，Struct复杂数据类型。
- 在MaxCompute创建Hologres外部表时，Hologres中有的JSON、JSONB、MONEY等数据类型，目前在MaxCompute没有对应的数据类型，暂时不支持。

注意事项

Hologres外部表使用过程中需要注意：

- 对于Hologres的父、子表，Hologres外部表中会指定表名，并执行SQL语句。父、子表都可以映射至Hologres外部表，但是父表只能读不能写。
- 如果在执行查询Hologres外部表或向Hologres外部表插入数据时，报错`FAILED: Generating job conf failed, gen jobconf failed: External table location scheme "jdbc:postgresql" is not supported`，原因是未打开对SQL执行计划优化升级的功能`jobconf2`。大部分项目已默认打开此功能，部分项目还未打开此功能。如果需要打开此功能，请在Session级别增加如下4个属性。

  ```
  set odps.sql.jobconf.odps2=true;
  set odps.sql.jobconf.odps2.enforce=true;
  set odps.sql.split.hive.bridge=true;
  set odps.sql.hive.compatible=true;
  ```

- 向Hologres外部表写入数据时，暂不支持Hologres的INSERT ON CONFLICT(UPSERT)机制。如果Hologres源表有主键，请避免写入的数据与Hologres源表中的数据产生主键唯一性冲突。
- 建表时，表名和字段名大小写不敏感。在查询表或字段时，无需区分大小写，且不支持强制转换大小写。

创建Hologres外部表语法

在创建外部表时，您需要在建表DDL语句中指定StorageHandler，并配置STS认证信息、JDBC连接地址实现访问Hologres数据源。建表语句定义如下。

```
create external table [if not exists] <table_name>(
  <col1_name> <data_type>,
  <col2_name> <data_type>,
  ......
)
stored by '<com.aliyun.odps.jdbc.JdbcStorageHandler>'
with serdeproperties (
  'odps.properties.rolearn'='<ram_arn>')
location '<jdbc:postgresql://<endpoint>:<port>/<database>?ApplicationName=MaxCompute&[currentSchema=<schema>&][useSSL={true|false}&]table=<holo_table_name>/>'
tblproperties (
  'mcfed.mapreduce.jdbc.driver.class'='org.postgresql.Driver',
  'odps.federation.jdbc.target.db.type'='holo',
  ['odps.federation.jdbc.colmapping'='<col1:column1,col2:column2,col3:column3,...>']
);
```

- if not exists：可选。如果不指定if not exists选项而存在同名表，会报错。如果指定if not exists，无论是否存在同名表，即使原表结构与要创建的目标表结构不一致，均返回成功。已存在的同名表的元数据信息不会被改动。
- table_name：必填。在MaxCompute上创建的Hologres外部表的名称。
- col_name：必填。Hologres外部表的列名称。
- data_type：必填。Hologres外部表的列的数据类型。
- stored by：必填。指定StorageHandler，定义了如何查询Hologres外部表。固定取值为 `com.aliyun.odps.jdbc.JdbcStorageHandler` ，使用JdbcStorageHandler连接方式。
- ram_arn：必填。指定RAM角色的ARN信息，用于创建外部表时STS认证信息的填写。您可以在RAM访问控制页面，单击目标RAM角色名称后，在**基本信息**区域获取。





- location：必填。Hologres实例的JDBC连接地址。其中：
  - endpoint：必填。Hologres实例的**经典网络域名**。获取方式，请参见实例配置。

- port：必填。Hologres实例的网络端口。获取方式，请参见实例配置。



- database：必填。连接的Hologres数据库名称。更多Hologres数据库信息，请参见CREATE DATABASE。
- ApplicationName：必填。默认为MaxCompute，无需修改。
- schema：可选。如果表名在Hologres数据库内是唯一的，或源表是默认Schema中的表，可以不配置该属性。更多Schema信息，请参见CREATE SCHEMA。
- holo_table_name：必填。Hologres源表名称。更多Hologres源表信息，请参见CREATE TABLE。



- tblproperties：
  - mcfed.mapreduce.jdbc.driver.class：必填。指定连接Hologres数据库的驱动程序。固定取值为 `org.postgresql.Driver` 。
  - odps.federation.jdbc.target.db.type：必填。指定连接的数据库类型。固定取值为 `holo` 。
  - odps.federation.jdbc.colmapping：可选。如果需要将指定数据源的部分列映射至Hologres外部表，需要配置该参数，指定Hologres源表的字段和MaxCompute外部表字段的映射关系。如果不配置该参数，按照源表字段顺序映射至Hologres外部表。格式为 `Hologres外部表列名1:Hologres源表列名1,Hologres外部表列名2:Hologres源表列名2,...` 。

操作流程

创建Hologres外部表的流程如下：

1. 步骤一：创建RAM角色

   创建RAM角色获取ARN信息，用于创建外部表时填写STS认证信息。

2. 步骤二：添加RAM角色至Hologres实例并授权

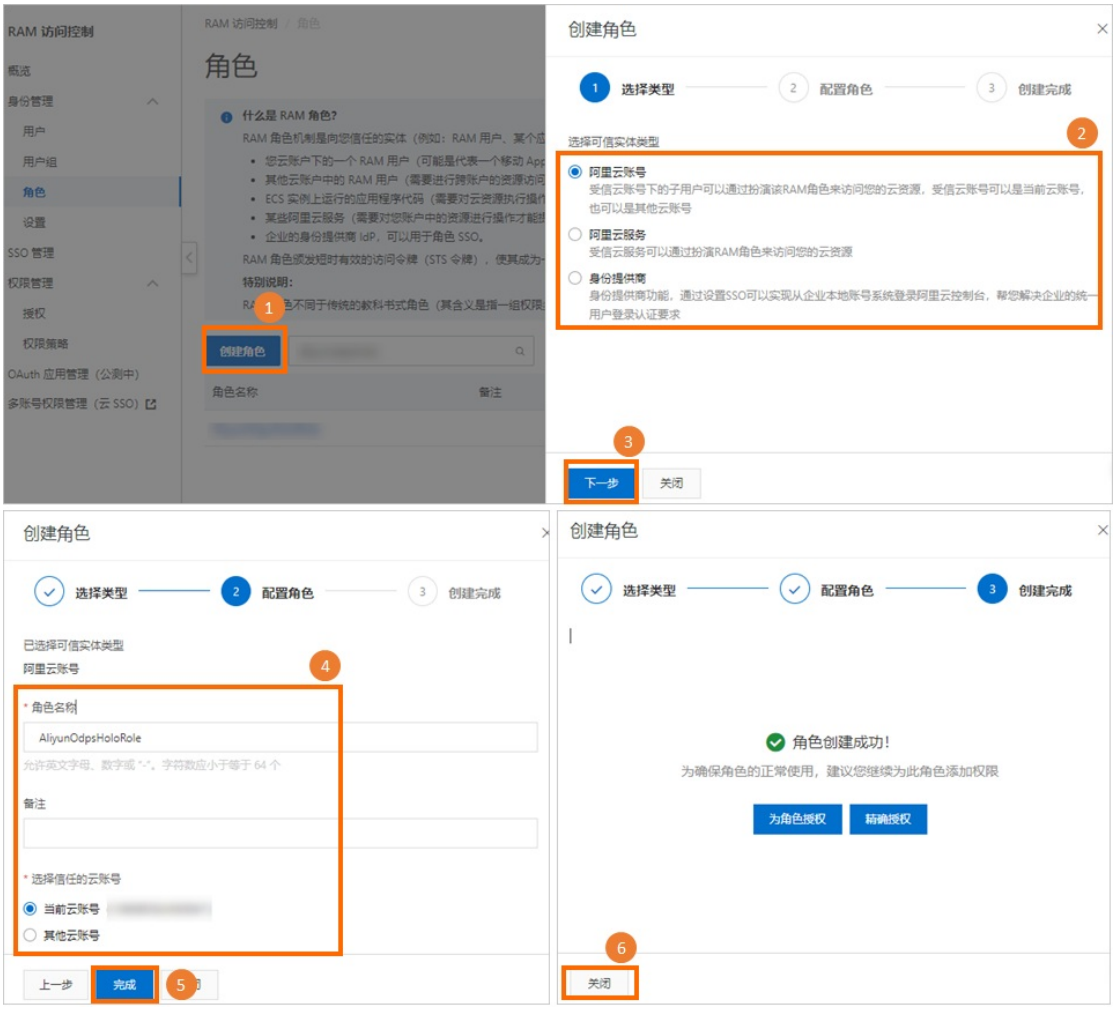
   将新建的RAM角色添加至Hologres实例中，授予RAM角色访问Hologres实例的权限。

3. 步骤三：创建Hologres外部表

   创建Hologres外部表，映射Hologres源表。

步骤一：创建RAM角色

1. 登录RAM访问控制创建RAM角色。



创建RAM角色的可信实体类型根据实际需求选择阿里云账号或者身份提供商。

- 阿里云账号：

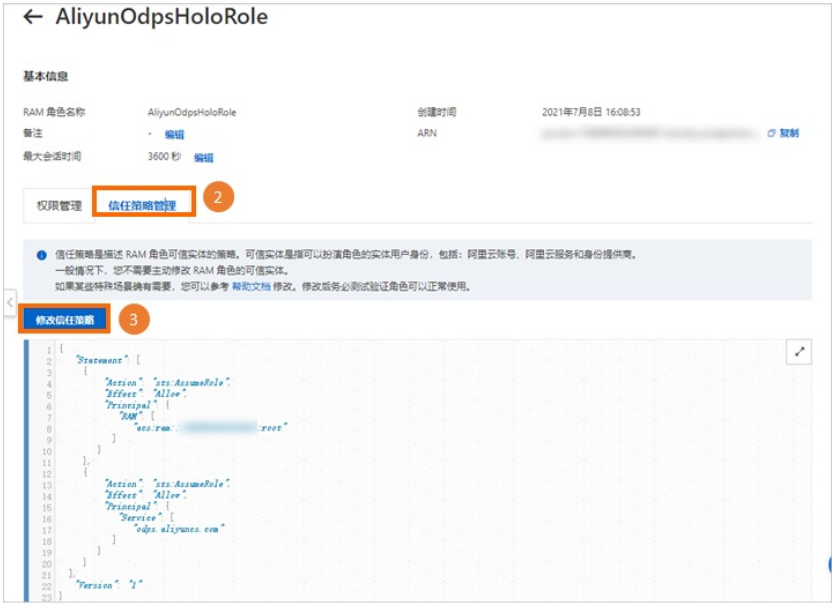
  阿里云账号下的RAM用户可以通过扮演RAM角色来访问云资源，详情请参见创建可信实体为阿里云账号的RAM角色。

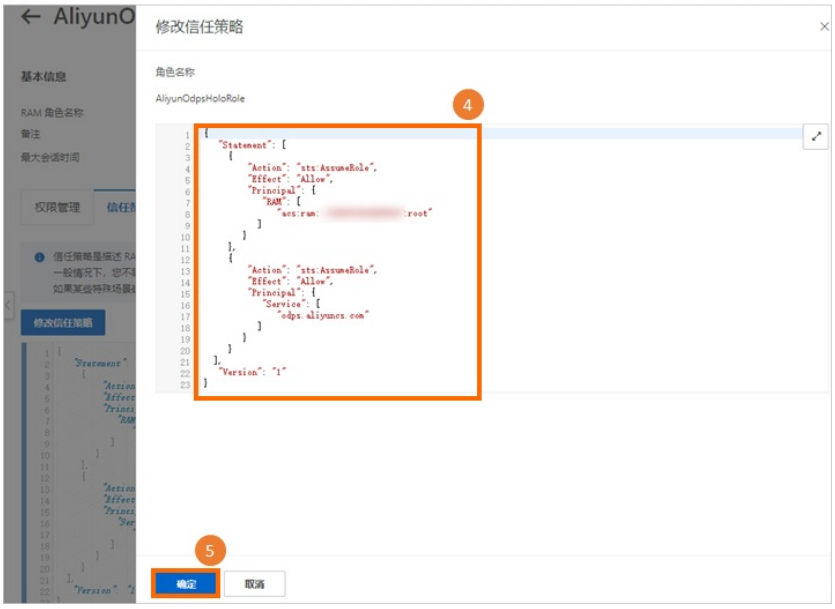
- 身份提供商：

  通过设置SSO实现从企业本地账号系统登录至阿里云控制台，帮您解决企业的统一用户登录认证要求。详情请参见创建可信实体为身份提供商的RAM角色。

2. 修改信任策略配置内容。







i. 在**角色**页面，单击已创建完成的RAM角色名称。

ii. 单击**信任策略管理**页签。

iii. 在**信任策略管理**页签，单击**修改信任策略**。

iv. 在**修改信任策略**面板，参照如下内容修改信任策略配置。

修改信任策略配置内容与选择的可信实体类型相关。

- 可信实体类型为阿里云账号：

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "RAM": [
          "acs:ram::<阿里云账号（主账号）ID>:root"
        ]
      }
    },
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

- 可信实体类型为身份提供商：

```
"Statement": [
        {
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "saml:recipient": "https://signin.aliyun.com/saml-role/sso"
                }
            },
            "Effect": "Allow",
            "Principal": {
                "Federated": [
                    "acs:ram::<阿里云账号（主账号）ID>:saml-provider/IDP"
                ]
            }
        },
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "odps.aliyuncs.com"
                ]
            }
        }
    ],
    "Version": "1"
}
}
```

> **说明** 阿里云账号ID可在用户信息页面获取。

v. 单击**确定**。

步骤二：添加RAM角色至Hologres实例并授权

RAM角色需要有Hologres实例的开发权限，才能在权限范围内使用Hologres。由于RAM角色默认没有Hologres管理控制台的查看和操作实例的权限，因此需要阿里云账号完成RAM相关权限授予才能进行后续操作。添加RAM角色至Hologres实例，您可以通过如下方式进行授权。

- 通过Hologres管理控制台授权。
  i. 登录Hologres管理控制台。
  ii. 在左侧导航栏单击**实例列表**，单击需要授权的Hologres实例名称。
  iii. 在实例详情页面，单击**账号管理**。

iv. 在**用户管理**页面单击**新增用户**添加RAM角色至Hologres实例。



v. 在**DB授权**页签，为该RAM角色授予实例的开发权限。

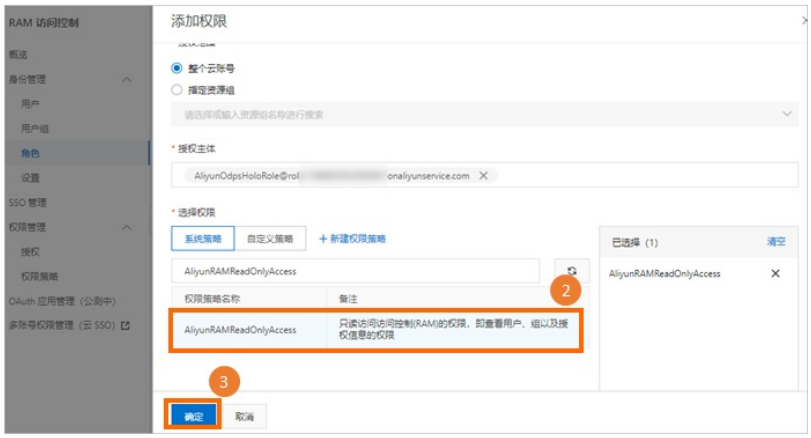


- 通过SQL方式授权。

  您可以通过SQL方式进行授权，授权SQL请参见Hologres权限模型概述。

- 若是通过RAM用户扮演RAM角色，RAM用户默认没有Hologres管理控制台的权限，需要阿里云账号给RAM用户在访问控制页面授予**AliyunRAMReadOnlyAccess**权限，否则RAM用户无法在Hologres管理控制台进行任何操作。详情请参见文档授予RAM用户权限。





步骤三：创建Hologres外部表

完成上述步骤后，基于已准备好的数据信息，您即可登录MaxCompute客户端，结合创建Hologres外部表语法创建Hologres外部表。

1. 登录MaxCompute客户端，进入目标MaxCompute项目。

   进入目标项目命令信息，请参见项目空间操作。

2. 执行如下命令创建Hologres外部表。

   命令示例如下。

```
create external table if not exists my_table_holo_jdbc
(
 id bigint,
 name string
)
stored by 'com.aliyun.odps.jdbc.JdbcStorageHandler'
with serdeproperties (
  'odps.properties.rolearn'='acs:ram::139699392458****:role/aliyunodpsholorole')
location 'jdbc:postgresql://hgprecn-cn-oew210ut****-cn-hangzhou-internal.hologres.aliyuncs.com:80/mc_test?ApplicationName=MaxCompute&currentSchema=public&useSSL=false&table=holo/'
tblproperties (
  'mcfed.mapreduce.jdbc.driver.class'='org.postgresql.Driver',
  'odps.federation.jdbc.target.db.type'='holo',
  'odps.federation.jdbc.colmapping'='id:id,name:name'
);
```

3. 执行如下命令基于新建的Hologres外部表查询Hologres源表信息。

   命令示例如下。

```
--访问Hologres外部表需要添加如下属性。
set odps.sql.split.hive.bridge=true;
set odps.sql.hive.compatible=true;
--查询Hologres外部表数据。
select * from my_table_holo_jdbc limit 10;
```

   返回结果如下。

```
+------------+------------+
| id         | name       |
+------------+------------+
| 1          | kate       |
| 2          | mary       |
| 3          | bob        |
| 4          | tom        |
| 5          | lulu       |
| 6          | mark       |
| 7          | haward     |
| 8          | lilei      |
| 9          | hanmeimei  |
| 10         | lily       |
+------------+------------+
```

4. （可选）基于Hologres外部表与Hologres进行数据交换、联合分析。

   例如，将MaxCompute加工后的数据利用Hologres外部表，写入Hologres实现加速分析、在线服务。命令示例如下。

```
--访问Hologres外部表需要添加如下属性。
set odps.sql.split.hive.bridge=true;
set odps.sql.hive.compatible=true;
--向Hologres外部表插入数据。
insert into my_table_holo_jdbc values (12,'alice');
--查询Hologres外部表数据。
select * from my_table_holo_jdbc;
```

   返回结果如下。

```
+------------+------------+
| id         | name       |
+------------+------------+
| 12          | alice      |
| 1          | kate       |
| 2          | mary       |
| 3          | bob        |
| 4          | tom        |
| 5          | lulu       |
| 6          | mark       |
| 7          | haward     |
| 8          | lilei      |
| 9          | hanmeimei  |
| 10         | lily       |
| 11         | lucy       |
+------------+------------+
```

   频繁更新的维度表保存在Hologres，满足实时动态更新需要。MaxCompute通过外部表方式访问维度表与MaxCompute中事实表进行关联分析，命令示例如下。

```
--访问Hologres外部表需要添加如下属性。
set odps.sql.split.hive.bridge=true;
set odps.sql.hive.compatible=true;
--创建MaxCompute内部表。
create table holo_test as select * from my_table_holo_jdbc;
--MaxCompute内部表与Hologres外部表进行关联分析。
select * from my_table_holo_jdbc t1 inner join holo_test t2 on t1.id=t2.id;
```

   返回结果如下。

```
+------------+------------+------------+------------+
| id         | name       | id2        | name2      |
+------------+------------+------------+------------+
| 1          | kate       | 1          | kate       |
| 2          | mary       | 2          | mary       |
| 3          | bob        | 3          | bob        |
| 4          | tom        | 4          | tom        |
| 5          | lulu       | 5          | lulu       |
| 6          | mark       | 6          | mark       |
| 7          | harward    | 7          | harward    |
| 8          | lilei      | 8          | lilei      |
| 9          | hanmeimei  | 9          | hanmeimei  |
| 10         | lily       | 10         | lily       |
| 11         | lucy       | 11         | lucy       |
| 12         | alice      | 12         | alice      |
+------------+------------+------------+------------+
```

10.7. RDS外部表

RDS（Relational Database Service）是阿里云用户主要使用的服务，需要通过内网域名访问。如果您需要通过MaxCompute将数据加载至RDS的表中，可参考该文档进行操作。本文为您介绍如何在外网或VPC网络环境下基于RDS数据源创建外部表并写入数据。

前提条件

已开通MaxCompute和云数据库RDS间的网络连接，详情请参见网络开通流程。

使用限制

- 只有华北2（北京）、华东2（上海）、华北3（张家口）、华东1（杭州）和华南1（深圳）地域支持通过服务映射或专有网络连接方案连接MaxCompute和云数据库RDS，仅这五个地域可以创建RDS数据源外部表，其他地域暂不支持。
- 暂不支持PrivateZone域名。
- 支持RDS MySQL5.x版本，暂不支持8.0版本。
- 当您需要向RDS外部表写入大量数据时，采用并行多进程写入方式，会小概率出现某个写入进程数据重写情况，导致数据重复。
- MaxCompute里建的RDS外部表，其中DECIMAL数据类型默认小数18位，不能修改，只能建成 `decimal(38, 18)` 。如果小数位数比较少，可以在MaxCompute中建外部表时数据类型选择String，使用的时候再使用 `cast` 函数强制转换。

语法说明

建表时，表名和列名大小写不敏感。在查询表或列名时，无需区分大小写，且不支持强制转换大小写。

```
--开启Hive兼容模式。
set odps.sql.hive.compatible = true;
--打开外部表VPC支持。
set odps.sql.external.net.vpc=true;
CREATE EXTERNAL TABLE <mcrdsexternal_table_name>(
  <col_name1> <data_type>,
  <col_name2> <data_type>,
  ......
)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'  --处理JDBC连接类数据源的Handler。
 WITH SERDEPROPERTIES (
 'odps.external.net.vpc'='true',
 'odps.vpc.id'='<VPC ID>',
 'odps.vpc.access.ips'='<realm_name:port>',
 'odps.vpc.region'='<RegionID>'
 )
location '<jdbc:mysql://realm_name:port/rds_database_name?useSSL=false&user=user_name&password=password_value&table=rds_table_nam
e>'
TBLPROPERTIES(
 ['odps.federation.jdbc.colmapping'='<col_name1:rdstable_colname1|select_alias1>[,<col_name2:rdstable_colname2|select_alias2>,...
]',]
 'mcfed.mapreduce.jdbc.input.query'='<select_sentence>'
)
;
```

- mcrdsexternal_table_name：必填。待创建外部表的名称。
- col_name：必填。外部表的列名称。
- data_type：必填。列的数据类型。
- WITH SERDEPROPERTIES：
  - 'odps.external.net.vpc'='true'：必填。表明外部表的数据源处于VPC网络。
  - 'odps.vpc.id'='<VPC ID>'：必填。VPC网络ID号。详情请参见获取RegionID及VPC ID。
  - 'odps.vpc.access.ips'='<realm_name:port>'：必填。RDS数据源外部表所属VPC域名（内网地址）及端口。查看内网地址及端口操作，请参见查看或修改内外网地址和端口。
  - 'odps.vpc.region'='<RegionID>'：必填。VPC网络所在的地域ID。详情请参见获取RegionID及VPC ID。
- jdbc:mysql://realm_name:port/rds_database_name?useSSL=false&user=user_name&password=password_value@&table=rds_table_name：必填。连接RDS数据源表的连接字符串。
  - rds_database_name：RDS数据库名称。
  - user_name：RDS数据库的账号。
  - password_value：RDS数据库的密码。
  - rds_table_name：RDS源表名称。
- TBLPROPERTIES：
  - 'odps.federation.jdbc.colmapping'='<col_name1:rdstable_colname1|select_alias1>[,<col_name2:rdstable_colname2|select_alias2>,...]'：可选。MaxCompute外部表与RDS数据源表列的映射关系，此处的映射列数，需要与MaxCompute外部表定义的列数保持一致。其中：rdstable_colname为RDS源表的列名（全部列映射），select_alias为给查询结果定义的列别名（指定列映射）。

- 'mcfed.mapreduce.jdbc.input.query'='<select_sentence>'：必填。读取RDS数据源表数据。新建外部表的列名与RDS数据源表的列名及数据类型保持一致。`select_sentence` 格式为 `select xx from rds_database_name.rds_table_name` 。

使用RDS数据源创建MaxCompute的外部表并加载数据

使用RDS数据源创建MaxCompute外部表的步骤如下：

1. 登录RDS数据库，执行建表语句并插入数据。操作详情请参见通过DMS登录RDS数据库。

   建表示例如下：

   ```
   CREATE TABLE `rds_mc_external` (
     `id` int(11) DEFAULT NULL,
     `name` varchar(32) DEFAULT NULL
   ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
   INSERT INTO `rds_mc_external`(`id` ,`name` ) VALUES(1,"张三");
   INSERT INTO `rds_mc_external`(`id` ,`name` ) VALUES(1,"zhangsan");
   ```

2. 登录MaxCompute客户端，设置MaxCompute需要访问的外部表所属RegionID和VPC ID。

   命令语法如下：

   ```
   --注册RDS数据源的VPC属性。
   setproject odps.security.outbound.destination=<RegionID>_<VPC ID>[*];
   ```

   - RegionID：必填。VPC网络所在的地域ID。详情请参见获取RegionID及VPC ID。
   - VPC ID：必填。VPC网络ID号。详情请参见获取RegionID及VPC ID。
   - [*]：可选。通配符，表示VPC网络下的所有IP和端口都加入MaxCompute白名单。

   命令示例如下：

   ```
   setproject odps.security.outbound.destination=cn-beijing_vpc-2ze7cqx2bqodp9ri1****[*];
   ```

3. 在MaxCompute客户端创建映射RDS数据源的外部表。您可以通过如下两种方式进行操作：
   - 创建MaxCompute外部表的列名和RDS中表的列名完全对应。
     a. 在MaxCompute客户端创建外部表，表列名与RDS中表的列名完全对应。命令示例如下：

     ```
     set odps.sql.hive.compatible = true;
     set odps.sql.external.net.vpc=true;
     CREATE EXTERNAL TABLE mc_vpc_rds_external (
     id int,
     name string
     )
     STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
     WITH SERDEPROPERTIES (
     'odps.external.net.vpc'='true',
     'odps.vpc.id'='vpc-2zeeopx9atnrkay8n****',
     'odps.vpc.access.ips'='rm-2ze01y92y1tzp****.mysql.rds.aliyuncs.com:3306',
     'odps.vpc.region'='cn-beijing'
     )
     location 'jdbc:mysql://rm-2ze01y92y1tzp****.mysql.rds.aliyuncs.com:3306/rds_test?useSSL=false&user=****&password=****&
     table=rds_mc_external'
     TBLPROPERTIES(
     'mcfed.mapreduce.jdbc.input.query'='select * from rds_test.rds_mc_external'
     );
     ```

     b. 向新建的MaxCompute表中插入数据。

     命令示例如下：

     ```
     --插入数据。
     INSERT INTO table mc_vpc_rds_external values(2,"zhagnsan");
     ```

     c. 查询数据插入结果。

     命令示例如下：

     ```
     --查询数据插入结果。
     SELECT * FROM mc_vpc_rds_external;
     ```

   - 创建MaxCompute外部表的列名和RDS表指定的列名进行映射。

a. 在MaxCompute客户端创建外部表，表列名与RDS表中指定的列名进行映射。命令示例如下：

```
set odps.sql.hive.compatible = true;
set odps.sql.external.net.vpc=true;
CREATE EXTERNAL TABLE mc_vpc_rds_external_mapping (
  key int,
  value string
)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
 WITH SERDEPROPERTIES (
 'odps.external.net.vpc'='true',
 'odps.vpc.id'='vpc-2zeeopx9atnrkay8n****',
 'odps.vpc.access.ips'='rm-2ze01y92y1tzp****.mysql.rds.aliyuncs.com:3306',
 'odps.vpc.region'='cn-beijing'
 )
location 'jdbc:mysql://rm-2ze01y92y1tzp****.mysql.rds.aliyuncs.com:3306/rds_test?useSSL=false&user=****&password=****&
table=rds_mc_external'
TBLPROPERTIES(
 'odps.federation.jdbc.colmapping'='key:id2,value:id3',
 'mcfed.mapreduce.jdbc.input.query'='select id*100 as id2,count(*) as id3 from test_mc_rdset.rds_mc_external group by
id'
)
;
```

b. 向新建的MaxCompute表中插入数据。

命令示例如下：

```
--插入数据。
INSERT INTO TABLE mc_vpc_rds_external_mapping VALUES(4,"lisi");
```

c. 查询数据插入结果。

命令示例如下：

```
--查询数据插入结果。
SELECT * FROM mc_vpc_rds_external_mapping;
```

10.8. HBase外部表（HBase标准版或增强版）

在MaxCompute与阿里云HBase服务间的网络处于连通状态时，MaxCompute支持您通过创建HBase外部表来关联HBase标准版或增强版集群中的表。使用HBase外部表，可以将HBase中的数据同步到MaxCompute进行后续ETL处理，也可以关联HBase外部表进行联邦计算，或将MaxCompute数据输出到HBase的表中。本文为您介绍如何创建HBase外部表。

背景信息

HBase是兼容标准HBase访问协议的分布式数据库，包含HBase标准版、HBase增强版和HBase Serverless版，MaxCompute仅支持访问HBase标准版或HBase增强版集群。

默认情况下，MaxCompute与阿里云HBase服务间的网络不可达，您需要提前通过专有网络连接方案（VPC）建立二者间的网络通路后，才可以创建HBase外部表来关联HBase标准版或增强版集群中的表。

> **说明** HBase增强版是基于云原生多模数据库Lindorm内核研发的，HBase增强版实例也可以通过Lindorm控制台创建，该场景下创建HBase外部表的操作，请参见Lindorm外部表（云原生多模数据库）。本文仅以通过HBase控制台创建的HBase实例为例进行介绍。

前提条件

创建HBase外部表前，请确认已经满足如下条件：

- 已创建HBase标准版或增强版集群，并完成环境配置。

  更多购买HBase标准版集群及配置环境信息，请参见HBase标准版快速入门。

  更多购买HBase增强版集群及配置环境信息，请参见HBase增强版快速入门。

- 已通过HBase客户端在HBase集群中创建好了表。

  更多创建HBase表信息，请参见使用HBase Shell。

  为便于展示使用示例，本文假设已在HBase标准版或增强版集群中创建了表mf_hbase_test_t，建表命令如下。

  ```
  --创建HBase表。
  create 'mf_hbase_test_t', 'f'
  --插入数据。其中：1表示RowKey，col_family:col_name表示列，f表示col_family，col1表示col_name，hello表示col_value。
  put 'mf_hbase_test_t', '1', 'f:col1', 'hello'
  ```

- 已创建待关联HBase表的目标MaxCompute项目。

  更多创建MaxCompute项目信息，请参见创建MaxCompute项目。

- 已安装MaxCompute客户端。

  更多安装MaxCompute客户端操作，请参见安装并配置MaxCompute客户端。
- 已基于专有网络连接方案，开通MaxCompute和HBase服务间的网络连接。

  更多专有网络连接方案信息，请参见专有网络连接方案。

使用限制

- 由于只有华北2（北京）、华东2（上海）、华北3（张家口）、华东1（杭州）和华南1（深圳）五个地域开通了专有网络连接方案，只有以上五个地域可以创建HBase外部表，其他地域暂不支持。
- HBase外部表仅支持访问HBase2.x版本的集群。
- HBase外部表暂不支持EMR HBase集群。

创建HBase外部表语法

在创建HBase外部表时，您需要在建表DDL语句中指定HBaseStorageHandler，并配置HBase标准版或增强版集群相关信息。建表时，表名和字段名大小写不敏感。在查询表或字段时，无需区分大小写，且不支持强制转换大小写。建表语句定义如下。

```
--打开外部表VPC支持。
set odps.sql.external.net.vpc=true;
--打开Hive兼容模式。
set odps.sql.hive.compatible = true;
--仅当HBase集群为增强版时，除上述2个SET参数外，还必须配置odps.sql.hbase.version参数，切换HBase版本为lindorm。
set odps.sql.hbase.version=lindorm;
--创建HBase外部表。
create external table if not exists <mc_hbase_external>
(
  <col1_name> <data_type>,
  <col2_name> <data_type>,
  ......
)
--处理HBase数据源的Handler。
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties (
                      'hbase.columns.mapping'='<:key,col-family:col-name1,col-family:col-name2,...>',
                      'odps.external.net.vpc'='<true>',
                      'odps.vpc.id'='<VPC ID>',
                      'odps.vpc.access.ips'='<VPC ip|host:port,VPC ipBegin-ipEnd:port|network_segment:port>',
                      'odps.vpc.region'='<RegionID>')
tblproperties (
--hbase.table.name和hbase.zookeeper.quorum必须配置。
'hbase.table.name'='<hbase_table_name>',
'hbase.zookeeper.quorum'='<VPC ip1|host1:port,VPC ip2|host2:port...>',
--仅当HBase集群为增强版时，还必须配置hbase.client.username和hbase.client.password参数。
'hbase.client.username'='****',
'hbase.client.password'='****',
--hbase.zookeeper.property.clientPort、mcfed.zookeeper.session.timeout和hbase.client.retries.number为可选参数。
--当HBase集群为标准版时，hbase.zookeeper.property.clientPort的默认值为2181。
--当HBase集群为增强版时，hbase.zookeeper.property.clientPort的默认值为30020。
['hbase.zookeeper.property.clientPort'='{2181|30020}',]
['mcfed.zookeeper.session.timeout'='<value>',]
['hbase.client.retries.number'='<value>']);
```

- mc_hbase_external：必填。待创建HBase外部表的名称。
- col_name：必填。HBase外部表的列名称。
- data_type：必填。列的数据类型。
- with serdeproperties：
  - 'hbase.columns.mapping'='<:key,col-family:col-name1,col-family:col-name2,...>'：必填。HBase外部表与HBase集群中表的列映射关系。
    - RowKey（`:key`）必须放在最前面。
    - 列格式为`col-family:col-name1,col-family:col-name2`。一个作业中col-family只能有一个。
  - 'odps.external.net.vpc'='true'：必填。固定取值为`true`，表明HBase外部表的数据源处于VPC网络。
  - 'odps.vpc.id'='<VPC ID>'：VPC实例ID。您可以登录HBase管理控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可在**连接信息**处获取VPC ID。

- 'odps.vpc.access.ips'='<VPC ip|host:port,VPC ipBegin-ipEnd:port|network_segment:port>'：必填。Zookeeper连接地址（HBase标准版）或HBase集群的Java API访问地址（HBase增强版）。也可以为关联的HBase表所属VPC网络IP、IP网段、端口。例如 `192.0.2.1-192.0.2.10:9000,192.0.2.20:70050` 或 `192.0.2.20/16:2181` 。您可以登录HBase管理控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可在**连接信息**处获取。
  - 对于分布式的数据源，请确认需要访问的前端IP端口以及DataNode的IP端口。HBase外部表需要填写Zookeeper、HMaster、Region Server的IP和端口（2181/30020、16000、16020）。HBase标准版的默认Zookeeper端口为2181。HBase增强版的默认Zookeeper端口为30020。
  - 由于该字段支持IP地址段，且映射IP会耗费资源，默认一次映射的IP数不能超过2000个。
- 'odps.vpc.region'='<RegionID>'：VPC网络所属地域ID。获取方法请参见获取RegionID及VPC ID。

- hbase://VPC ip|host:port：必填。HBase上配置的Zookeeper Master。即Zookeeper连接地址（HBase标准版）或HBase集群的Java API访问地址（HBase增强版）。
- tblproperties：
  - 'hbase.table.name'='<hbase_table_name>'：必填。MaxCompute所关联的HBase集群中的表的名称。必须提前通过HBase客户端建好。
  - 'hbase.zookeeper.quorum'='<VPC ip1|host1:port,VPC ip2|host2:port...>'：必填。HBase的Zookeeper的IP或域名。即Zookeeper连接地址（HBase标准版）或HBase集群的Java API访问地址（HBase增强版）。
  - hbase.client.username：必填。仅当HBase集群为增强版时，需要配置该参数。您可以登录HBase管理控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可通过**连接信息**处的**默认用户名**获取。
  - hbase.client.password：必填。仅当HBase集群为增强版时，需要配置该参数。您可以登录HBase管理控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可通过**连接信息**处的**默认密码**获取。
  - 'hbase.zookeeper.property.clientPort'='{2181|30020}'：可选。HBase的Zookeeper端口。HBase标准版的默认Zookeeper端口为2181。HBase增强版的默认Zookeeper端口为30020。
  - 'mcfed.zookeeper.session.timeout'='<value>'：可选。Zookeeper的超时时间。单位为毫秒，建议设置为30000。
  - 'hbase.client.retries.number'='<value>'：可选。HBase客户端连接重试次数。

> **说明** mcfed.zookeeper和hbase开头的配置是Zookeeper的相关配置。

使用示例

1. 登录MaxCompute客户端，执行 `CREATE EXTERNAL TABLE` 语句创建HBase外部表。

   命令示例如下。

   - HBase标准版

```
set odps.sql.external.net.vpc=true;
set odps.sql.hive.compatible = true;
create external table if not exists mf_hbase_test_t_ext
(
  key  int,
  col1 string,
  col2 string
)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties (
                    'hbase.columns.mapping'=':key,f:col1,f:col2',
                    'odps.external.net.vpc'='true',
                    'odps.vpc.id'='vpc-2zee****',
                    'odps.vpc.access.ips'='hb-2zef****-master1-001.hbase.rds.aliyuncs.com:2181,hb-2zef****-master2-001.hbase.rds.aliyuncs.com:2181,hb-2zef****-master3-001.hbase.rds.aliyuncs.com:2181,hb-2zef****-master1-001.hbase.rds.aliyuncs.com:16000,hb-2zef****-master2-001.hbase.rds.aliyuncs.com:16000,hb-2zef****-master3-001.hbase.rds.aliyuncs.com:16000,hb-2zef****-master1-001.hbase.rds.aliyuncs.com:16020,hb-2zef****-master2-001.hbase.rds.aliyuncs.com:16020,hb-2zef****-master3-001.hbase.rds.aliyuncs.com:16020',
                    'odps.vpc.region'='cn-beijing')
tblproperties (
'hbase.table.name'='mf_hbase_test_t',
'hbase.zookeeper.quorum'='hb-2zef****-master1-001.hbase.rds.aliyuncs.com,hb-2zef****-master2-001.hbase.rds.aliyuncs.com,hb-2zef****-master3-001.hbase.rds.aliyuncs.com',
'hbase.zookeeper.property.clientPort'='2181');
```

   - HBase增强版

```
set odps.sql.external.net.vpc=true;
set odps.sql.hive.compatible = true;
set odps.sql.hbase.version=lindorm;
CREATE EXTERNAL TABLE mf_hbase_test_t_ext
(
  key  int,
  col1 string,
  col2 string
)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties (
                      'hbase.columns.mapping':key,f:col1,f:col2',
                      'odps.external.net.vpc'='true',
                      'odps.vpc.id'='vpc-2zen****',
                      'odps.vpc.access.ips'='ld-2ze9****-proxy-hbaseue.hbaseue.rds.aliyuncs.com:30020',
                      'odps.vpc.region'='cn-beijing')
tblproperties (
'hbase.table.name'='mf_hbase_test_t',
'hbase.zookeeper.quorum'='ld-2ze9****-proxy-hbaseue.hbaseue.rds.aliyuncs.com:30020',
"hbase.client.username"="root",
"hbase.client.password"="root");
```

2. 执行 `select` 命令查看关联结果，确保关联成功。

   命令示例如下。

   ```
   select * from mf_hbase_test_t_ext;
   ```

   返回结果如下。

   ```
   +------+------------+------------+
   | key  | col1       | col2       |
   +------+------------+------------+
   | 1    | hello      | NULL       |
   +------+------------+------------+
   ```

3. 执行 `INSERT OVERWRITE` 语句向HBase中的表写入数据。

   命令示例如下。

   ```
   INSERT OVERWRITE TABLE mf_hbase_test_t_ext SELECT '2', 'hbase', 'mftestinsert';
   ```

4. 执行 `SELECT` 语句验证写入的数据。

   命令示例如下。

   ```
   select * from mf_hbase_test_t_ext;
   ```

   返回结果如下。

   ```
   +------+------------+--------------------+
   | key  | col1       | col2               |
   +------+------------+--------------------+
   | 1    | hello      | NULL               |
   | 2    | hbase      | mftestinsert       |
   +------+------------+--------------------+
   ```

10.9. Lindorm外部表（云原生多模数据库）

在MaxCompute与阿里云Lindorm服务间的网络处于连通状态时，MaxCompute支持您通过创建外部表来关联Lindorm表。使用Lindorm外部表，可以将Lindorm表中的数据同步到MaxCompute进行后续ETL处理，也可以关联Lindorm表进行联邦计算，或将MaxCompute数据输出到Lindorm表中。本文为您介绍如何通过MaxCompute创建Lindorm外部表并映射Lindorm数据源。

背景信息

Lindorm是面向物联网、互联网、车联网等设计和优化的云原生多模超融合数据库，支持宽表、时序、文本、对象、流、空间等多种数据的统一访问和融合处理，并兼容SQL、HBase/Cassandra/S3、TSDB、HDFS、Solr、Kafka等多种标准接口和无缝集成三方生态工具，是日志、监控、账单、广告、社交、出行、风控等场景首选数据库，也是为阿里巴巴核心业务提供支撑的数据库之一。更多Lindorm信息请参见Lindorm产品概述。

前提条件

创建Lindorm外部表前，请确认已经满足如下条件：

- 已创建Lindorm集群，并完成环境配置。详情请参见Lindorm快速入门。
- 已创建待关联Lindorm表的目标MaxCompute项目。

  更多创建MaxCompute项目信息，请参见创建MaxCompute项目。
- 已安装MaxCompute客户端。

更多安装MaxCompute客户端操作，请参见安装并配置MaxCompute客户端。

- 已基于专有网络连接方案，开通MaxCompute和Lindorm服务间的网络连接。

  更多专有网络连接方案信息，请参见专有网络连接方案。
- 已开通Lindorm服务并创建Lindorm实例。详情请参见创建实例。
- 已下载并配置Lindorm Shell。详情请参见通过Lindorm宽表SQL连接宽表引擎。

使用限制

由于只有华北2（北京）、华东2（上海）、华北3（张家口）、华东1（杭州）和华南1（深圳）五个地域开通了专有网络连接方案，只有以上五个地域可以创建Lindorm外部表，其他地域暂不支持。

创建Lindorm外部表语法

在创建Lindorm外部表时，您需要在建表DDL语句中指定HBaseStorageHandler，并配置Lindorm集群相关信息。建表时，表名和字段名大小写不敏感。在查询表或字段时，无需区分大小写，且不支持强制转换大小写。建表语句定义如下。

```
--打开外部表VPC支持。
set odps.sql.external.net.vpc=true;
--打开Hive兼容模式。
set odps.sql.hive.compatible = true;
--配置odps.sql.hbase.version参数，切换HBase版本为lindorm。
set odps.sql.hbase.version=lindorm;
--创建Lindorm外部表。
create external table if not exists <mc_lindorm_external>
(
  <col1_name> <data_type>,
  <col2_name> <data_type>,
  ......
)
--处理Lindorm数据源的Handler。
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties (
                    'hbase.columns.mapping'='<:key,col-family:col-name1,col-family:col-name2,...>',
                    'odps.external.net.vpc'='<true>',
                    'odps.vpc.id'='<VPC ID>',
                    'odps.vpc.access.ips'='<VPC ip|host:port,VPC ipBegin-ipEnd:port|network_segment:port>',
                    'odps.vpc.region'='<RegionID>')
tblproperties (
--hbase.table.name和mcfed.hbase.zookeeper.quorum必须配置。
'hbase.table.name'='<lindorm_table_name>',
'hbase.zookeeper.quorum'='<VPC ip1|host1:port,VPC ip2|host2:port...>',
--配置hbase.client.username和hbase.client.password参数。
'hbase.client.username'='****',
'hbase.client.password'='****',
--hbase.zookeeper.property.clientPort、mcfed.zookeeper.session.timeout和hbase.client.retries.number为可选参数。
--hbase.zookeeper.property.clientPort的默认值为30020。
['hbase.zookeeper.property.clientPort'='{30020}',]
['mcfed.zookeeper.session.timeout'='<value>',]
['hbase.client.retries.number'='<value>']);
```

- mc_lindorm_external：必填。待创建Lindorm外部表的名称。
- col_name：必填。Lindorm外部表的列名称。
- data_type：必填。列的数据类型。
- with serdeproperties：
  - 'hbase.columns.mapping'='<:key,col-family:col-name1,col-family:col-name2,...>'：必填。Lindorm外部表与Lindorm集群中表的列映射关系。
    - RowKey（`:key`）必须放在最前面。
    - 列格式为`col-family:col-name1,col-family:col-name2`。一个作业中col-family只能有一个。
  - 'odps.external.net.vpc'='true'：必填。固定取值为`true`，表明Lindorm外部表的数据源处于VPC网络。
  - 'odps.vpc.id'='<VPC ID>'：VPC实例ID。您可以登录Lindorm控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可在**连接信息**处获取VPC ID。
  - 'odps.vpc.access.ips'='<VPC ip|host:port,VPC ipBegin-ipEnd:port|network_segment:port>'：必填。Lindorm集群的Java API访问地址。也可以为关联的Lindorm表所属VPC网络IP、IP网段、端口。例如`192.0.2.1-192.0.2.10:9000, 192.0.2.20:70050`或`192.0.2.20/16:2181`。您可以登录Lindorm控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可在**连接信息**处获取。
    - 对于分布式的数据源，请确认需要访问的前端IP端口以及DataNode的IP端口。Lindorm外部表需要填写Zookeeper、HMaster、Region Server的IP和端口（2181/30020、16000、16020）。Lindorm的默认Zookeeper端口为30020。
    - 由于该字段支持IP地址段，且映射IP会耗费资源，默认一次映射的IP数不能超过2000个。
  - 'odps.vpc.region'='<RegionID>'：VPC网络所属地域ID。获取方法请参见获取RegionID及VPC ID。

- hbase://VPC ip|host:port：必填。Lindorm上配置的Zookeeper Master。Lindorm集群的Java API访问地址。
- tblproperties：
  - 'hbase.table.name'='<lindorm_table_name>'：必填。MaxCompute所关联的Lindorm集群中的表的名称。必须提前通过Lindorm控制台建好。
  - 'hbase.zookeeper.quorum'='<VPC ip1|host1:port,VPC ip2|host2:port...>'：必填。Lindorm的Zookeeper的IP或域名。即Lindorm集群的Java API访问地址。
  - hbase.client.username：必填。您可以登录Lindorm控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可通过**连接信息**处的**默认用户名**获取。
  - hbase.client.password：必填。您可以登录Lindorm控制台，在**集群列表**页面单击目标集群名称后，在左侧导航栏单击**数据库连接**，即可通过**连接信息**处的**默认密码**获取。
  - 'hbase.zookeeper.property.clientPort'='{2181|30020}'：可选。Lindorm的Zookeeper端口。Lindorm的默认Zookeeper端口为30020。
  - 'mcfed.zookeeper.session.timeout'='<value>'：可选。Zookeeper的超时时间。单位为毫秒，建议设置为30000。
  - 'hbase.client.retries.number'='<value>'：可选。Lindorm控制台连接重试次数。

> **说明** mcfed.zookeeper和hbase开头的配置是Zookeeper的相关配置。

使用示例

1. 使用Lindorm Shell创建Lindorm表并插入数据。
   i. 创建mf_lindorm_test_t表。
   ```
   create 'mf_lindorm_test_t','f';
   ```
   ii. 向表mf_lindorm_test_t插入数据。
   ```
   put 'mf_lindorm_test_t', '1122', 'f:col1', 'hello';
   put 'mf_lindorm_test_t', '1122', 'f:col2', 'hbase';
   ```
   iii. 查询表mf_lindorm_test_t数据。
   ```
   scan 'mf_lindorm_test_t';
   ```
   查询结果如下图所示：
   ```
   hbase(main):003:0> scan 'mf_lindorm_test_t'
   ROW                                   COLUMN+CELL
    1122                                 column=f:col1, timestamp=1634201741745, value=hello
    1122                                 column=f:col2, timestamp=1634201835934, value=hbase
   1 row(s)
   Took 0.0766 seconds
   ```
2. 创建MaxCompute外部表并映射Lindorm数据源。
   i. 配置环境信息并创建MaxCompute的Lindorm外部表mf_lindorm_test_t_ext。
   ```
   set odps.sql.external.net.vpc=true;
   set odps.sql.hive.compatible = true;
   set odps.sql.hbase.version=lindorm;
   CREATE EXTERNAL TABLE mf_lindorm_test_t_ext
   (
   key  int,
   col1 string,
   col2 string
   )
   STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
   WITH SERDEPROPERTIES (
   'hbase.columns.mapping'=':key,f:col1,f:col2',
   'odps.external.net.vpc'='true',
   'odps.vpc.id'='vpc-2z****',
   'odps.vpc.access.ips'='ld-2ze****-proxy-lindorm.lindorm.rds.aliyuncs.com:30020',
   'odps.vpc.region'='cn-beijing')
   TBLPROPERTIES (
   'hbase.table.name'='mf_lindorm_test_t',
   'hbase.zookeeper.quorum'='ld-2z****-proxy-lindorm.lindorm.rds.aliyuncs.com:30020',
   "hbase.client.username"="root",
   "hbase.client.password"="root");
   ```

ii. 查询外部表mf_lindorm_test_t_ext数据。

```
select * from mf_lindorm_test_t_ext;
```

查询结果如下：

```
+------------+------------+----------+
| key        | col1       | col2     |
+------------+------------+----------+
| 1122       | hello      | hbase    |
+------------+------------+----------+
```

iii. 向外部表mf_lindorm_test_t_ext插入数据。

执行 `INSERT OVERWRITE` 语句向已创建的外部表写入数据。

```
INSERT OVERWRITE TABLE mf_lindorm_test_t_ext SELECT '1123', 'lindorm', 'mftestinsert';
```

iv. 再次查询外部表mf_lindorm_test_t_ext数据。

```
select * from mf_lindorm_test_t_ext;
```

查询结果如下：

```
+------------+------------+-------------+
| key        | col1       | col2        |
+------------+------------+-------------+
| 1122       | hello      | hbase       |
| 1123       | lindorm    | mftestinsert|
+------------+------------+-------------+
```

v. 使用 `scan` 语句查询Lindorm表mf_lindorm_test_t数据。

```
scan 'mf_lindorm_test_t';
```

查询结果如下：

```
hbase(main):004:0> scan 'mf_lindorm_test_t'
ROW                                    COLUMN+CELL
 1122                                  column=f:col1, timestamp=1634201741745, value=hello
 1122                                  column=f:col2, timestamp=1634201835934, value=hbase
 1123                                  column=f:col1, timestamp=1634552508947, value=lindorm
 1123                                  column=f:col2, timestamp=1634552508947, value=mftestinsert
2 row(s)
Took 0.0683 seconds
```

10.10. 外部表常见问题

本文为您介绍外部表的常见问题。

问题类别	常见问题
OSS外部表	• 自定义Extractor在读取非结构化数据时，如果数据字段存在DATETIME类型，报错ODPS-0123131，如何解决？ • 在MaxCompute上访问OSS外部表，编写UDF本地测试通过，上传后报错内存溢出，如何解决？ • 通过外部表处理OSS数据时，报错Inline data exceeds the maximun allowed size，如何解决？ • 如何在MaxCompute中使用OSS外部表读取JSON数据？ • 如何通过OSS外部表将多个小文件输出为一个文件？
性能问题	基于外部表执行SQL作业时，运行慢，如何解决？

自定义Extractor在读取非结构化数据时，如果数据字段存在DATETIME类型，报错ODPS-0123131，如何解决？

- 问题现象

  自定义Extractor在读取非结构化数据时，如果数据字段存在DATETIME类型（例如2019-11-11 06:43:36），会返回如下报错。

```
FAILED: ODPS-0123131:User defined function exception - Traceback:
java.lang.IllegalArgumentException
    at java.sql.Date.valueOf(Date.java:143)
    at com.aliyun.odps.udf.example.text.TextExtractor.textLineToRecord(TextExtractor.java:194)
    at com.aliyun.odps.udf.example.text.TextExtractor.extract(TextExtractor.java:153)
    at com.aliyun.odps.udf.ExtractorHandler.extract(ExtractorHandler.java:120)
```

- 产生原因

查看指定位置的代码 `Date.valueOf(parts[i])` ，其中 `java.sql.Date.valueOf()` 函数只支持形如 `"yyyy-[m]m-[d]d"` 的STRING类型参数，不支持DATETIME时间类型参数。

- 解决措施
  i. 引入Joda-Time依赖并在代码中增加导入信息。

```
--依赖。
<dependency>
  <groupId>joda-time</groupId>
  <artifactId>joda-time</artifactId>
  <version>2.10</version>
</dependency>
--导入信息。
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
```

  ii. 引入 `DateTimeFormat.forPattern()` 函数，将DATETIME类型的日期格式转化为STRING类型进行读取。

```
record.setDate(index, new Date(DateTime.parse(parts[i], DateTimeFormat.forPattern("yyyy-MM-dd HH:mi:ss")).getMillis()));
```

使用示例如下。

  i. 通过MaxCompute客户端上传Extractor项目打包生成的JAR包。

```
add jar /Users/gary/big_data/odps/text_extractor/target/text_extractor-1.0-SNAPSHOT.jar
```

`/Users/gary/big_data/odps/text_extractor/target/text_extractor-1.0-SNAPSHOT.jar` 为生成JAR包的本地保存路径。

  ii. 通过MaxCompute客户端上传Joda-Time第三方JAR包。

```
add jar /Users/gary/.m2/repository/joda-time/joda-time/2.10/joda-time-2.10.jar
```

`/Users/gary/.m2/repository/joda-time/joda-time/2.10/joda-time-2.10.jar` 为Joda-Time第三方JAR包本地存放路径。

  iii. 上传测试数据至OSS指定的目录下。假设文件名为 `video_play_log.txt` ，示例数据如下。

```
5c661071dba64d5080c91da085ff1073^音乐-点击-快进^26.12.04.68^2019-11-11 06:43:36
```

  iv. 通过外部表读取数据。

```
select * from <project_name>.video_play_log;
```

读取结果如下。

```
+------+-------+---+----------------+
| uuid  | action  | ip  | time      |
+------+-------+---+----------------+
| 5c661071dba64d5080c91da085ff1073 | 音乐-点击-快进 | 26.12.04.68 | 2019-11-11 06:43:36 |
+------+-------+---+----------------+
```

在MaxCompute上访问OSS外部表，编写UDF本地测试通过，上传后报错内存溢出，如何解决？

- 问题现象

在MaxCompute上访问OSS外部表，编写UDF本地测试通过，上传后返回如下报错。

```
FAILED: ODPS-0123131:User defined function exception - Traceback:
java.lang.OutOfMemoryError: Java heap space
```

设置如下参数后运行时间增加但依然报错。

```
set odps.stage.mapper.mem = 2048;
set odps.stage.mapper.jvm.mem = 4096;
```

- 产生原因

外部表的对象文件太多，内存占用过大且未设置分区。

- 解决措施
  - 使用小数据量查询。
  - 将对象文件进行分区，以减少内存占用。

通过外部表处理OSS数据时，报错Inline data exceeds the maximun allowed size，如何解决？

- 问题现象

处理OSS数据时，报错 `Inline data exceeds the maximun allowed size` 。

- 产生原因

OSS Store对于每一个小文件有一个大小限制，如果超过3 GB则报错。

- 解决措施

  针对该问题，您可以通过调整以下两个属性进行处理。其原理是通过属性值调整执行计划，控制每个Reducer写入外部表OSS的数据大小，使得OSS Store文件不超过3 GB的限制。

```
set odps.sql.mapper.split.size=256; #调整每个Mapper读取数据的大小，单位是MB。
set odps.sql.reducer.instances=100; #调整执行计划的Reducer数量。
```

如何在MaxCompute中使用OSS外部表读取JSON数据？

在MaxCompute中使用OSS外部表读取JSON数据的操作，请参见在MaxCompute中使用OSS外部表读取JSON数据。

如何通过OSS外部表将多个小文件输出为一个文件？

通过Logview日志，查看SQL的执行计划中最后一个是Reducer还是Joiner。如果是Reducer，则执行语句`set odps.stage.reducer.num=1;`，如果是Joiner，则执行语句`set odps.stage.joiner.num=1;`。

基于外部表执行SQL作业时，运行慢，如何解决？

基于外部表执行SQL作业时，运行慢的常见情况如下：

- OSS外部表中的GZ压缩文件读取慢
  - 问题现象

    用户创建了一个OSS外部表，数据源为OSS中的GZ压缩文件，大小为200 GB。在读取数据过程中执行缓慢。
  - 产生原因

    由于Map端执行计算的Mapper数量过少，所以SQL处理慢。
  - 解决措施
    - 对于结构化数据，您可以设置以下参数调整单个Mapper读取数据量的大小，加速SQL执行。

```
set odps.sql.mapper.split.size=256; #调整每个Mapper读取Table数据的大小，单位是MB。
```

    - 对于非结构化数据，您需要查看OSS外部表路径下的OSS文件是否只有1个。如果只有1个，由于压缩方式下的非结构化数据不支持拆分，所以只能生产1个Mapper，导致处理速度较慢。建议您在OSS对应的外部表路径下，将OSS大文件拆分为小文件，从而增加读取外部表生成的Mapper数量，提升读取速度。
- 使用SDK搜索MaxCompute外部表数据速度慢
  - 问题现象

    使用SDK搜索MaxCompute外部表数据速度慢。
  - 解决措施

    外部表仅支持全量搜索，所以较慢，建议您改用MaxCompute内部表。
- 查询外部表Tablestore数据慢
  - 问题现象

    查询外部表Tablestore的数据慢，同样的业务数据，1个实时写入Tablestore，1个定时写入MaxCompute，两个表结构和数据量一样。查询MaxCompute内部表耗时远小于查询Tablestore外部表。
  - 解决措施

    这种情况可能是对1份数据进行了多次计算，导致速度慢。相比每次从Tablestore远程读取数据，更高效快速的方法是先一次性把需要的数据导入到MaxCompute内部，转为MaxCompute内部表，再进行查询。

11.交互式分析 (Lightning)

11.1. MaxCompute Lightning服务下线公告

MaxCompute Lightning服务（即使用 `lightning.cn-*` 相关Endpoint的服务）由于技术架构变更，将于2021年12月31日00点00分关闭MaxCompute Lightning新服务的开通入口。

已存在的MaxCompute Lightning服务目前仍可以继续使用，将于2022年12月31日00点00分下线。

针对MaxCompute查询加速的交互式分析场景，建议您升级成MaxCompute交互式分析（Hologres），详情请参见MaxCompute Lightning升级为实时数仓Hologres。MaxCompute交互式分析（Hologres）兼容Postgres协议，原有应用无需修改，查询性能更稳定，功能更丰富，如需咨询相关详情，请添加钉钉群：Lightning迁移支持群（群号：35028911）。

> 说明 您现有通过MaxCompute Lightning实现的业务也可以通过MaxCompute MCQA（查询加速）来支持，详情请参见MCQA概述。

11.2. MaxCompute Lightning升级为实时数仓Hologres

MaxCompute Lightning正式升级为实时数仓Hologres产品。本文为您介绍实时数仓Hologres的功能、开通方式及在线支持。

功能介绍

实时数仓Hologres是为大数据设计的实时交互式分析产品，它与MaxCompute无缝打通，支持数据实时写入，支持PB级数据进行高并发、低延时的分析处理。兼容PostgreSQL协议，可以使用您最熟悉的BI工具对海量数据进行自助的多维分析透视和业务探索，同时也支持超高QPS点查能力，满足数仓分析、服务一体化需求。

开通方式

实时数仓Hologres于2020年11月20日最新推出共享集群，提供无服务器计算（Serverless）的服务方式，消除了资源扩展和弹性方面的限制，让您能够直接对MaxCompute数据仓库中的离线数据运行SQL查询进行实时BI分析，只需按扫描的字节数付费。

实时数仓Hologres共享集群正在火热公测中，欢迎体验试用，您可以单击实时数仓Hologres共享集群（MaxCompute BI加速版）开通。

在线支持

欢迎您加入实时数仓Hologres官方答疑群（钉钉群号为：32314975），以获得更加专业、高效的支持。

11.3. 概述

MaxCompute Lightning是MaxCompute产品的交互式查询服务，支持以PostgreSQL协议及语法连接访问MaxCompute项目。方便您使用熟悉的工具以标准SQL查询分析MaxCompute项目中的数据，快速获取查询结果。

您可以使用主流BI工具（例如Tableau、帆软等）或SQL客户端轻松连接到MaxCompute项目，开展BI分析或即席查询。或者利用MaxCompute Lightning的快速查询特性，将项目表数据封装成API对外服务，无需数据迁移就能够支持更丰富的应用场景。

MaxCompute Lightning提供无服务器计算（Serverless）的服务方式，您无需管理任何基础设施，仅需要为运行的查询付费。

关键特性

- 兼容PostgreSQL

  MaxCompute Lightning提供兼容PostgreSQL协议的JDBC/ODBC接口，所有支持PostgreSQL数据库的工具或应用使用默认驱动都可以轻松地连接到MaxCompute项目。您也可以使用更广泛的PostgreSQL生态工具来分析MaxCompute的数据。

- 显著提升性能

  针对MaxCompute表的快速查询进行了优化，特别是在小数据集、并发场景下有更好的性能表现。从而能够支撑更丰富的应用场景，如固定报表、API开放等。

- 统一的权限管理

  作为MaxCompute产品内的服务，通过MaxCompute Lightning连接到MaxCompute项目的访问完全遵循MaxCompute项目的权限体系，在访问用户权限范围内安全地查询数据。

- 开箱即用，按查询付费

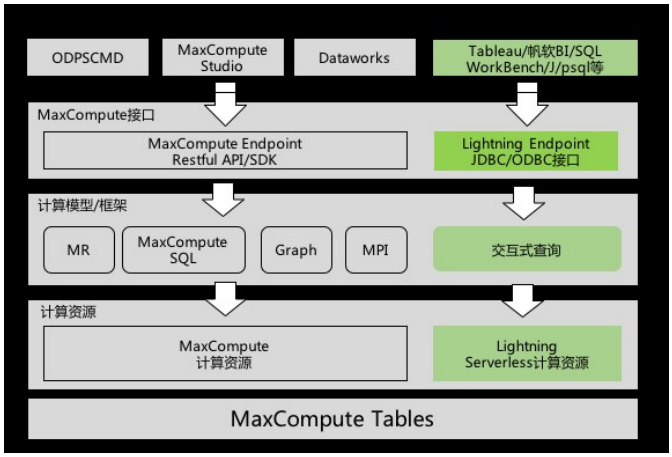
  MaxCompute Lightning是在MaxCompute已有的计算资源之外提供的无服务器的计算服务，您不需要设置、管理或运维MaxCompute Lightning资源，通过MaxCompute Lightning连接后即可开展查询。

  使用MaxCompute Lightning时，只需为每次查询所实际处理的数据量付费，不查询时不产生费用。

系统结构

作为MaxCompute的交互式查询服务，MaxCompute Lightning提供了配套的接入访问域名地址（Endpoint），客户端工具及应用通过PostgreSQL驱动连接访问Lightning JDBC/ODBC接口服务，在MaxCompute项目统一的权限体系下安全地访问项目数据。

通过该服务接口连接并提交的查询任务，都将使用MaxCompute Lightning的Serverless计算资源以保障交互式查询的服务质量。



应用场景

- 即席查询（Ad Hoc）

  利用MaxCompute Lightning面向小规模数据集（百GB规模内）查询性能优化的特性，您可以直接对MaxCompute表开展低时延的查询操作，而不需要再把数据再导入到其它各种系统进行加速（例如ADS、RDS），节约资源和管理成本。

  场景特点：查询的数据对象自由不固定，逻辑相对复杂，期望快速获取查询结果并调整查询逻辑，对查询时延的要求在几十秒内。使用者为掌握SQL技能的数据分析师，希望使用熟悉的客户端工具来开展查询分析。

- Reporting报表分析

  对MaxCompute项目中通过ETL加工汇总后的结果数据制作分析报表，提供给管理者和业务人员定期查看。

  场景特点：查询的数据对象通常为聚合后的结果数据， 数据量较小、查询逻辑固定且较简单。时延要求低，秒级返回（例如大部分查询不超过5秒，不同查询根据其数据规模和查询复杂度有较大差异）。

- 面向在线应用的消费场景

  直接将MaxCompute项目中的数据封装成为Restful API，支撑在线应用。

  场景特点：利用MaxCompute Lightning作为加速查询引擎，结合阿里云DataWorks的数据服务组件，零开发、无运维地将MaxCompute的表数据开放为API服务。

使用限制

DDL、DML、查询及UDF限制。

11.4. 访问域名（Endpoint）

MaxCompute Lightning提供了单独的域名访问地址（Endpoint），通过该地址您可以访问到阿里云不同区域的MaxCompute Lightning服务。

外网网络下Region和服务连接对照表

Region	开服状态	外网Endpoint
华东1	已开服	lightning.cn-hangzhou.maxcompute.aliyun.com
华东2	已开服	lightning.cn-shanghai.maxcompute.aliyun.com
华北2	已开服	lightning.cn-beijing.maxcompute.aliyun.com
华南1	已开服	lightning.cn-shenzhen.maxcompute.aliyun.com
中国（香港）	公测中	lightning.cn-hongkong.maxcompute.aliyun.com
亚太东南1	公测中	lightning.ap-southeast-1.maxcompute.aliyun.com
其他区域	暂未开服	不涉及

经典网络下Region和服务连接对照表

Region	开服状态	经典网络Endpoint
华东1	已开服	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
华东2	已开服	lightning.cn-shanghai.maxcompute.aliyun-inc.com

Region	开服状态	经典网络Endpoint
华北2	已开服	lightning.cn-beijing.maxcompute.aliyun-inc.com
华南1	已开服	lightning.cn-shenzhen.maxcompute.aliyun-inc.com
中国（香港）	公测中	lightning.cn-hongkong.maxcompute.aliyun-inc.com
亚太东南1	公测中	lightning.ap-southeast-1.maxcompute.aliyun-inc.com
其他区域	暂未开服	不涉及

VPC网络下Region和服务连接对照表

Region	开服状态	VPC网络Endpoint
华东1	已开服	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
华东2	已开服	lightning.cn-shanghai.maxcompute.aliyun-inc.com
华东2金融云	已开服	lightning.cn-shanghai-finance.maxcompute.aliyun-inc.com
华北2	已开服	lightning.cn-beijing.maxcompute.aliyun-inc.com
华南1	已开服	lightning.cn-shenzhen.maxcompute.aliyun-inc.com
中国（香港）	公测中	lightning.cn-hongkong.maxcompute.aliyun-inc.com
亚太东南1	公测中	lightning.ap-southeast-1.maxcompute.aliyun-inc.com
其他区域	暂未开服	不涉及

11.5. 使用限制

本文为您介绍使用MaxCompute Lightning服务的使用限制。

DDL/DML语句使用限制

MaxCompute Lightning目前不支持Update、Create、Delete、Insert操作，仅支持对MaxCompute表执行Select操作，后续版本将陆续开放相关能力。

查询使用限制

- 查询分区表时，扫描分区数的最大值为1024个。
- 目前不支持创建和使用视图。
- 目前不支持的数据类型：MAP、ARRAY、TINYINT、BINARY、TIMESTAMP以及带精度的DECIMAL。
- 每个查询中对单张表最大的数据扫描量为1 TB。
- 提交的查询语句的长度不超过100 KB。
- 查询超时时间为1小时。
- 单个查询语句的扫描数据量不超过100 GB。
- 单个查询语句的Join、Group By数量总和最大值为5。
- 单个查询的结果集不能超过10000，如果超出此数量限制，则会被截断。
- 由于数据服务通过Lightning查询MaxCompute，Lightning兼容PostgreSQL，而PostgreSQL中没有DATETIME数据类型，所以如果您的数据有DATETIME类型，数据服务中会映射为PostgreSQL的TIMESTAMP类型进行查询。

UDF使用限制

- 当前不支持在MaxCompute Lightning中使用MaxCompute创建的UDF。
- 当前不支持在MaxCompute Lightning中创建和使用PostgreSQL UDF（支持使用PostgreSQL内建函数）。
- 当前不支持MaxCompute内建函数。

Query并发约束

单个MaxCompute项目的MaxCompute Lightning查询并发数限制为20。

11.6. 快速入门

本教程将引导您使用主流的第三方工具连接MaxCompute Lightning服务，查看指定MaxCompute项目下的数据表，进行BI分析。

前提条件

- 开通MaxCompute并创建项目。

  使用MaxCompute Lightning需要您已经有创建好的MaxCompute项目。

  如果您还没有开通阿里云MaxCompute项目，请参见开通MaxCompute和DataWorks。

- 创建表并导入数据。

  使用MaxCompute Lightning前需要您创建表并导入数据，详情请参见创建表。

- 获取云账号信息。

  使用MaxCompute Lightning前需要MaxCompute项目用户的Accesss ID及Access Key云账号信息。

  您可登录阿里云官网，进入管理控制台**用户信息管理**页面进行查看。如果您的子账号没有查看权限，请联系主账号管理员索取，同时确定该子账号有权限查看指定的数据表。



准备第三方客户端工具

MaxCompute Lightning兼容PostgreSQL接口，所有支持PostgreSQL数据库连接的客户端工具都可以用于连接MaxCompute Lightning。

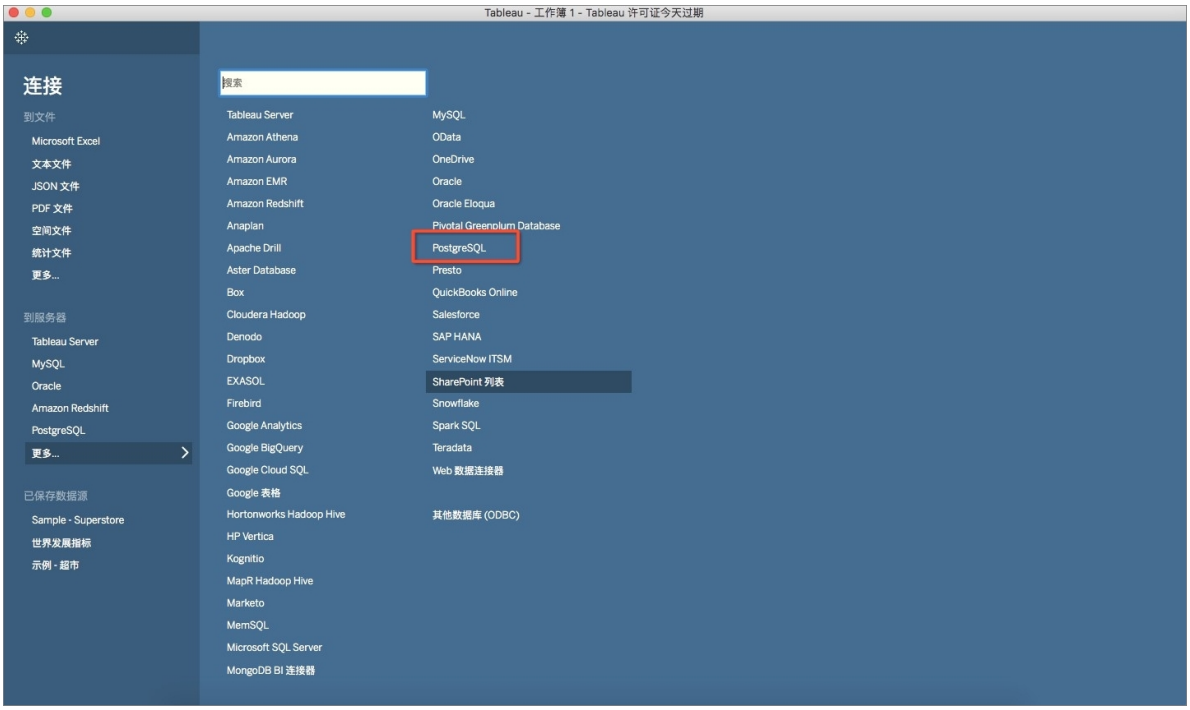
本文选择大家熟悉的BI工具Tableau Desktop进行举例，相关工具请到Tableau官网进行下载。其他常见的客户端工具，例如SQL Workbench/J、PSQL、帆软BI、MicroStrategy BI等，都可以像连接PostgreSQL数据库一样配置连接Lightning服务。

连接服务并开展分析

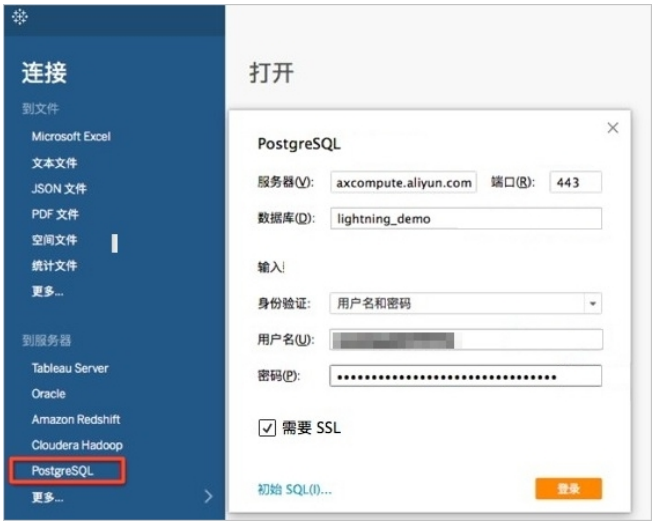
配置、连接服务成功后，您可以查看指定MaxCompute项目下的数据表，进行BI分析。

1. 连接服务器时选择PostgreSQL。

   打开Tableau Desktop，选择**连接 > 到服务器 > 更多 > PostgreSQL**。



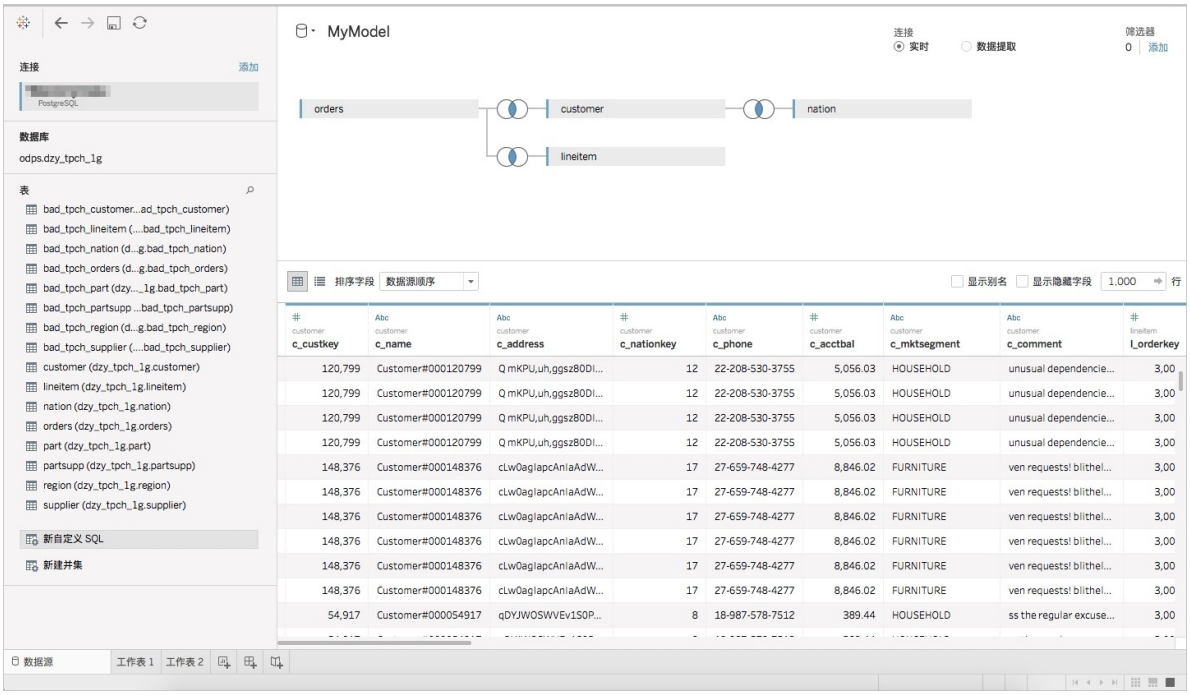
2. 填写服务连接及用户认证信息。



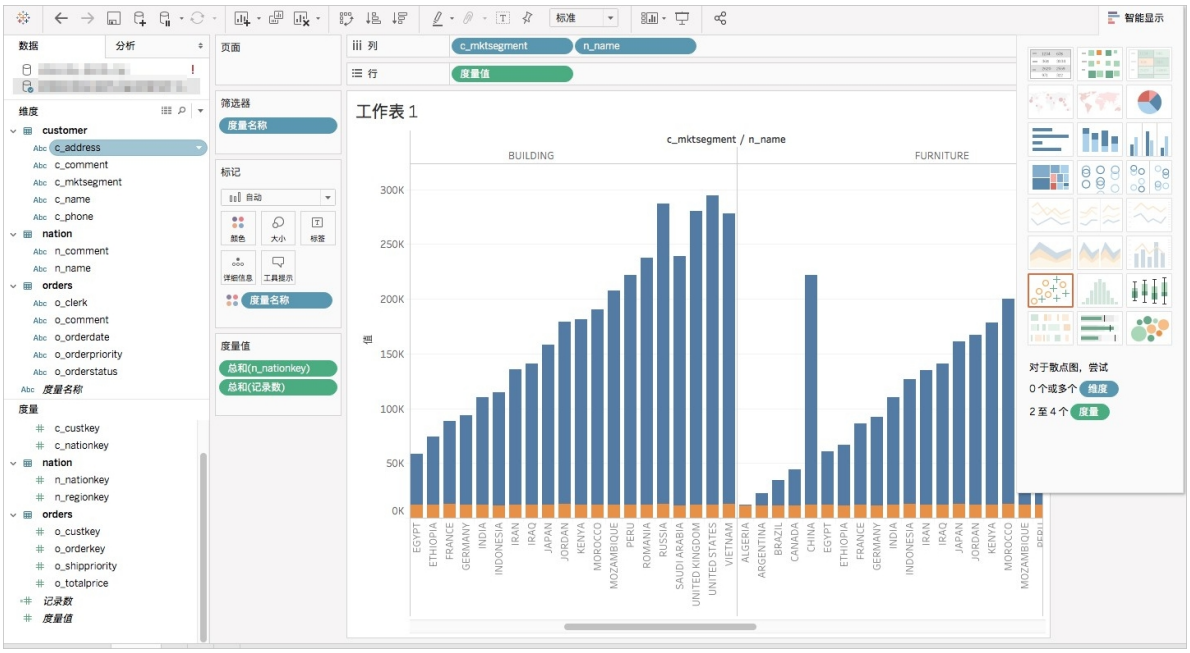
参数	说明
服务器	填写所在区域对应的MaxCompute Lightning服务的EndPoint，如华东2区域的Endpoint为：`lightning.cn-shanghai.maxcompute.aliyun.com`，详情请参见访问域名（Endpoint）。
端口	443。
数据库	MaxCompute项目名称。
身份验证	用户名和密码。
用户名/密码	您的Access Key ID、Access Key Secret。
SSL连接	勾选需要SSL。

3. 获取项目表信息，创建数据源及模型。

   配置好信息并登录后，Tableau会加载所连接的MaxCompute项目下所有的表，您可以根据需要选择对应的表创建模型和工作表。



选择特定数据的维度和指标，创建工作表。



至此，您已经使用Tableau工具成功连接MaxCompute Lightning服务，可以对连接到MaxCompute项目内的数据进行BI分析。

> 说明 为了获得更好的性能和体验，建议您使用Tableau支持的TDC文件方式对Lightning数据源进行连接定制优化，详情请参见通过其他常见工具连接MaxCompute Lightning服务。

11.7. 使用说明

11.7.1. 通过JDBC连接MaxCompute服务

11.7.1.1. JDBC驱动程序

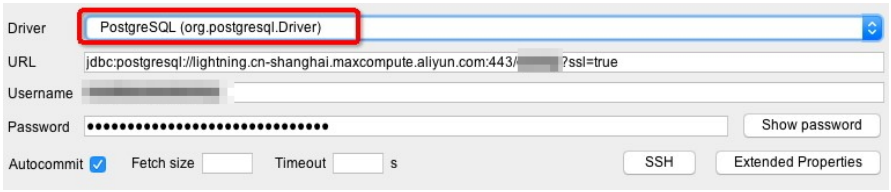
MaxCompute提供完全兼容PostgreSQL消息协议的Java数据库连接（JDBC）接口，使用者可以通过JDBC将SQL客户端工具连接到MaxCompute Lightning服务。

MaxCompute Lightning支持PostgeSQL官方驱动连接，同时也提供了为Lightning服务而优化的驱动程序供选择。

MaxCompute Lightning查询引擎基于PostgreSQL 8.2，当前仅支持对已有MaxCompute表进行SELECT查询，更多详情请参见查询语法及函数。

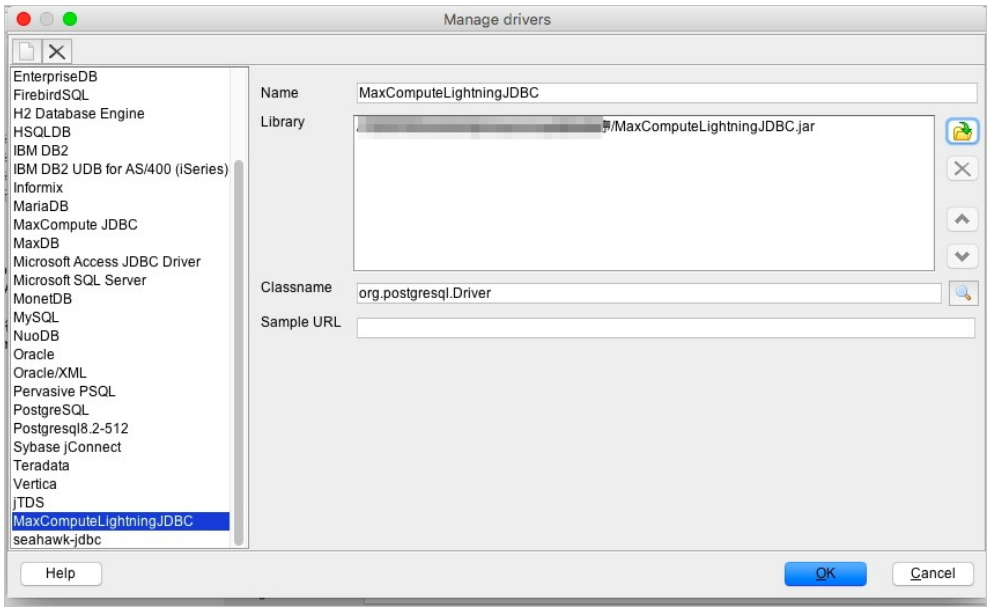
- 使用PostgeSQL官方提供的JDBC驱动程序。

  通常，客户端工具默认集成了PostgreSQL数据库的驱动，直接使用工具自带的驱动即可。如果未集成，可从客户端官网下载。以SQL Workbench/J客户端为例，创建连接时可选择PostgreSQL官方驱动。

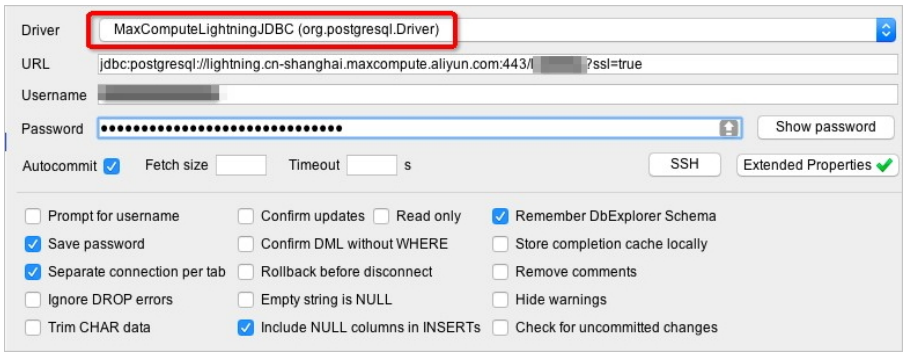


- 使用阿里云MaxCompute Lightning优化过的JDBC驱动，以获取更好的性能。

将下载后的MaxCompute Lightning的JDBC驱动程序保存为*MaxComputeLightningJDBC.jar*文件。以SQL Workbench/J客户端为例，在**驱动管理**菜单中，添加MaxCompute Lightning JDBC驱动程序项界面如下。



在创建连接时，从Driver列表中选择刚才添加的MaxCompute Lightning JDBC驱动。



11.7.1.2. 配置JDBC连接

您需要通过配置MaxCompute Lightning JDBC URL，才能将SQL客户端与MaxCompute Lightning服务连通。

- JDBC URL命名格式

```
jdbc:postgresql://Endpoint:port/project_name[?ssl=true][&prepareThreshold=0][&preferQueryMode=simple][&user=UserName][&password=PassWord]
```

- 参数说明

参数	取值	说明
Endpoint	所在地域网络环境下的Lightning访问域名。	例如通过外网访问上海地域的服务使用 `lightning.cn-shanghai.maxcompute.aliyun.com` 。Endpoint详情请参见访问域名（Endpoint）。
port	443。	无。
project_name	MaxCompute的项目名称。	您可以通过登录MaxCompute控制台，在**项目管理**页签查看需要连接的MaxCompute的项目名称。
user	访问用户的AccessKey ID。	阿里云账号或RAM用户的AccessKey ID。您可以进入AccessKey管理页面获取AccessKey ID。
password	访问用户的AccessKey Secret。	AccessKey ID对应的AccessKey Secret。您可以进入AccessKey管理页面获取AccessKey Secret。
ssl	true。	MaxCompute Lightning服务端默认开启SSL服务，客户端需要使用SSL进行连接。

参数	取值	说明
prepareThreshold	0。	可选。需要使用JDBC PrepareStatement功能时，建议设置 `prepareThreshold=0` 。
preferQueryMode	simple。	可选。当PreparedStatement中使用分区过滤条件时，建议使用此参数。

- 使用示例
  - 示例1：JDBC URL中未配置访问用户的AccessKey ID和访问用户的AccessKey Secret。

    ```
    jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject?ssl=true
    ```

    > **说明** 按照如上配置后，您还需要输入user、password参数后才能连接成功。您可以将参数直接添加到JDBC URL中，详情请参考示例2。

  - 示例2：JDBC URL中配置访问用户的AccessKey ID和访问用户的AccessKey Secret。

    ```
    jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject?ssl=true&user=Tom&password=******
    ```

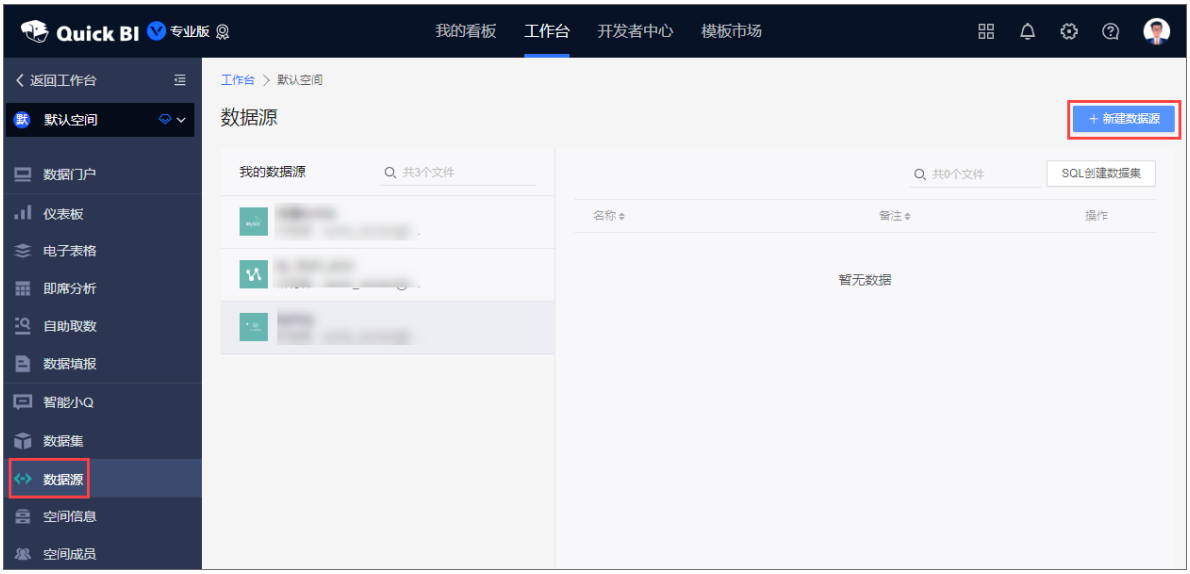
11.7.1.3. 通过其他常见工具连接MaxCompute Lightning服务

本文为您介绍几款常见客户端工具，除此之外，原则上支持PostgreSQL的工具都可以通过Lightning来对接访问MaxCompute项目。

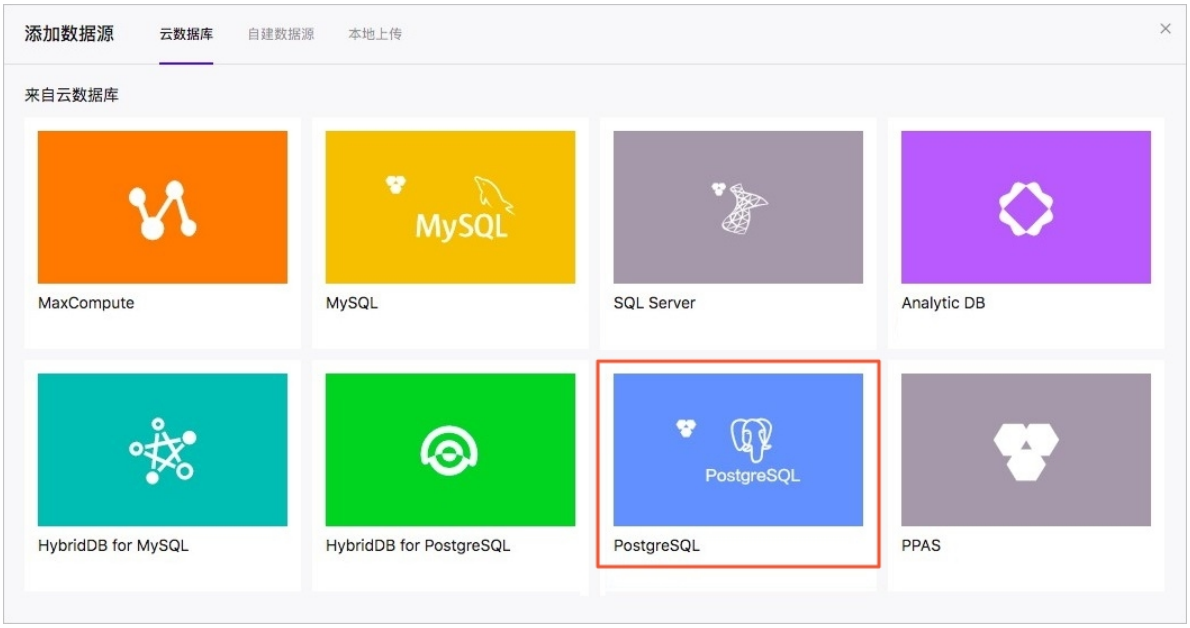
阿里云Quick BI

Quick BI是一个专为云上用户量身打造的易上手、性能强的大数据分析及可视化平台，可以让每个人都能成为数据分析师。使用Quick BI连接MaxCompute Lightning服务的操作步骤如下：

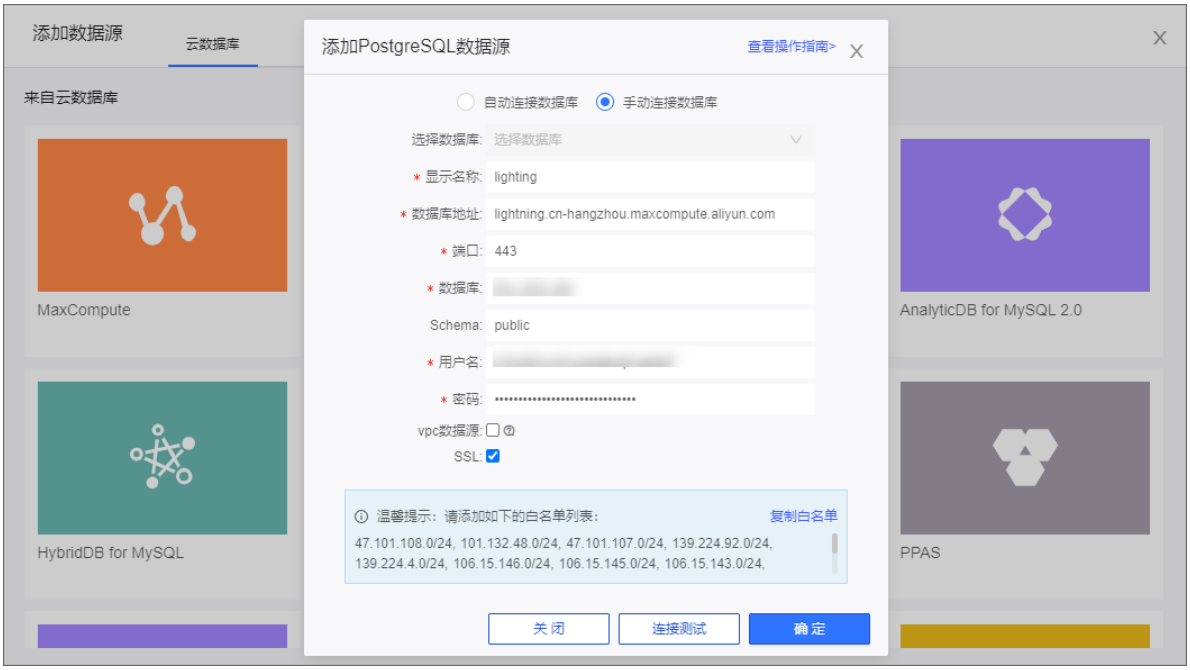
1. 登录Quick BI控制台，在左侧导航栏单击**数据源**。
2. 在**数据源**界面右上角，单击**新建数据源**。



3. 在**添加数据源**对话框的**云数据库**或**自建数据源**页签，选择**PostgreSQL**类型。



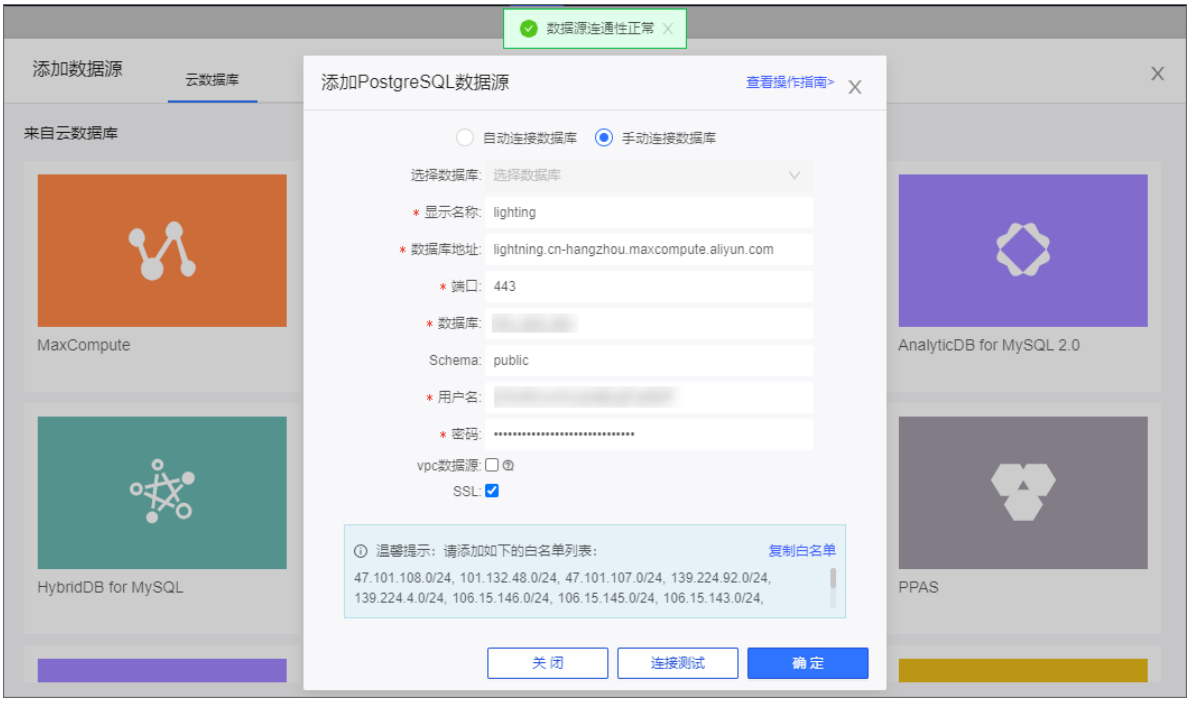
4. 在**添加PostgreSQL数据源**对话框，选择**手动连接数据库**，并配置MaxCompute Lightning的连接信息。



参数	说明
显示名称	新建数据源的名称。
数据库地址	MaxCompute项目所属地域的Lighting服务Endpoint，您可以根据网络类型及项目所属地域选择Endpoint。不同网络下不同地域的Lightning服务Endpoint信息，请参见访问域名（Endpoint）。
端口	固定取值为443。
数据库	待连接MaxCompute项目的名称。 此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
Schema	固定取值为public。
用户名	具备目标MaxCompute项目访问权限的AccessKey ID。 您可以进入AccessKey管理页面获取AccessKey ID。

参数	说明
密码	AccessKey ID对应的AccessKey Secret。 您可以进入AccessKey管理页面获取AccessKey Secret。
SSL	选中SSL。

5. 在**添加PostgreSQL数据源**对话框，单击**连接测试**，连接成功后单击**确定**。



更多通过Quick BI执行查询及分析数据操作的详细信息，请参见使用Quick BI查询及分析数据。

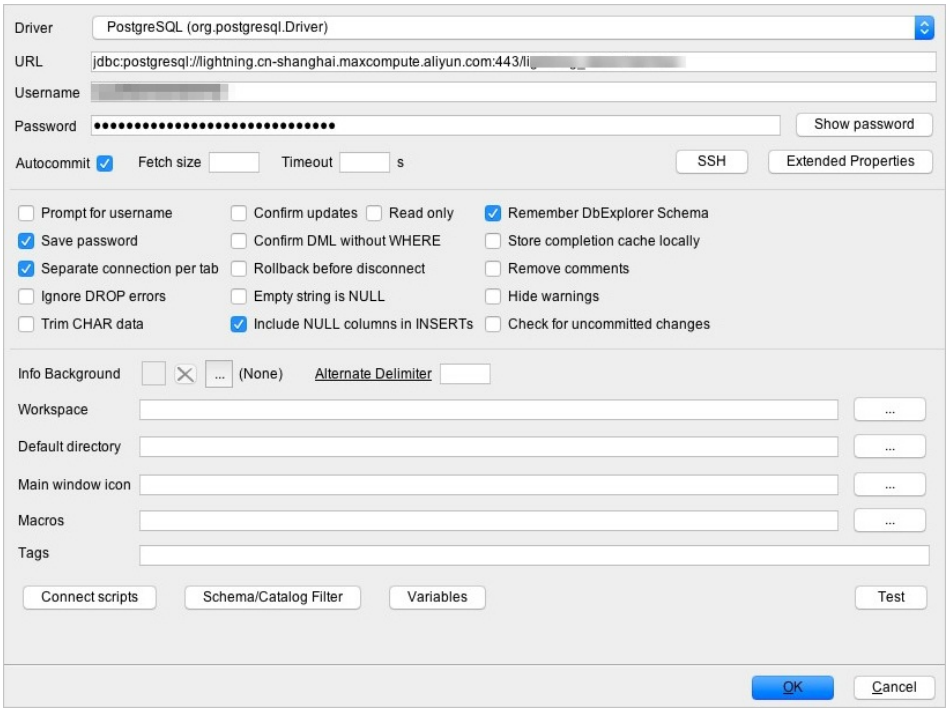
SQL Workbench/J

SQL Workbench/J是一款流行的免费、跨平台SQL查询分析工具，使用SQL Workbench/J可以通过PostgreSQL驱动连接MaxCompute Lightning服务。使用SQL Workbench/J连接MaxCompute Lightning服务的操作步骤如下：

1. 启动SQL Workbench/J，进入SQL Workbench界面。

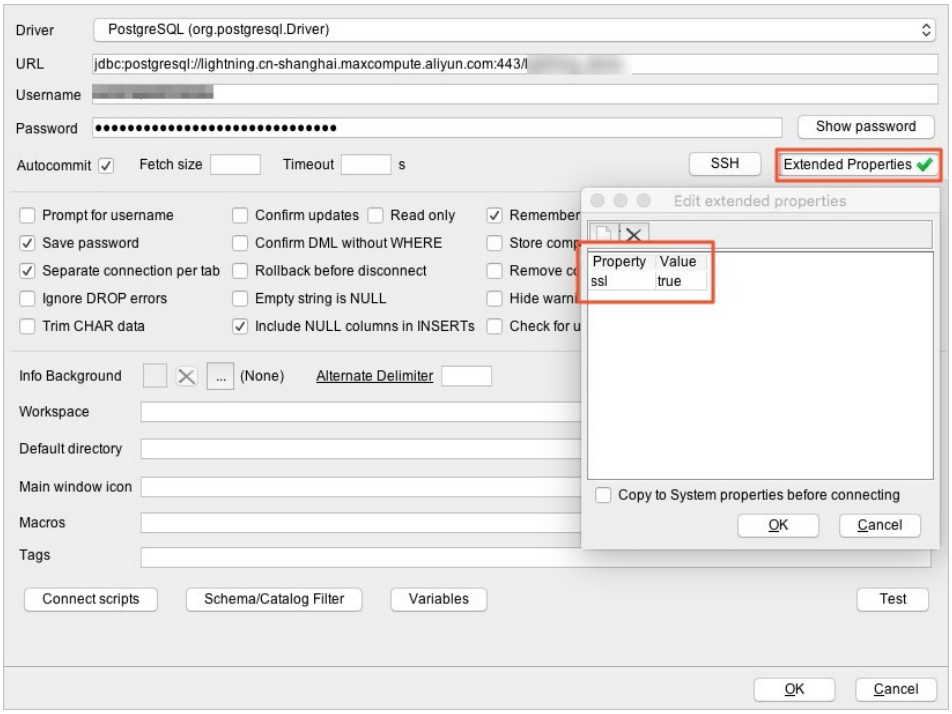
   进入SQL Workbench界面时，系统会自动弹出**Select Connection Profile**对话框。

2. 在**Select Connection Profile**对话框右侧的配置界面，配置下表所列参数信息。其他选项按照图示配置即可。



参数	说明
Driver	用于连接MaxCompute项目的驱动程序。固定取值为**PostgreSQL (org.postgresql.Driver)**。
URL	连接MaxCompute项目的URL。格式为 `jdbc:postgresql://<Lighting_endpoint>:<port>/<MaxCompute_project_name>` ，配置时删除 `<>` 符号。参数说明如下： ○ Lighting_endpoint：MaxCompute项目所属地域的Lighting服务Endpoint，您可以根据网络类型及项目所属地域选择Endpoint。不同网络下不同地域的Lightning服务Endpoint信息，请参见访问域名（Endpoint）。 ○ port：固定取值为443。 ○ MaxCompute_project_name：待连接的目标MaxCompute项目名称。 此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
Username	具备目标MaxCompute项目访问权限的AccessKey ID。 您可以进入AccessKey管理页面获取AccessKey ID。
Password	AccessKey ID对应的AccessKey Secret。 您可以进入AccessKey管理页面获取AccessKey Secret。

3. 在**Select Connection Profile**对话框，单击**Extended Properties**，添加SSL属性，取值为True，单击**OK**。



4. 在**Select Connection Profile**对话框右下角单击**Test**，连接成功后单击**OK**。

更多通过SQL Workbench执行查询及分析数据操作的详细信息，请参见使用SQL Workbench查询及分析数据。

psql工具

psql是PostgreSQL的一个命令行交互式客户端工具，在本机安装PostgreSQL数据库将默认安装psql客户端。

您可以直接在psql客户端运行如下命令来连接MaxCompute Lightning服务。

```
psql -h <endpoint> -U <username> -d <MaxCompute_project_name> -p <port>
```

参数	说明
endpoint	MaxCompute项目所属地域的Lighting服务Endpoint，您可以根据网络类型及项目所属地域选择Endpoint。不同网络下不同地域的Lightning服务Endpoint信息，请参见访问域名（Endpoint）。
username	具备目标MaxCompute项目访问权限的AccessKey ID。 您可以进入AccessKey管理页面获取AccessKey ID。
MaxCompute_project_name	待连接的目标MaxCompute项目名称。 此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
port	固定取值为443。

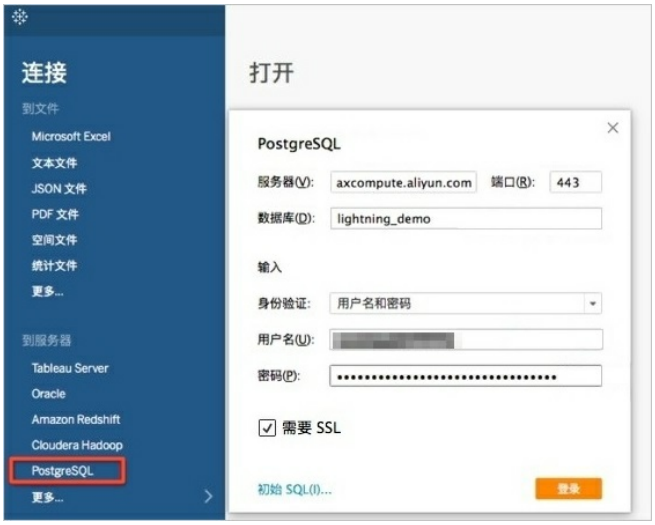
执行上述命令后，在psql客户端的密码提示符处，输入**username**对应的AccessKey Secret，即可创建连接。示例如下。

```
[~]$psql -h lightning.cn-shanghai.maxcompute.aliyun.com  -U [redacted]z  -d [redacted] -p 443
Password for user [redacted]
psql (10.3, server 8.2.15)
Type "help" for help.

lightning=> \d
                List of relations
  Schema   |       Name        | Type  |   Owner
-----------+-------------------+-------+-----------
 lightning | customer          | table | proxy_role
 lightning | lineitem          | table | proxy_role
 lightning | nation            | table | proxy_role
 lightning | orders            | table | proxy_role
 lightning | orders_partition  | table | proxy_role
 lightning | orders_partition2 | table | proxy_role
 lightning | orders_partition3 | table | proxy_role
 lightning | part              | table | proxy_role
 lightning | partsupp          | table | proxy_role
 lightning | region            | table | proxy_role
 lightning | supplier          | table | proxy_role
 lightning | test              | table | proxy_role
(12 rows)

lightning=> select * from orders limit 10;
 o_orderkey | o_custkey | o_orderstatus | o_totalprice | o_orderdate | o_orderpriority |     o_clerk     | o_shippriority |                                o_comment
-----------+-----------+---------------+--------------+-------------+-----------------+-----------------+----------------+-----------------------------------------------------------------------
   21638945 |   9836128 | O             |    149664.71 | 1997-08-01  | 5-LOW           | Clerk#000000016 |              0 | ular requests are quickly ironic reque
   21638946 |    781658 | O             |     70792.99 | 1997-03-08  | 1-URGENT        | Clerk#000066763 |              0 | ons wake even, bold requests. slyly bold requests snooze slyly fin
   21638947 |  12320353 | O             |    231895.68 | 1996-01-13  | 3-MEDIUM        | Clerk#000078663 |              0 | lithely regular deposits affix q
   21638948 |   8140645 | F             |     128980.5 | 1995-02-18  | 4-NOT SPECIFIED | Clerk#000082968 |              0 |  regular, even requests? furiously enticing ins
   21638949 |   4800883 | P             |    289947.16 | 1995-05-08  | 4-NOT SPECIFIED | Clerk#000035612 |              0 | d theodolites among the slow dolphins ca
   21638950 |   3428813 | F             |    243629.29 | 1993-09-08  | 1-URGENT        | Clerk#000024564 |              0 | ic, ironic excuses haggle silent instructions.
   21638951 |  13056718 | F             |    187252.37 | 1994-09-05  | 1-URGENT        | Clerk#000045762 |              0 | rouches engage blithely among the blithely regu
   21638976 |   9240733 | F             |     130092.7 | 1993-04-09  | 2-HIGH          | Clerk#000020449 |              0 | nto beans. furiously express Tiresias above the regular a
   21638977 |  13996019 | F             |    247587.28 | 1994-02-16  | 3-MEDIUM        | Clerk#000071761 |              0 | posits haggle. deposits
   21638978 |   1084246 | F             |    275689.34 | 1993-09-01  | 4-NOT SPECIFIED | Clerk#000085923 |              0 |  the blithely regular deposits. requests kindle fluffily. ideas nag s
(10 rows)
```

Tableau Desktop

Tableau Desktop是Tableau基于斯坦福大学突破性技术研发的软件应用程序，可以帮助您生动地分析实际存在的任何结构化数据，并在几分钟内生成美观的图表、坐标图、仪表盘与报告。使用Tableau Desktop连接MaxCompute Lightning服务的操作步骤如下：

1. 启动Tableau Desktop，进入Tableau Desktop界面。
2. 在左侧导航栏的**到服务器**区域，选择**PostgreSQL**。
3. 在**PostgreSQL**对话框，配置下表所列参数信息。

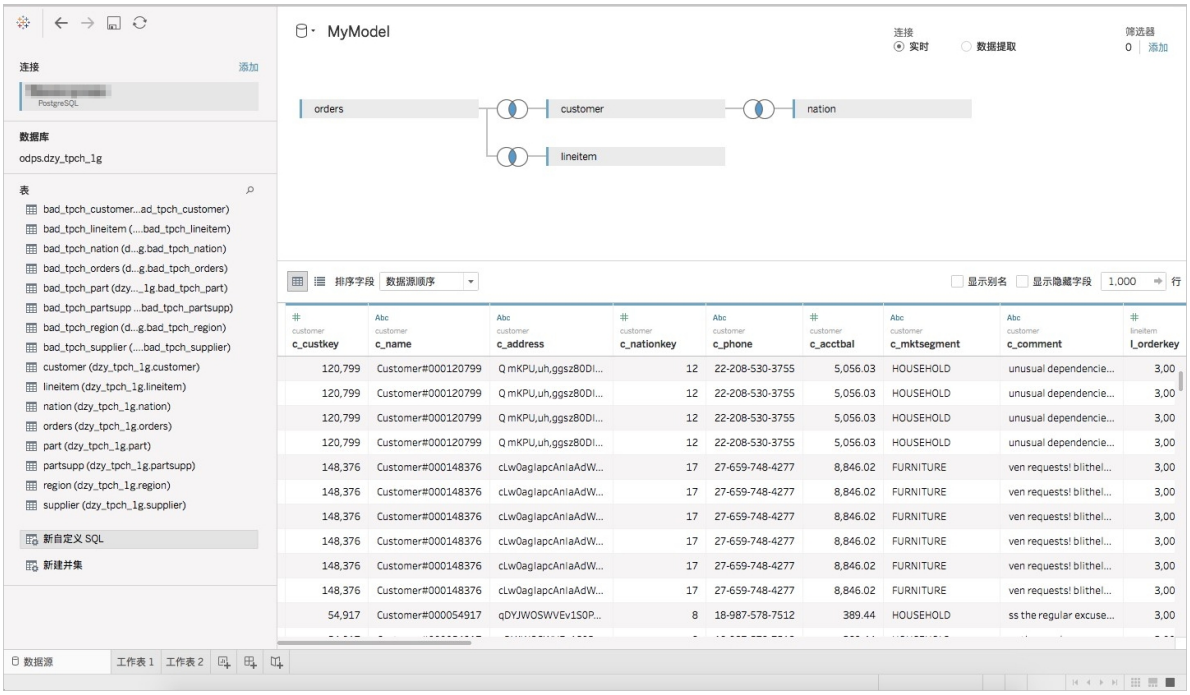


参数	说明
服务器	MaxCompute项目所属地域的Lighting服务Endpoint，您可以根据网络类型及项目所属地域选择Endpoint。不同网络下不同地域的Lightning服务Endpoint信息，请参见访问域名（Endpoint）。
端口	固定取值为443。
数据库	待连接MaxCompute项目的名称。 此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
身份验证	在下拉列表选择**用户名和密码**。
用户名	具备目标MaxCompute项目访问权限的AccessKey ID。 您可以进入AccessKey管理页面获取AccessKey ID。

参数	说明
密码	AccessKey ID对应的AccessKey Secret。 您可以进入AccessKey管理页面获取AccessKey Secret。
需要SSL	选中**需要SSL**。

4. 在**PostgreSQL**对话框右下角，单击**登录**。

   登录后，即可通过Tableau创建工作表并进行可视化分析。



> **说明** 时间函数会默认返回UTC时区的时间。如果需要返回北京时间，则需要进行如下配置。

```
SET timezone = 'Asia/Shanghai';
```

为了获得更好的性能和体验，建议您使用Tableau支持的TDC文件方式，对Lightning数据源进行连接定制优化。具体操作如下：

1. 将如下xml内容保存为*postgresql.tdc*文件。

```
<?xml version='1.0' encoding='utf-8' ?>
<connection-customization class='postgres' enabled='true' version='8.10'>
<vendor name='postgres'/>
<driver name='postgres'/>
<customizations>
<customization name='CAP_CREATE_TEMP_TABLES' value='no' />
<customization name='CAP_STORED_PROCEDURE_TEMP_TABLE_FROM_BUFFER' value='no' />
<customization name='CAP_CONNECT_STORED_PROCEDURE' value='no' />
<customization name='CAP_SELECT_INTO' value='no' />
<customization name='CAP_SELECT_TOP_INTO' value='no' />
<customization name='CAP_ISOLATION_LEVEL_SERIALIZABLE' value='yes' />
<customization name='CAP_SUPPRESS_DISCOVERY_QUERIES' value='yes' />
<customization name='CAP_SKIP_CONNECT_VALIDATION' value='yes' />
<customization name='CAP_ODBC_TRANSACTIONS_SUPPRESS_EXPLICIT_COMMIT' value='yes' />
<customization name='CAP_ODBC_TRANSACTIONS_SUPPRESS_AUTO_COMMIT' value='yes' />
<customization name='CAP_ODBC_REBIND_SKIP_UNBIND' value='yes' />
<customization name='CAP_FAST_METADATA' value='no' />
<customization name='CAP_ODBC_METADATA_SUPPRESS_SELECT_STAR' value='yes' />
<customization name='CAP_ODBC_METADATA_SUPPRESS_EXECUTED_QUERY' value='yes' />
<customization name='CAP_ODBC_UNBIND_AUTO' value='yes' />
<customization name='SQL_TXN_CAPABLE' value='0' />
<customization name='CAP_ODBC_CURSOR_FORWARD_ONLY' value='yes' />
<customization name='CAP_ODBC_TRANSACTIONS_COMMIT_INVALIDATES_PREPARED_QUERY' value='yes' />
</customizations>
</connection-customization>
```

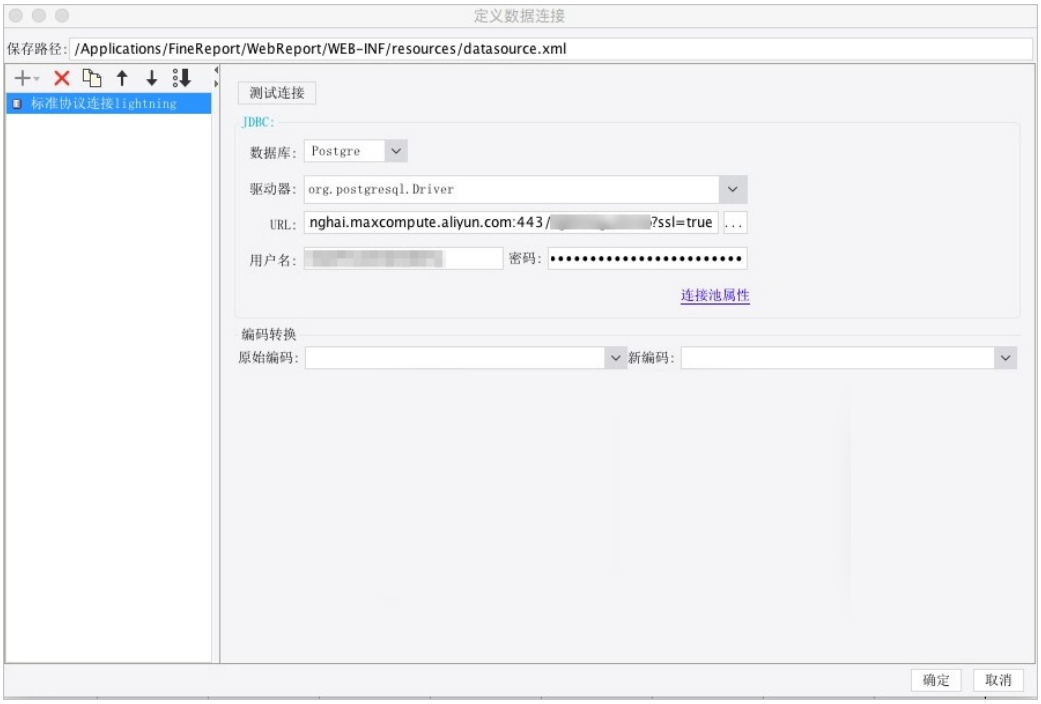
2. 将文件保存到 `\My Tableau Repository\Datasources` 目录下。如果是Tableau Server，Windows系统请保存在 `\Tableau\Tableau Server\data\tabsvc\vizqlserver\Datasources` ；Linux系统请保存在 `/var/opt/tableau/tableau_server/data/tabsvc/vizqlserver/Datasources/` 。
3. 重启Tableau，使用PostgreSQL数据源连接MaxCompute Lightning服务。更多关于TDC文件定制数据源的内容，请参见Tableau官方帮助文档。

更多通过Tableau执行查询及分析数据操作的详细信息，请参见使用Tableau查询及分析数据。

FineReport

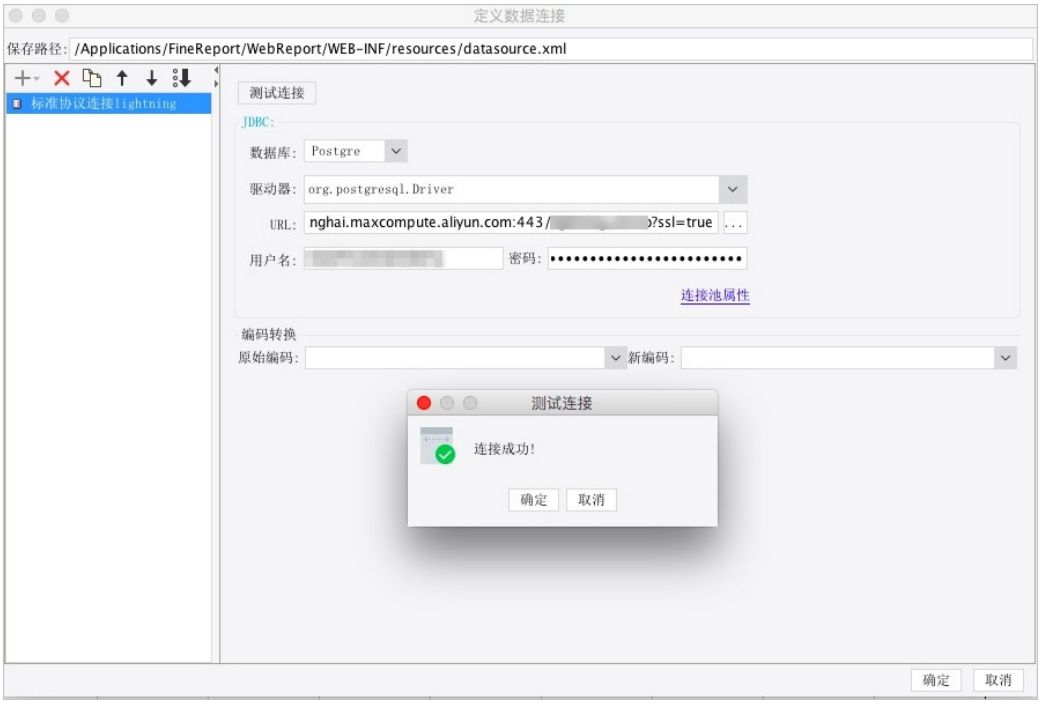
FineReport是帆软软件有限公司自主研发的企业级Web报表工具，秉持零编码的理念，易学易用、功能强大，通过简单的拖拽操作便可制作中国式复杂报表，轻松实现报表的多样化展示、交互分析、数据录入、权限管理、定时调度、打印输出、门户管理和移动应用等需求。使用FineReport连接MaxCompute Lightning服务的操作步骤如下：

1. 启动FineReport，进入FineReport界面。
2. 在顶部菜单栏，选择**服务器 > 定义数据库连接**。
3. 在**定义数据连接**对话框，单击+图标，选择**JDBC**，配置下表所列参数信息。



参数	说明
数据库	在下拉列表选择**Postgre**。
驱动器	用于连接MaxCompute项目的驱动程序。固定取值为**org.postgresql.Driver**，您需要手动输入，不支持在下拉列表选择。
URL	连接MaxCompute项目的URL。格式为 `jdbc:postgresql://<Lightning_endpoint>:<port>/<MaxCompute_project_name>?ssl=true&prepareThreshold=0` 。配置时删除 `<>` 符号。参数说明如下： ○ Lighting_endpoint：MaxCompute项目所属地域的Lighting服务Endpoint，您可以根据网络类型及项目所属地域选择Endpoint。不同网络下不同地域的Endpoint信息，请参见访问域名（Endpoint）。 ○ port：固定取值为443。 ○ MaxCompute_project_name：待连接的目标MaxCompute项目名称。 此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。
用户名	具备目标MaxCompute项目访问权限的AccessKey ID。 您可以进入AccessKey管理页面获取AccessKey ID。
密码	AccessKey ID对应的AccessKey Secret。 您可以进入AccessKey管理页面获取AccessKey Secret。

4. 在参数信息上方单击**测试连接**，连接成功后依次单击**确定**。



更多通过FineReport执行查询及分析数据操作的详细信息，请参见使用FineReport查询及分析数据。

11.7.2. 查询语法与内建函数

在PostgreSQL官方函数基础上，MaxCompute Lightning还支持查询语法、MAX_PT等内建函数。

查询语法

MaxCompute Lightning查询引擎基于PostgreSQL 8.2。当前仅支持对已有MaxCompute表进行SELECT查询，相关语法参见PostgreSQL官方文档。

函数

命令格式：

```
max_pt(table_full_name)
```

命令说明：

对于分区的表，此函数返回该分区表的一级分区的最大值，以及该分区下所有的数据文件并按照字母排序。

参数说明：

table_full_name：String类型，用于指定表名（必须携带项目空间名称，例如 `prj.src` ）。您必须对此表有读权限。

返回值：

返回最大的一级分区的值。

示例：

假设tbl为分区表，对应分区如下，且包含数据文件如下。

```
pt ='20120901'
pt ='20120902'
```

如下语句中分区max_pt返回值为20120902，MaxCompute SQL语句读出pt='20120902'分区下的数据。

```
select * from tbl where pt=max_pt('myproject.tbl');
```

11.7.3. 查看或取消查询作业

Lightning为用户提供了一个系统视图stv_recents，通过查询该视图能够获取当前用户正在执行的所有查询作业。

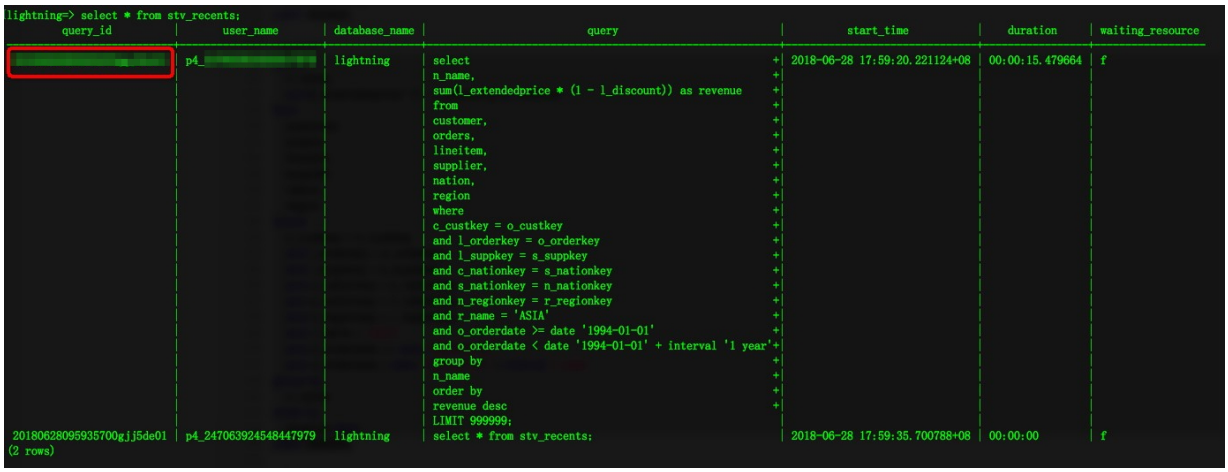
查看运行中的查询

您可以执行查询命令查看正在执行作业的ID、用户、SQL语句、作业开始时间、运行时间、是否等待资源（t为等待资源未执行，f为获取资源并执行中）等信息。

执行如下查询命令。

```
select * from stv_recents;
```

查询结果如下所示。



取消正在运行的查询

通过查询stv_recents获得运行中查询的信息，如果想要取消某个查询，可以执行下述语句。

```
select cancel('query_id');
```

其中的query_id，是运行中查询的query_id信息，示例如下。



12.Graph

12.1. 概述

MaxCompute Graph是一套面向迭代的图计算处理框架。图计算作业使用图进行建模，图由点（Vertex）和边（Edge）组成，点和边包含权值（Value）。

MaxCompute Graph支持以下图编辑操作：

- 修改点或边的权值。
- 增加、删除点。
- 增加、删除边。

> 说明 编辑点和边时，需要您同时在代码中维护点与边的关系。

通过迭代对图进行编辑、演化，最终求解出结果。典型应用有PageRank、单源最短距离算法、K-均值聚类算法 等。您可以使用MaxCompute Graph提供的接口Java SDK编写图计算程序。

基本概念

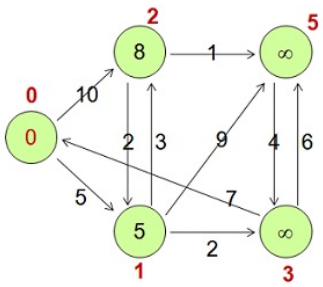
- 图（Graph）：是用于表示对象之间关联关系的一种抽象数据结构。使用顶点（Vertex）和边（Edge）进行描述，顶点表示对象，边表示对象之间的关系。可以抽象为用图描述的数据即为图数据。
- 点（Vertex）：在图模型中用于表示对象。
- 边（Edge）：在图模型中用于表示对象之间的关系。由源ID、目标ID和与该边缘关联的数据组成的单个定向边缘。
- 有向图：即边有方向性的图模型，一条边的两个顶点一般为不同的角色，例如页面A连接向页面B。有向图中的边分为出边和入边。
- 无向图：即边无方向性的图模型，例如用户组中的普通用户。
- 出边：指从当前顶点指向其它顶点的边。
- 入边：其它顶点指向当前顶点的边。
- 度：度表示一个顶点的所有边的数量。
- 出度：是一个顶点出边的数量。
- 入度：是一个顶点入边的数量。
- 超步（SuperStep）：图进行迭代计算时，一次迭代称为一个超步。

Graph数据结构

MaxCompute Graph能够处理的图必须是一个由点（Vertex）和边（Edge）组成的有向图。由于MaxCompute仅提供二维表的存储结构，因此需要您自行将图数据分解为二维表存储在MaxCompute中。您可以根据自身的业务场景进行分解。

在进行图计算分析时，使用自定义的GraphLoader将二维表数据转换为MaxCompute Graph引擎中的点和边。

- 点的结构为<ID, Value, Halted, Edges>，参数分别表示：
  - 点标识符（ID）
  - 权值（Value）
  - 状态（Halted， 表示是否要停止迭代。）
  - 出边集合（Edges，以该点为起始点的所有边列表。）
- 边的结构为<DestVertexID，Value>，参数分别为：
  - 目标点（DestVertexID）
  - 权值（Value）



例如，上图可以表述为以下二维表格式。

Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >]>
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>

Vertex	<ID, Value, Halted, Edges>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

Graph程序逻辑

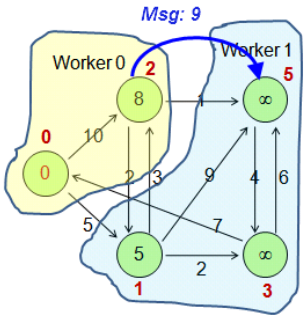
Graph程序主要包含图加载、迭代计算、迭代终止等处理步骤。

1. 图加载

   图加载包含两个步骤：

   i. 图加载：框架调用您自定义的GraphLoader，将输入表的记录解析为点或边。

   ii. 分布式化：框架调用您自定义的Partitioner对点进行分片（默认的分片逻辑是，根据点ID的哈希值对Worker个数取模分片），分配到相应的Worker。

   例如，假设上图中Worker数是2，则v0、v2会被分配到Worker0，因为ID对2取模结果为0。而v1、v3、v5将被分配到Worker1，ID对2取模结果为1。

2. 迭代计算

   一次迭代为一个超步（SuperStep），遍历所有非结束状态（Halted值为False）的点或者收到消息的点（处于结束状态的点收到信息会被自动唤醒），并调用其 `compute(ComputeContext context, Iterable messages)` 方法。

   在您实现的 `compute(ComputeContext context, Iterable messages)` 方法中：

   - 处理上一个超步发给当前点的消息（Messages）。
   - 根据需要对图进行编辑：
     - 修改点、边的取值。
     - 发送消息给某些点。
     - 增加、删除点或边。
   - 通过Aggregator汇总信息到全局信息，详情请参见Aggregator机制。
   - 设置当前点状态，结束或非结束状态。
   - 迭代进行过程中，框架会将消息以异步的方式发送到对应Worker，并在下一个超步进行处理，无需人工干预。

3. 迭代终止

   满足以下任意一条，迭代即终止：

   - 所有点处于结束状态（Halted值为True）且没有新消息产生。
   - 达到最大迭代次数。
   - 某个Aggregator的 `terminate` 方法返回True。

   Graph程序的伪代码描述如下所示。

```
// 1. load
for each record in input_table {
  GraphLoader.load();
}
// 2. setup
WorkerComputer.setup();
for each aggr in aggregators {
  aggr.createStartupValue();
}
for each v in vertices {
  v.setup();
}
// 3. superstep
for (step = 0; step < max; step ++) {
  for each aggr in aggregators {
    aggr.createInitialValue();
  }
  for each v in vertices {
     v.compute();
   }
}
// 4. cleanup
for each v in vertices {
  v.cleanup();
}
WorkerComputer.cleanup();
```

12.2. Aggregator机制

本文将为您介绍Aggregator的执行机制、相关API，并以Kmeans Clustering为例说明Aggregator的具体用法。

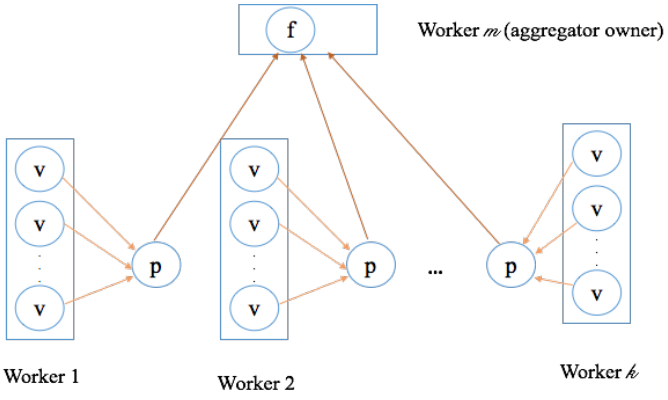
Aggregator是MaxCompute Graph作业中常用的特征，特别适用于解决机器学习问题。MaxCompute Graph中，Aggregator用于汇总并处理全局信息。

Aggregator机制

Aggregator的逻辑分为两部分：

- 一部分在所有Worker上执行，即分布式执行。
- 一部分只在Aggregator Owner所在的Worker上执行，即单点执行。

其中，在所有Worker上执行的操作包括创建初始值及局部聚合，然后将局部聚合结果发送给Aggregator Owner所在的Worker上。Aggregator Owner所在的Worker上聚合普通Worker发送过来的局部聚合对象，得到全局聚合结果，然后判断迭代是否结束。全局聚合的结果会在下一轮超步（迭代）分发给所有Worker，供下一轮迭代使用。



Aggregator的基本流程如下：

1. 每个Worker启动时执行createStartupValue用于创建AggregatorValue。
2. 每轮迭代开始前，每个Worker执行createInitialValue来初始化本轮的AggregatorValue。
3. 一轮迭代中每个点通过 `context.aggregate()` 来执行 `aggregate()` 实现Worker内的局部迭代。
4. 每个Worker将局部迭代结果发送给AggregatorOwner所在的Worker。
5. AggregatorOwner所在Worker执行多次 `merge` ，实现全局聚合。
6. AggregatorOwner所在Worker执行 `terminate` 用于处理全局聚合结果，并决定是否结束迭代。

Aggregator的API

Aggregator共提供了5个API供您实现。API的调用时机及常规用途如下：

- createStartupValue(context)

  该API在所有Worker上执行一次，调用时机是所有超步开始之前，通常用于初始化 `AggregatorValue` 。在第0轮超步中，调用 `WorkerContext.getLastAggregatedValue()` 或 `ComputeContext.getLastAggregatedValue()` 可以获取该API初始化的 `AggregatorValue` 对象。

- createInitialValue(context)

  该API在所有Worker上每轮超步开始时调用一次，用于初始化本轮迭代所用的 `AggregatorValue` 。通常操作是通过 `WorkerContext.getLastAggregatedValue()` 得到上一轮迭代的结果，然后执行部分初始化操作。

- aggregate(value, item)

  该API同样在所有Worker上执行，与上述API不同的是，该API由用户显示调用 `ComputeContext#aggregate(item)` 来触发，而上述两个API由框架自动调用。该API用于执行局部聚合操作，其中第一个参数 `value` 是本Worker在该轮超步已经聚合的结果（初始值是 `createInitialValue` 返回的对象），第二个参数是您的代码调用 `ComputeContext#aggregate(item)` 传入的参数。该API中通常用 `item` 来更新 `value` 实现聚合。所有 `aggregate` 执行完后，得到的 `value` 就是该Worker的局部聚合结果，然后由框架发送给Aggregator Owner所在的Worker。

- merge(value, partial)

  该API执行于Aggregator Owner所在Worker，用于合并各Worker局部聚合的结果，达到全局聚合对象。与 `aggregate` 类似， `value` 是已经聚合的结果，而 `partial` 待聚合的对象，同样用 `partial` 更新 `value` 。

  假设有3个Worker，分别是w0、w1、w2，其局部聚合结果是p0、p1、p2。例如，发送到Aggregator Owner所在Worker的顺序为p1、p0、p2，则 `merge` 执行次序为：

  i. 首先执行 `merge(p1, p0)` ，这样p1和p0就聚合为p1。

  ii. 然后执行 `merge(p1, p2)` ，p1和p2聚合为p1，而p1即为本轮超步全局聚合的结果。

  由上述示例可见，当只有一个Worker时，不需要执行 `merge` 方法，即 `merge()` 不会被调用。

- terminate(context, value)

  当Aggregator Owner所在Worker执行完 `merge()` 后，框架会调用 `terminate(context, value)` 执行最后的处理。其中第二个参数 `value` ，即为 `merge()` 最后得到全局聚合，在该方法中可以对全局聚合继续修改。执行完 `terminate()` 后，框架会将全局聚合对象分发给所有Worker，供下一轮超步使用。 `terminate()` 方法的一个特殊之处在于，如果返回True，则整个作业就结束迭代，否则继续执行。在机器学习场景中，通常判断收敛后返回True以结束作业。

Kmeans Clustering示例

下面以典型的Kmeans Clustering为例，为您介绍Aggregator的具体用法。

> 说明 完整代码请参见Kmeans，此处为解析代码。

- GraphLoader部分

  GraphLoader部分用于加载输入表，并转换为图的点或边。这里我们输入表的每行数据为一个样本，一个样本构造一个点，并用Vertex的Value来存放样本。

  首先定义一个Writable类 `KmeansValue` 作为Vertex的value类型。

```
public static class KmeansValue implements Writable {
    DenseVector sample;
    public KmeansValue() {
    }
    public KmeansValue(DenseVector v) {
        this.sample = v;
    }
    @Override
        public void write(DataOutput out) throws IOException {
        wirteForDenseVector(out, sample);
    }
    @Override
        public void readFields(DataInput in) throws IOException {
        sample = readFieldsForDenseVector(in);
    }
}
```

  `KmeansValue` 中封装一个 `DenseVector` 对象来存放一个样本，这里 `DenseVector` 类型来自matrix-toolkits-java，而 `wirteForDenseVector()` 及 `readFieldsForDenseVector()` 用于实现序列化及反序列化。

  自定义的 `KmeansReader` 代码，如下所示。

```
public static class KmeansReader extends
    GraphLoader<LongWritable, KmeansValue, NullWritable, NullWritable> {
    @Override
        public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, KmeansValue, NullWritable, NullWritable> context)
        throws IOException {
        KmeansVertex v = new KmeansVertex();
        v.setId(recordNum);
        int n = record.size();
        DenseVector dv = new DenseVector(n);
        for (int i = 0; i < n; i++) {
            dv.set(i, ((DoubleWritable)record.get(i)).get());
        }
        v.setValue(new KmeansValue(dv));
        context.addVertexRequest(v);
    }
}
```

`KmeansReader` 中，每读入一行数据创建一个点，这里用 `recordNum` 作为点的ID，将 `record` 内容转换成 `DenseVector` 对象并封装进 `VertexValue` 中。

- Vertex部分

  自定义的 `KmeansVertex` 代码如下。逻辑非常简单，每轮迭代要做的事情就是将自己维护的样本执行局部聚合。具体逻辑参见下面Aggregator的实现。

```
public static class KmeansVertex extends
    Vertex<LongWritable, KmeansValue, NullWritable, NullWritable> {
    @Override
        public void compute(
        ComputeContext<LongWritable, KmeansValue, NullWritable, NullWritable> context,
        Iterable<NullWritable> messages) throws IOException {
        context.aggregate(getValue());
    }
}
```

- Aggregator部分

  整个Kmeans的主要逻辑集中在Aggregator中。首先是自定义的 `KmeansAggrValue` ，用于维护要聚合及分发的内容。

```
public static class KmeansAggrValue implements Writable {
    DenseMatrix centroids;
    DenseMatrix sums; // used to recalculate new centroids
    DenseVector counts; // used to recalculate new centroids
    @Override
        public void write(DataOutput out) throws IOException {
        wirteForDenseDenseMatrix(out, centroids);
        wirteForDenseDenseMatrix(out, sums);
        wirteForDenseVector(out, counts);
    }
    @Override
        public void readFields(DataInput in) throws IOException {
        centroids = readFieldsForDenseMatrix(in);
        sums = readFieldsForDenseMatrix(in);
        counts = readFieldsForDenseVector(in);
    }
}
```

`KmeansAggrValue` 维护了3个对象：

  - `centroids` 是当前的K个中心点。如果样本是m维， `centroids` 就是一个K*m的矩阵。
  - `sums` 是和 `centroids` 大小一样的矩阵，每个元素记录了到特定中心点最近的样本特定维之和。例如 `sums(i,j)` 是到第i个中心点最近的样本的第j维度之和。
  - `counts` 是个K维的向量，记录到每个中心点距离最短的样本个数。 `sums` 和 `counts` 一起用于计算新的中心点，也是要聚合的主要内容。

  接下来是自定义的Aggregator实现类 `KmeansAggregator` ，按照上述API的顺序分析其实现。

  i. `createStartupValue()` 的实现。

```
public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {
    public KmeansAggrValue createStartupValue(WorkerContext context) throws IOException {
        KmeansAggrValue av = new KmeansAggrValue();
        byte[] centers = context.readCacheFile("centers");
        String lines[] = new String(centers).split("\n");
        int rows = lines.length;
        int cols = lines[0].split(",").length; // assumption rows >= 1
        av.centroids = new DenseMatrix(rows, cols);
        av.sums = new DenseMatrix(rows, cols);
        av.sums.zero();
        av.counts = new DenseVector(rows);
        av.counts.zero();
        for (int i = 0; i < lines.length; i++) {
            String[] ss = lines[i].split(",");
            for (int j = 0; j < ss.length; j++) {
                av.centroids.set(i, j, Double.valueOf(ss[j]));
            }
        }
        return av;
    }
```

在该方法中初始化一个 `KmeansAggrValue` 对象，然后从资源文件 `centers` 中读取初始中心点，并赋值给 `centroids` 。而 `sums` 和 `counts` 初始化为0。

ii. `createInitialValue()` 的实现。

```
@Override
public KmeansAggrValue createInitialValue(WorkerContext context)
    throws IOException {
    KmeansAggrValue av = (KmeansAggrValue)context.getLastAggregatedValue(0);
    // reset for next iteration
    av.sums.zero();
    av.counts.zero();
    return av;
}
```

该方法首先获取上一轮迭代的 `KmeansAggrValue` ，然后将 `sums` 和 `counts` 清零，只保留了上一轮迭代出的 `centroids` 。

iii. `aggregate()` 的实现。

```
@Override
public void aggregate(KmeansAggrValue value, Object item)
    throws IOException {
    DenseVector sample = ((KmeansValue)item).sample;
    // find the nearest centroid
    int min = findNearestCentroid(value.centroids, sample);
    // update sum and count
    for (int i = 0; i < sample.size(); i ++) {
        value.sums.add(min, i, sample.get(i));
    }
    value.counts.add(min, 1.0d);
}
```

该方法中调用 `findNearestCentroid()` （实现见附件）找到样本 `item` 距离最近的中心点索引，然后将其各个维度加到 `sums` 上，最后 `counts` 计数加1。

以上3个方法执行于所有Worker上，实现局部聚合。在Aggregator Owner所在Worker执行的全局聚合相关操作如下：

i. `merge()` 的实现。

```
@Override
public void merge(KmeansAggrValue value, KmeansAggrValue partial)
    throws IOException {
    value.sums.add(partial.sums);
    value.counts.add(partial.counts);
}
```

`merge` 的实现逻辑很简单，就是把各个Worker聚合出的 `sums` 和 `counts` 相加即可。

ii. `terminate()` 的实现。

```
@Override
public boolean terminate(WorkerContext context, KmeansAggrValue value)
    throws IOException {
    // Calculate the new means to be the centroids (original sums)
    DenseMatrix newCentriods = calculateNewCentroids(value.sums, value.counts, value.centroids);
    // print old centroids and new centroids for debugging
    System.out.println("\nsuperstep: " + context.getSuperstep() +
                       "\nold centriod:\n" + value.centroids + " new centriod:\n" + newCentriods);
    boolean converged = isConverged(newCentriods, value.centroids, 0.05d);
    System.out.println("superstep: " + context.getSuperstep() + "/"
                       + (context.getMaxIteration() - 1) + " converged: " + converged);
    if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
        // converged or reach max iteration, output centriods
        for (int i = 0; i < newCentriods.numRows(); i++) {
            Writable[] centriod = new Writable[newCentriods.numColumns()];
            for (int j = 0; j < newCentriods.numColumns(); j++) {
                centriod[j] = new DoubleWritable(newCentriods.get(i, j));
            }
            context.write(centriod);
        }
        // true means to terminate iteration
        return true;
    }
    // update centriods
    value.centroids.set(newCentriods);
    // false means to continue iteration
    return false;
}
```

`teminate()` 中首先根据 `sums` 和 `counts` 调用 `calculateNewCentroids()` 求平均计算出新的中心点。然后调用 `isConverged()` 根据新老中心点欧拉距离判断是否已经收敛。如果收敛或迭代次数达到最大数，则将新的中心点输出并返回True，以结束迭代。否则更新中心点并返回False以继续迭代。其中 `calculateNewCentroids()` 和 `isConverged()` 的实现见附件。

- `main` 方法

  `main` 方法用于构造 `GraphJob` ，然后设置相应配置，并提交作业。

```
public static void main(String[] args) throws IOException {
    if (args.length < 2)
        printUsage();
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(KmeansReader.class);
    job.setRuntimePartitioning(false);
    job.setVertexClass(KmeansVertex.class);
    job.setAggregatorClass(KmeansAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
        job.setMaxIteration(Integer.parseInt(args[2]));
    long start = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
                       + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}
```

> **说明** `job.setRuntimePartitioning(false)` 设置为False后，各个Worker加载的数据不再根据Partitioner重新分区，即谁加载的数据谁维护。

12.3. 使用限制

本文为您介绍MaxCompute Graph的使用限制。

- 单个Job引用的Resource数量不超过256个，Table、Archive按照一个单位计算。
- 单个Job引用的Resource总计字节数大小不超过512 MB。
- 单个Job的输入路数不能超过1024（输入表的个数不能超过64）。单个Job的输出路数不能超过256。
- 多路输出中指定的Label不能为NULL或者空字符串，长度不能超过256个字符串，只能包括A-Z、a-z、0-9、下划线（_）、井号（#）、英文句点（.）和短划线（-）。
- 单个Job中自定义Counter的数量不能超过64个。Counter的 `group name` 和 `counter name` 中不能带有井号（#），两者长度和不能超过100。
- 单个Job的Worker数由框架计算得出，最大为1000个，超过抛异常。

- 单个Worker占用CPU默认为200个，范围为[50,800]。
- 单个Worker占用Memory默认为4096 MB，范围为[256 MB,12 GB]。
- 单个Worker重复读一个Resource次数限制不大于64次。
- `split_size` 默认为64 MB，您可自行设置，范围为0< `split_size` ≤（9223372036854775807>>20）。
- MaxCompute Graph程序中的GraphLoader、Vertex、Aggregator等在集群运行时，受到Java沙箱的限制（Graph作业的主程序则不受此限制），具体限制请参见Java沙箱。

MaxCompute Graph处于公测阶段，各开服区域仅提供少量的资源用来公测。

- 华东2（上海）：仅提供2000 CU。
- 华南1（深圳）：仅提供500 CU。
- 其它开服区域：仅提供100 CU。

12.4. 使用说明

本文为您介绍如何运行MaxCompute Graph作业。

运行作业

MaxCompute客户端提供一个JAR命令用于运行MaxCompute Graph作业，其使用方式与MapReduce中的JAR命令相同。

使用语法如下。

```
Usage: jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS]
    -conf <configuration_file>         Specify an application configuration file
    -classpath <local_file_list>       classpaths used to run mainClass
    -D <name>=<value>                  Property value pair, which will be used to run mainClass
    -local                             Run job in local mode
    -resources <resource_name_list>    file/table resources used in graph, seperate by comma
```

其中<GENERIC_OPTIONS>包括以下参数（均为可选）：

- -conf <configuration file>：指定JobConf配置文件。
- -classpath <local_file_list>：本地执行时的classpath，主要用于指定main函数所在的JAR包。

  通常，用户更习惯于将main函数与Graph作业编写在一个包中，例如单源最短距离。因此，在执行示例程序时，-resources及-classpath的参数中都出现了JAR包，但二者意义不同：

  - -resources引用的是Graph作业，运行于分布式环境中。
  - -classpath引用的是main函数，运行于本地，指定的JAR包路径也是本地文件路径。包名之间使用系统默认的文件分割符进行分割（通常，Windows系统是分号，Linux系统是冒号）。
- -D <prop_name>=<prop_value>：本地执行时，<mainClass>的Java属性可以定义多个。
- -local：以本地模式执行Graph作业，主要用于程序调试。
- -resources <resource_name_list>：Graph作业运行时使用的资源声明。通常，resource_name_list中需要指定运行Graph作业所使用的资源名称。如果您在Graph作业中读取了其他MaxCompute资源，则这些资源名称也需要被添加到<resource_name_list>中。资源之间使用逗号分隔。如果使用跨项目空间资源时，需要在前面加上PROJECT_NAME/resources/。示例：`-resources otherproject/resources/resfile`。

同时，您也可以直接运行Graph作业的main函数，直接将作业提交到MaxCompute，而不是通过MaxCompute客户端提交作业。以PageRank算法为例，如下所示。

```
public static void main(String[] args) throws Exception {
  if (args.length < 2)
    printUsage();
  Account account = new AliyunAccount(accessId, accessKey);
  Odps odps = new Odps(account);
  odps.setEndpoint(endPoint);
  odps.setDefaultProject(project);
  SessionState ss = SessionState.get();
  ss.setOdps(odps);
  ss.setLocalRun(false);
  String resource = "mapreduce-examples.jar";
  GraphJob job = new GraphJob();
  // 将使用的JAR及其他文件添加到class cache resource，对应于JAR命令中-libjars中指定的资源。
  job.addCacheResourcesToClassPath(resource);
  job.setGraphLoaderClass(PageRankVertexReader.class);
  job.setVertexClass(PageRankVertex.class);
  job.addInput(TableInfo.builder().tableName(args[0]).build());
  job.addOutput(TableInfo.builder().tableName(args[1]).build());
  // default max iteration is 30
  job.setMaxIteration(30);
  if (args.length >= 3)
    job.setMaxIteration(Integer.parseInt(args[2]));
  long startTime = System.currentTimeMillis();
  job.run();
  System.out.println("Job Finished in "
      + (System.currentTimeMillis() - startTime) / 1000.0
      + " seconds");
}
```

输入及输出

MaxCompute Graph作业的输入输出限制为表，不允许您自定义输入输出格式。

- 作业输入定义如下。

```
GraphJob job = new GraphJob();
job.addInput(TableInfo.builder().tableName("tblname").build());  //表作为输入。
job.addInput(TableInfo.builder().tableName("tblname").partSpec("pt1=a/pt2=b").build()); //分区作为输入。
//只读取输入表的col2和col0列，在GraphLoader的load方法中，record.get(0)得到的是col2列，顺序一致。
job.addInput(TableInfo.builder().tableName("tblname").partSpec("pt1=a/pt2=b").build(), new String[]{"col2", "col0"});
```

> **说明**
> - 支持多路输入。
> - addInput框架读取输入表的记录传给用户自定义的GraphLoader，载入图数据。
> - 暂时不支持分区过滤条件，更多应用限制请参见 应用限制。

- 作业输出定义如下所示。

```
GraphJob job = new GraphJob();
//输出表为分区表时需要给到最末一级分区
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").build());
//下面的参数中，true表示覆盖tableinfo指定的分区，即INSERT OVERWRITE语义，false表示INSERT INTO语义。
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").lable("output1").build(), true);
```

> **说明**
> - 支持多路输出，通过Label标识每路输出。
> - Graph作业在运行时可以通过WorkerContext的write方法写出记录到输出表，多路输出需要指定标识。

读取资源

- 通过GraphJob读取资源。

  除了通过JAR命令指定Graph读取的资源外，还可以通过GraphJob的以下两种方法指定。

```
void addCacheResources(String resourceNames)
void addCacheResourcesToClassPath(String resourceNames)
```

- 通过WorkerContext读取资源。

  您可以通过WorkerContext的相应上下文对象读取资源。

```
public byte[] readCacheFile(String resourceName) throws IOException;
public Iterable<byte[]> readCacheArchive(String resourceName) throws IOException;
public Iterable<byte[]> readCacheArchive(String resourceName, String relativePath)throws IOException;
public Iterable<WritableRecord> readResourceTable(String resourceName);
public BufferedInputStream readCacheFileAsStream(String resourceName) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName, String relativePath) throws IOException;
```

> 说明
> - Graph也支持在WorkerComputer的setup方法中读取资源，保存在Worker Value中，之后通过getWorkerValue方法获取。
> - 如果您需要在读取资源的同时并行进行资源处理，可以使用上述流接口。该读取方式可以减少内存的消耗。
> - 更多应用限制请参见应用限制。

12.5. 编写Graph

本文将以单源最短距离（Single Source Shortest Path，SSSP）算法为例，为您介绍如何提交Graph作业。

前提条件

- 已安装并配置MaxCompute客户端。

  安装与配置MaxCompute客户端，详情请参见MaxCompute客户端。
- 已安装并配置MaxCompute Studio。

  安装与配置MaxCompute Studio，详情请参见安装MaxCompute Studio和配置MaxCompute Studio。
- 已安装JDK 1.8或以上版本。
- 已准备数据文件。本文以sssp.txt文件为例进行演示，数据示例如下。

```
1 2:2,3:1,4:4
2 1:2,3:2,4:1
3 1:1,2:2,5:1
4 1:4,2:1,5:1
5 3:1,4:1
```

操作步骤

1. 运行MaxCompute客户端，并执行如下命令创建输入表sssp_in和输出表sssp_out。

   ```
   CREATE TABLE sssp_in (v bigint, es string);
   CREATE TABLE sssp_out (vertex bigint, value bigint);
   ```

   > 说明 更多表语法说明，请参见表操作。

2. 执行Tunnel命令，上传sssp.txt文件中的数据至表sssp_in， 并以空格键做两列的分隔符。

   ```
   tunnel u -fd " " sssp.txt sssp_in;
   ```

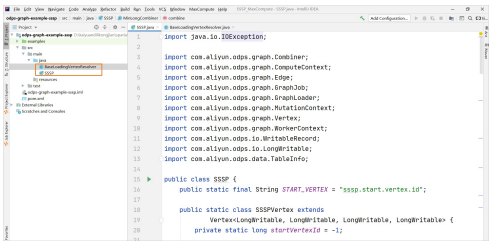
   > 说明 本示例将sssp.txt文件保存在MaxCompute客户端 `bin` 目录下，实际操作中需要注意sssp.txt文件的保存路径。

3. 编写SSSP示例。

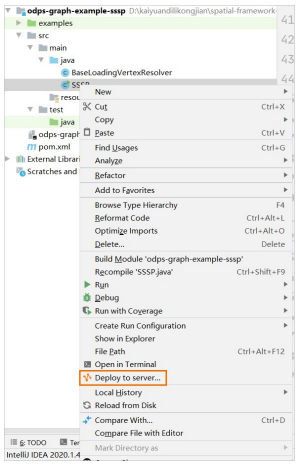
   i. 在IntelliJ IDEA中，创建**MaxCompute Java Module**，并命名为odps-graph-example-sssp。

   > 说明 创建MaxCompute Java Module。详情请参见创建MaxCompute Java Module。

   ii. 在新建的**odps-graph-example-sssp**中，创建 `BaseLoadingVertexResolver` 类和 `SSSP` 类。代码详情，请参见单源最短距离有向图。



iii. 在IntelliJ IDEA中，通过MaxCompute Studio完成一键式打包。操作详情，请参见打包、上传及注册。



> **说明** 本示例将部署到MaxCompute项目的JAR包命名为odps-graph-example-sssp.jar。

4. 在MaxCompute客户端，执行如下命令运行SSSP。

```
jar -libjars odps-graph-example-sssp.jar -classpath <LOCAL_JAR_PATH>/odps-graph-example-sssp.jar SSSP 1 sssp_in sssp_out;
```

LOCAL_JAR_PATH：odps-graph-example-sssp.jar的本地路径。

运行结果。

```
vertex    value
1         0
2         2
3         1
4         3
5         2
```

- vertex：代表当前顶点。
- value：代表当前vertex到达源点（1）的最短距离。

> **说明** 如果您需要使用Graph功能，直接提交图计算作业即可。

12.6. SDK配置说明

本文为您介绍几种不同版本的Java SDK的相关配置信息。

如果您使用Maven，可以从Maven库中搜索odps-sdk-graph获取不同版本的Java SDK。相关配置信息，如下所示。

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-graph</artifactId>
    <version>0.20.7</version>
</dependency>
```

主要接口	说明
GraphJob	GraphJob继承自JobConf，用于定义、提交和管理一个MaxCompute Graph作业。
Vertex	Vertex是图的点的抽象，包含属性：id、value、halted、edges。通过GraphJob的setVertexClass接口提供Vertex实现。
Edge	Edge是图的边的抽象，包含属性：destVertexId、value。图数据结构采用邻接表，点的出边保存在点的edges中。
GraphLoader	GraphLoader用于载入图。通过GraphJob的setGraphLoaderClass接口提供GraphLoader实现。
VertexResolver	VertexResolver用于自定义图拓扑修改时的冲突处理逻辑。通过GraphJob的setLoadingVertexResolverClass和setComputingVertexResolverClass接口，提供图加载和迭代计算过程中的图拓扑修改的冲突处理逻辑。
Partitioner	Partitioner用于对图进行划分，使得计算可以分片进行。通过GraphJob的接口提供Partitioner实现，默认采用HashPartitioner，即对点ID求哈希值然后对Worker数目取模。

主要接口	说明
WorkerComputer	WorkerComputer允许在Worker开始和退出时执行用户自定义的逻辑。通过GraphJob的setWorkerComputerClass接口提供WorkerComputer实现。
Aggregator	`Aggregator`的`setAggregatorClass(Class...)` 定义一个或多个Aggregator。
Combiner	Combiner的setCombinerClass设置Combiner。
Counters	计数器。在作业运行逻辑中，可以通过WorkerContext接口取得计数器并进行计数，框架会自动进行汇总。
WorkerContext	上下文对象，封装了框架提供的功能。例如，修改图拓扑结构、发送消息、写结果、读取资源等。

12.7. 开发和调试

MaxCompute不提供Graph开发插件，您可以基于Eclipse开发MaxCompute Graph程序。

开发流程如下：

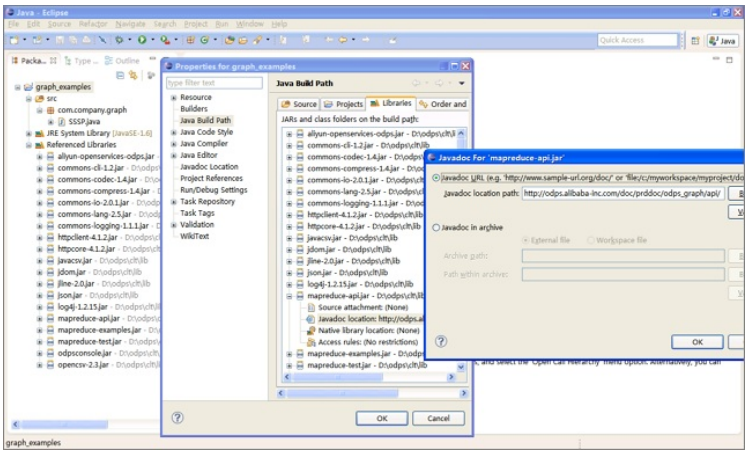
1. 编写Graph代码，使用本地调试进行基本的测试。
2. 进行集群调试，验证结果。

开发示例

本节以SSSP算法为例，为您介绍如何使用Eclipse开发和调试Graph程序。

操作步骤：

1. 创建Java工程，工程名为*graph_examples*。
2. 将MaxCompute客户端*lib*目录下的Jar包加到Eclipse工程的**Java Build Path**中。



3. 开发MaxCompute Graph程序。

   实际开发过程中，通常会先复制一个例子（例如单源最短距离），然后进行修改。在本示例中，仅修改了Package的路径为*package com.aliyun.odps.graph.example*。

4. 编译打包。

   在Eclipse环境中，右键单击源代码目录（图中的*src*目录），**Export > Java > JAR file**生成Jar包，选择目标Jar包的保存路径，例如*D:\\odps\\clt\\odps-graph-example-sssp.jar*。

5. 使用MaxCompute客户端运行SSSP，详情请参见运行Graph。

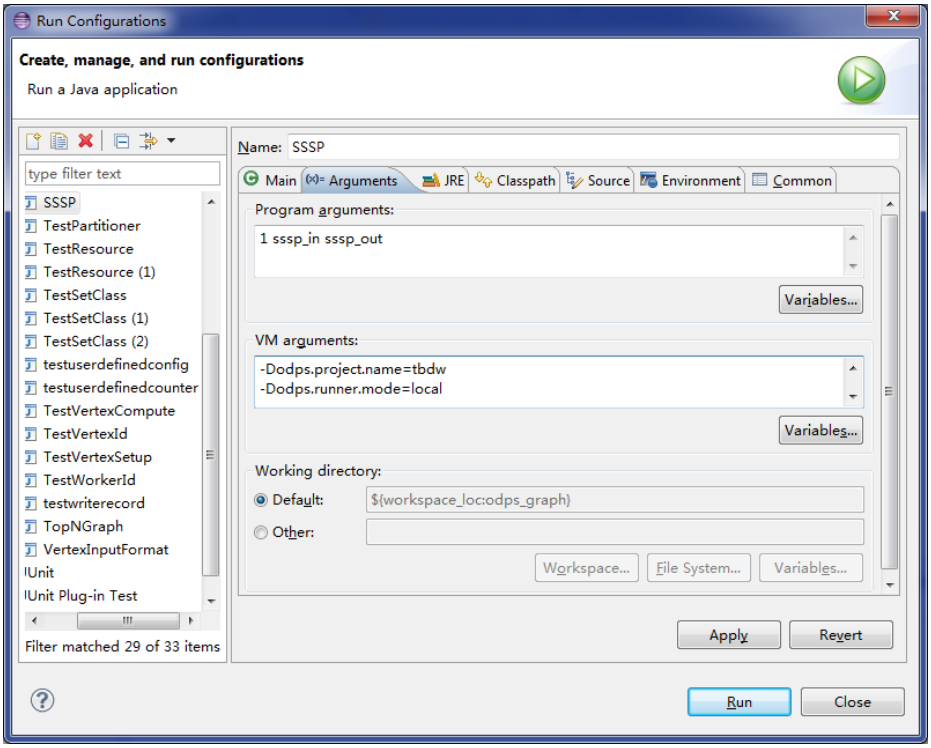
本地调试

MaxCompute Graph支持本地调试模式，您可以使用Eclipse进行断点调试。

操作步骤：

1. 下载一个*odps-graph-local*的Maven包。
2. 选择Eclipse工程，右键单击Graph作业主程序（包含`main`函数）文件，配置其运行参数（**Run As > Run Configurations**）。
3. 在**Arguments**页签中，设置Program arguments参数为*1 sssp_in sssp_out*，作为主程序的输入参数。
4. 在**Arguments**页签中，设置VM arguments参数如下。

```
-Dodps.runner.mode=local
-Dodps.project.name=<project.name>
-Dodps.end.point=<end.point>
-Dodps.access.id=<access.id>
-Dodps.access.key=<access.key>
```



5. 对于本地模式（即不指定odps.end.point参数），需要在*warehouse*创建*sssp_in*、*sssp_out*表，为输入表*sssp_in*添加数据，输入数据如下所示。

```
1,"2:2,3:1,4:4"
2,"1:2,3:2,4:1"
3,"1:1,2:2,5:1"
4,"1:4,2:1,5:1"
5,"3:1,4:1"
```

关于*warehouse*的介绍请参见本地运行。

6. 单击**Run**，即可本地跑SSSP。

> **说明** 参数设置可参见MaxCompute客户端中*conf/odps_config.ini*的设置，上述为几个常用参数，其他参数说明如下：
>
> - odps.runner.mode：取值为*local*，本地调试功能必须指定。
> - odps.project.name：指定当前Project，必须指定。
> - odps.end.point：指定当前MaxCompute服务的地址，可以不指定。如果不指定，只从*warehouse*读取表或资源的meta和数据，不存在则抛异常。如果指定，会先从*warehouse*读取，不存在时会远程连接MaxCompute读取。
> - odps.access.id：连接MaxCompute服务的AccessKey ID，只在指定odps.end.point时有效。
> - odps.access.key：连接MaxCompute服务的AccessKey Secret，只在指定odps.end.point时有效。
> - odps.cache.resources：指定使用的资源列表，效果与Jar命令的 `-resources` 相同。
> - odps.local.warehouse：本地*warehouse*路径，不指定时默认为*./warehouse*。

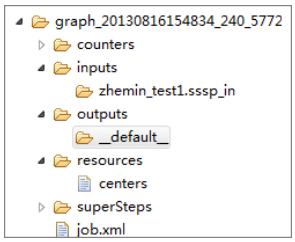
在Eclipse中本地运行SSSP的调试输出信息，如下所示。

```
Counters: 3
        com.aliyun.odps.graph.local.COUNTER
                TASK_INPUT_BYTE=211
                TASK_INPUT_RECORD=5
                TASK_OUTPUT_BYTE=161
                TASK_OUTPUT_RECORD=5
 graph task finish
```

> **说明** 上述示例中，需要本地*warehouse*下有*sssp_in*及*sssp_out*表。*sssp_in*及*sssp_out*的详细信息请参见编写Graph。

本地作业临时目录

每运行一次本地调试，都会在Eclipse工程目录下新建一个临时目录。



本地运行的Graph作业，临时目录包括以下几个目录和文件：

- *counters*：存放作业运行的一些计数信息。
- *inputs*：存放作业的输入数据，优先取自本地的*warehouse*，如果本地没有，会通过MaxCompute SDK从服务端读取（如果设置了odps.end.point），默认一个*input*只读10条数据，可以通过 `-Dodps.mapred.local.record.limit` 参数进行修改，但是也不能超过1万条记录。
- *outputs*：存放作业的输出数据，如果本地*warehouse*中存在输出表，*outputs*中的结果数据在作业执行完后会覆盖本地*warehouse*中对应的表。
- *resources*：存放作业使用的资源，与输入类似，优先取自本地的*warehouse*，如果本地没有，会通过MaxCompute SDK从服务端读取（如果设置了odps.end.point）。
- *job.xml*：作业配置。
- *superstep*：存放每一轮迭代的持久化信息。

> 说明 如果需要本地调试时输出详细日志，需要在*src*目录下放一个*log4j*的配置文件*log4j.properties_odps_graph_cluster_debug*。

集群调试

通过本地的调试后，您即可提交作业到集群进行测试。

操作步骤如下：

1. 配置MaxCompute客户端。
2. 使用 `add jar /path/work.jar -f;` 命令更新Jar包。
3. 使用Jar命令运行作业，查看运行日志和结果数据。

> 说明 集群运行Graph的详细介绍请参见编写Graph。

性能调优

对性能有所影响的Graph Job配置项如下：

- `setSplitSize(long)`：输入表切分大小，单位MB，大于0，默认64。
- `setNumWorkers(int)`：设置作业Worker数量，范围为[1，1000]，默认值为1。Worker数由作业输入字节数和 `splitSize` 决定。
- `setWorkerCPU(int)`：Map CPU资源，100为1CPU核，[50，800]之间，默认值为200。
- `setWorkerMemory(int)`：Map内存资源，单位MB，范围为[256，12288]，默认值为4096。
- `setMaxIteration(int)`：设置最大迭代次数，默认为-1，小于或等于0时表示最大迭代次数不作为作业终止条件。
- `setJobPriority(int)`：设置作业优先级，范围为[0，9]，默认值为9，数值越大优先级越小。

通常情况下的调优建议：

- 可以考虑使用 `setNumWorkers` 方法增加Worker数目。
- 可以考虑使用 `setSplitSize` 方法减少切分大小，提高作业载入数据速度。
- 加大Worker的CPU或内存。
- 设置最大迭代次数，如果有些应用结果精度要求不高，可以考虑减少迭代次数，尽快结束。

接口 `setNumWorkers` 与 `setSplitSize` 配合使用，可以提高数据的载入速度。假设 `setNumWorkers` 为 `workerNum`，`setSplitSize` 为 `splitSize`，总输入字节数为 `inputSize`，则输入被切分后的块数 `splitNum=inputSize/splitSize`。`workerNum` 和 `splitNum` 之间的关系如下：

- 情况一：`splitNum` = `workerNum`，每个Worker负责载入1个Split。
- 情况二：`splitNum` > `workerNum`，每个Worker负责载入1个或多个Split。
- 情况三：`splitNum` < `workerNum`，每个Worker负责载入0个或1一个Split。

因此，应调节 `workerNum` 和 `splitSize`，在满足前两种情况时，数据载入比较快。迭代阶段只需调节 `workerNum`。如果设置 `runtime partitioning` 为False，则建议直接使用 `setSplitSize` 控制Worker数量，或者保证满足前两种情况。当出现第三种情况时，部分Worker上的切片会为0，可以在Jar命令前使用 `set odps.graph.split.size=<m>; set odps.graph.worker.num=<n>;` 与 `setNumWorkers`，和 `setSplitSize` 等效。

另外一种常见的性能问题为数据倾斜，反应到Counters就是某些Worker处理的点或边数量远超过其他Worker。数据倾斜的原因通常是某些Key对应的点、边，或者消息的数量远超过其他Key，这些Key被分到少量的Worker处理，导致这些Worker运行时间较长。解决方法：

- 尝试Combiner，将这些Key对应点的消息进行本地聚合，减少消息发生。

  开发人员可定义Combiner以减少存储消息的内存和网络数据流量，缩短作业的执行时间。

- 改进业务逻辑。

  数据量大时，读取磁盘中的数据可能耗费一部分处理时间。减少需要读取的数据字节数可以提高总体的吞吐量，提高作业性能。您可通过以下方法进行改进：

  - 减少输入数据量：对某些决策性质的应用，处理数据采样后子集所得到的结果只可能影响结果的精度，而并不会影响整体的准确性，因此可以考虑先对数据进行特定采样后再导入输入表中进行处理。
  - 避免读取用不到的字段：MaxCompute Graph框架的TableInfo类支持读取指定的列（以列名数组方式传入），而非整个表或表分区，这样也可以减少输入的数据量，提高作业性能。

内置Jar包

以下Jar包会默认加载到运行Graph程序的JVM中，您不必上传这些资源，也不必在命令行的 `-libjars` 带上这些Jar包。

- *commons-codec-1.3.jar*
- *commons-io-2.0.1.jar*
- *commons-lang-2.5.jar*
- *commons-logging-1.0.4.jar*
- *commons-logging-api-1.0.4.jar*
- *guava-14.0.jar*
- *json.jar*
- *log4j-1.2.15.jar*
- *slf4j-api-1.4.3.jar*
- *slf4j-log4j12-1.4.3.jar*
- *xmlenc-0.52.jar*

> **说明** 在JVM的Classpath中，上述内置Jar包会位于您的Jar包之前，可能产生版本冲突。例如您的程序中使用了*commons-codec-1.5.jar*某个类的函数，但是这个函数不在*commons-codec-1.3.jar*中，这时如果1.3版本无法实现您的需求，则只能等待MaxCompute升级新版本。

12.8. 示例程序

12.8.1. 单源最短距离

单源最短距离是指给定图中一个源点，计算源点到其它所有节点的最短距离。Dijkstra算法是求解有向图中单源最短距离SSSP（Single Source Shortest Path）的经典算法。

算法原理

Dijkstra算法是通过**点**去更新**最短距离值**，每个**点**维护到**源点**的当前**最短距离值**，当这个值发生变化时，将新值加上**边**的**权值**，发送消息通知其**邻接点**。下一轮迭代时，**邻接点**根据收到的消息，更新其当前**最短距离值**，当所有**点**的当前**最短距离值**不再变化时，迭代结束。

- 初始化：源点s到s自身的距离为0（ `d[s]=0` ），其他点u到s的距离为无穷（ `d[u]=∞` ）。
- 迭代：如果存在一条从u到v的边，则从s到v的最短距离更新为 `d[v]=min(d[v], d[u]+weight(u, v))` ，直到所有的点到s的距离不再发生变化时，迭代结束。

> **说明** 对一个有权重的有向图 `G=(V,E)` ，从一个源点s到汇点v有很多路径，其中边权和最小的路径，称为从s到v的最短距离。

由算法基本原理可以看出，此算法非常适合用MaxCompute Graph程序进行求解。

使用场景

图类型通常分为有向图和无向图两种，MaxCompute均支持。基于源数据的分布，构造有向图和无向图时的路径通路会存在差异，求解SSSP时会产生不同的结果。MaxCompute Graph以有向图为基础数据模型，框架内会以有向图的模型参与计算。

代码示例

以下代码基于不同的场景，提供不同的代码示例。

- 有向图

- 定义类 `BaseLoadingVertexResolver` ，此异常类会在 `SSSP` 类中被引用。

```
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.LoadingVertexResolver;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableComparable;
import java.io.IOException;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
@SuppressWarnings("rawtypes")
public class BaseLoadingVertexResolver<I extends WritableComparable, V extends Writable, E extends Writable, M extends Writa
ble>
        extends LoadingVertexResolver<I, V, E, M> {
  @Override
  public Vertex<I, V, E, M> resolve(I vertexId, VertexChanges<I, V, E, M> vertexChanges) throws IOException {
    Vertex<I, V, E, M> vertex = addVertexIfDesired(vertexId, vertexChanges);
    if (vertex != null) {
      addEdges(vertex, vertexChanges);
    } else {
      System.err.println("Ignore all addEdgeRequests for vertex#" + vertexId);
    }
    return vertex;
  }
  protected Vertex<I, V, E, M> addVertexIfDesired(
          I vertexId,
          VertexChanges<I, V, E, M> vertexChanges) {
    Vertex<I, V, E, M> vertex = null;
    if (hasVertexAdditions(vertexChanges)) {
      vertex = vertexChanges.getAddedVertexList().get(0);
    }
    return vertex;
  }
  protected void addEdges(Vertex<I, V, E, M> vertex,
                          VertexChanges<I, V, E, M> vertexChanges) throws IOException {
    Set<I> destVertexId = new HashSet<I>();
    if (vertex.hasEdges()) {
      List<Edge<I, E>> edgeList = vertex.getEdges();
      for (Iterator<Edge<I, E>> edges = edgeList.iterator(); edges.hasNext(); ) {
        Edge<I, E> edge = edges.next();
        if (destVertexId.contains(edge.getDestVertexId())) {
          edges.remove();
        } else {
          destVertexId.add(edge.getDestVertexId());
        }
      }
    }
    for (Vertex<I, V, E, M> vertex1 : vertexChanges.getAddedVertexList()) {
      if (vertex1.hasEdges()) {
        List<Edge<I, E>> edgeList = vertex1.getEdges();
        for (Edge<I, E> edge : edgeList) {
          if (destVertexId.contains(edge.getDestVertexId())) continue;
          destVertexId.add(edge.getDestVertexId());
          vertex.addEdge(edge.getDestVertexId(), edge.getValue());
        }
      }
    }
  }
  protected boolean hasVertexAdditions(VertexChanges<I, V, E, M> changes) {
    return changes != null && changes.getAddedVertexList() != null
            && !changes.getAddedVertexList().isEmpty();
  }
}
```

代码说明：

- 第15行：定义BaseLoadingVertexResolver。用于处理有向图数据在加载过程中所遇到的冲突。
- 第18行：resolve为处理冲突的具体方法。例如当前的某一顶点进行了两次添加的过程（即进行了两次addVertexRequest操作），这种行为便称为冲突加载，需要人为处理冲突之后再参与计算。

- 定义类 `SSSP` 。

```
import java.io.IOException;
```

```
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.data.TableInfo;
public class SSSP {
  public static final String START_VERTEX = "sssp.start.vertex.id";
  public static class SSSPVertex extends
          Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
    private static long startVertexId = -1;
    public SSSPVertex() {
      this.setValue(new LongWritable(Long.MAX_VALUE));
    }
    public boolean isStartVertex(
            ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context) {
      if (startVertexId == -1) {
        String s = context.getConfiguration().get(START_VERTEX);
        startVertexId = Long.parseLong(s);
      }
      return getId().get() == startVertexId;
    }
    @Override
    public void compute(
            ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
            Iterable<LongWritable> messages) throws IOException {
      long minDist = isStartVertex(context) ? 0 : Long.MAX_VALUE;
      for (LongWritable msg : messages) {
        if (msg.get() < minDist) {
          minDist = msg.get();
        }
      }
      if (minDist < this.getValue().get()) {
        this.setValue(new LongWritable(minDist));
        if (hasEdges()) {
          for (Edge<LongWritable, LongWritable> e : this.getEdges()) {
            context.sendMessage(e.getDestVertexId(), new LongWritable(minDist + e.getValue().get()));
          }
        }
      } else {
        voteToHalt();
      }
    }
    @Override
    public void cleanup(
            WorkerContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
            throws IOException {
      context.write(getId(), getValue());
    }
    @Override
    public String toString() {
      return "Vertex(id=" + this.getId() + ",value=" + this.getValue() + ",#edges=" + this.getEdges() + ")";
    }
  }
  public static class SSSPGraphLoader extends
          GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
    @Override
    public void load(
            LongWritable recordNum,
            WritableRecord record,
            MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
            throws IOException {
      SSSPVertex vertex = new SSSPVertex();
      vertex.setId((LongWritable) record.get(0));
      String[] edges = record.get(1).toString().split(",");
      for (String edge : edges) {
        String[] ss = edge.split(":");
        vertex.addEdge(new LongWritable(Long.parseLong(ss[0])), new LongWritable(Long.parseLong(ss[1])));
      }
      context.addVertexRequest(vertex);
```

```
    }
  }
  public static class MinLongCombiner extends
          Combiner<LongWritable, LongWritable> {
    @Override
    public void combine(LongWritable vertexId, LongWritable combinedMessage,
                        LongWritable messageToCombine) throws IOException {
      if (combinedMessage.get() > messageToCombine.get()) {
        combinedMessage.set(messageToCombine.get());
      }
    }
  }
  public static void main(String[] args) throws IOException {
    if (args.length < 3) {
      System.out.println("Usage: <startnode> <input> <output>");
      System.exit(-1);
    }
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(SSSPGraphLoader.class);
    job.setVertexClass(SSSPVertex.class);
    job.setCombinerClass(MinLongCombiner.class);
    job.setLoadingVertexResolver(BaseLoadingVertexResolver.class);
    job.set(START_VERTEX, args[0]);
    job.addInput(TableInfo.builder().tableName(args[1]).build());
    job.addOutput(TableInfo.builder().tableName(args[2]).build());
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
            + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
  }
}
```

代码说明：

- 第19行：定义SSSPVertex。其中：
  - 点值表示该顶点到源点startVertexId的最短距离。
  - compute()方法使用迭代公式 `d[v]=min(d[v], d[u]+weight(u, v))` 计算最短距离值并更新至当前点值。
  - cleanup()方法将当前顶点到源点的最短距离写入结果表中。
- 第54行：当前顶点的**Value**值（即该顶点到源点的最短路径）未发生变化时，即调用voteToHalt()通过框架使该顶点进入**halt**状态。当所有顶点都进入**halt**状态时，计算结束。
- 第71行：定义GraphLoader图数据以有向图的方式加载图数据。通过将表内存储的记录解析为图的顶点或边信息加载至框架内。如上示例代码中，用户可通过addVertexRequest方式将图的顶点信息加载至图计算的上下文内。
- 第90行：定义MinLongCombiner。对发送给同一个点的消息进行合并，优化性能，减少内存占用。
- 第101行：主程序main函数中定义GraphJob。指定Vertex、GraphLoader、BaseLoadingVertexResolver、Combiner等的实现，指定输入输出表。
- 第110行：添加处理冲突的类BaseLoadingVertexResolver。

- 无向图

```
import com.alibaba.odps.graph.impl.SSSPNullComputingVertexResolver;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.*;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableRecord;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
public class SSSPBenchmark4 {
    public static final String START_VERTEX = "sssp.start.vertex.id";
    public static class SSSPVertex extends
            Vertex<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> {
        private static long startVertexId = -1;
        public SSSPVertex() {
            this.setValue(new DoubleWritable(Double.MAX_VALUE));
        }
        public boolean isStartVertex(
                ComputeContext<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> context) {
            if (startVertexId == -1) {
                String s = context.getConfiguration().get(START_VERTEX);
                startVertexId = Long.parseLong(s);
            }
```

```
            return getId().get() == startVertexId;
        }
        @Override
        public void compute(
                ComputeContext<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> context,
                Iterable<DoubleWritable> messages) throws IOException {
            double minDist = isStartVertex(context) ? 0 : Double.MAX_VALUE;
            for (DoubleWritable msg : messages) {
                if (msg.get() < minDist) {
                    minDist = msg.get();
                }
            }
            if (minDist < this.getValue().get()) {
                this.setValue(new DoubleWritable(minDist));
                if (hasEdges()) {
                    for (Edge<LongWritable, DoubleWritable> e : this.getEdges()) {
                        context.sendMessage(e.getDestVertexId(), new DoubleWritable(minDist
                                + e.getValue().get()));
                    }
                }
            } else {
                voteToHalt();
            }
        }
        @Override
        public void cleanup(
                WorkerContext<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> context)
                throws IOException {
            context.write(getId(), getValue());
        }
    }
    public static class MinLongCombiner extends
            Combiner<LongWritable, DoubleWritable> {
        @Override
        public void combine(LongWritable vertexId, DoubleWritable combinedMessage,
                            DoubleWritable messageToCombine) {
            if (combinedMessage.get() > messageToCombine.get()) {
                combinedMessage.set(messageToCombine.get());
            }
        }
    }
    public static class SSSPGraphLoader extends
            GraphLoader<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> {
        @Override
        public void load(
                LongWritable recordNum,
                WritableRecord record,
                MutationContext<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> context)
                throws IOException {
            LongWritable sourceVertexID = (LongWritable) record.get(0);
            LongWritable destinationVertexID = (LongWritable) record.get(1);
            DoubleWritable edgeValue = (DoubleWritable) record.get(2);
            Edge<LongWritable, DoubleWritable> edge = new Edge<LongWritable, DoubleWritable>(destinationVertexID, edgeValue);
            context.addEdgeRequest(sourceVertexID, edge);
            Edge<LongWritable, DoubleWritable> edge2 = new
                    Edge<LongWritable, DoubleWritable>(sourceVertexID, edgeValue);
            context.addEdgeRequest(destinationVertexID, edge2);
        }
    }
    public static class SSSPLoadingVertexResolver extends
            LoadingVertexResolver<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> {
        @Override
        public Vertex<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> resolve(
                LongWritable vertexId,
                VertexChanges<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> vertexChanges) throws IOException
{
            SSSPVertex computeVertex = new SSSPVertex();
            computeVertex.setId(vertexId);
            Set<LongWritable> destinationVertexIDSet = new HashSet<>();
            if (hasEdgeAdditions(vertexChanges)) {
                for (Edge<LongWritable, DoubleWritable> edge : vertexChanges.getAddedEdgeList()) {
                    if (!destinationVertexIDSet.contains(edge.getDestVertexId())) {
                        destinationVertexIDSet.add(edge.getDestVertexId());
                        computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
                    }
```

```
                }
            }
            return computeVertex;
        }
        protected boolean hasEdgeAdditions(VertexChanges<LongWritable, DoubleWritable, DoubleWritable, DoubleWritable> changes
) {
            return changes != null && changes.getAddedEdgeList() != null
                    && !changes.getAddedEdgeList().isEmpty();
        }
    }
    public static void main(String[] args) throws IOException {
        if (args.length < 2) {
            System.out.println("Usage: <startnode> <input> <output>");
            System.exit(-1);
        }
        GraphJob job = new GraphJob();
        job.setGraphLoaderClass(SSSPGraphLoader.class);
        job.setLoadingVertexResolver(SSSPLoadingVertexResolver.class);
        job.setVertexClass(SSSPVertex.class);
        job.setCombinerClass(MinLongCombiner.class);
        job.set(START_VERTEX, args[0]);
        job.addInput(TableInfo.builder().tableName(args[1]).build());
        job.addOutput(TableInfo.builder().tableName(args[2]).build());
        long startTime = System.currentTimeMillis();
        job.run();
        System.out.println("Job Finished in "
                + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
    }
}
```

代码说明：

- 第15行：定义SSSPVertex。其中：
  - 点值表示该顶点到源点startVertexId的最短距离。
  - compute()方法使用迭代公式 `d[v]=min(d[v], d[u]+weight(u, v))` 计算最短距离值并更新至当前点值。
  - cleanup()方法将当前顶点到源点的最短距离写入结果表中。
- 第54行：当前顶点的**Value**值（即该顶点到源点的最短路径）未发生变化时，即调用voteToHalt()通过框架使该顶点进入**halt**状态。当所有顶点都进入**halt**状态时，计算结束。
- 第61行：定义MinLongCombiner。对发送给同一个点的消息进行合并，优化性能，减少内存占用。
- 第72行：定义GraphLoader图数据以无向图的方式加载图数据。通过addEdgeRequest以两点之间的边作为无向边加载边信息，这样便可保证当前表内存储的图信息加载成无向图。
  - 第80行：第一列表示初始点的ID。
  - 第81行：第二列表示终点的ID。
  - 第82行：第三列表示边的权重。
  - 第83行：创建边，由终点ID和边的权重组成。
  - 第84行：请求给初始点添加边。
  - 第85行 - 第87行：每条Record表示双向边，重复第83行与第84行。
- 定义SSSPLoadingVertexResolver。用于处理无向图数据在加载过程中所遇到的冲突。例如当前的某一边进行了两次添加的过程（即进行了两次addEdgeRequest操作），这种行为便称为冲突加载，需要人为处理重复添加的边才可保证计算正确性。
- 第101行：主程序main函数中定义GraphJob。指定Vertex、GraphLoader、SSSPLoadingVertexResolver、Combiner等的实现，指定输入输出表。

运行结果

以下是基于有向图的代码示例的运行结果。操作详情，请参见编写Graph。

```
vertex    value
1         0
2         2
3         1
4         3
5         2
```

- vertex：代表当前顶点。
- value：代表当前vertex到达源点（1）的最短距离。

> **说明** 无向图数据，用户可以参考无向图代码示例中的初始点ID，终点ID，边的权值自行构造。

12.8.2. PageRank

PageRank算法是计算网页排名的经典算法。输入是一个有向图G，其中顶点表示网页。如果存在网页A到网页B的链接，则存在连接A到B的边。

算法的基本原理如下：

- 初始化：点值表示PageRank的rank值（DOUBLE类型）。初始时，所有点取值为 `1/TotalNumVertices` 。
- 迭代公式： `PageRank(i)=0.15/TotalNumVertices+0.85*sum` 。其中sum为所有指向i点的点（设为j） `PageRank(j)/out_degree(j)` 的累加值。

由算法基本原理可以看出，此算法非常适合使用MaxCompute Graph程序进行求解。每个点j维护其PageRank值，每一轮迭代都将 `PageRank(j)/out_degree(j)` 发给其邻接点（向其投票）。下一轮迭代时，每个点根据迭代公式重新计算PageRank取值。

代码示例

```
import java.io.IOException;
import org.apache.log4j.Logger;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
public class PageRank {
  private final static Logger LOG = Logger.getLogger(PageRank.class);
  public static class PageRankVertex extends
      Vertex<Text, DoubleWritable, NullWritable, DoubleWritable> {
    @Override
    public void compute(
        ComputeContext<Text, DoubleWritable, NullWritable, DoubleWritable> context,
        Iterable<DoubleWritable> messages) throws IOException {
      if (context.getSuperstep() == 0) {
        setValue(new DoubleWritable(1.0 / context.getTotalNumVertices()));
      } else if (context.getSuperstep() >= 1) {
        double sum = 0;
        for (DoubleWritable msg : messages) {
          sum += msg.get();
        }
        DoubleWritable vertexValue = new DoubleWritable(
            (0.15f / context.getTotalNumVertices()) + 0.85f * sum);
        setValue(vertexValue);
      }
      if (hasEdges()) {
        context.sendMessageToNeighbors(this, new DoubleWritable(getValue()
            .get() / getEdges().size()));
      }
    }
    @Override
    public void cleanup(
        WorkerContext<Text, DoubleWritable, NullWritable, DoubleWritable> context)
        throws IOException {
      context.write(getId(), getValue());
    }
  }
  public static class PageRankVertexReader extends
      GraphLoader<Text, DoubleWritable, NullWritable, DoubleWritable> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<Text, DoubleWritable, NullWritable, DoubleWritable> context)
        throws IOException {
      PageRankVertex vertex = new PageRankVertex();
      vertex.setValue(new DoubleWritable(0));
      vertex.setId((Text) record.get(0));
      System.out.println(record.get(0));
      for (int i = 1; i < record.size(); i++) {
        Writable edge = record.get(i);
        System.out.println(edge.toString());
```

```
        if (!(edge.equals(NullWritable.get()))) {
          vertex.addEdge(new Text(edge.toString()), NullWritable.get());
        }
      }
      LOG.info("vertex edgs size: "
          + (vertex.hasEdges() ? vertex.getEdges().size() : 0));
      context.addVertexRequest(vertex);
    }
  }
  private static void printUsage() {
    System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
    System.exit(-1);
  }
  public static void main(String[] args) throws IOException {
    if (args.length < 2)
      printUsage();
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(PageRankVertexReader.class);
    job.setVertexClass(PageRankVertex.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
      job.setMaxIteration(Integer.parseInt(args[2]));
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
  }
}
```

上述代码，说明如下：

- 第 23 行：定义PageRankVertex类，其中：
  - 点值表示该点（网页）的当前PageRank取值。
  - compute()方法使用迭代公式：`PageRank(i)=0.15/TotalNumVertices+0.85*sum`更新点值。
  - cleanup()方法把点及其PageRank取值写到结果表中。
- 第 55 行：定义PageRankVertexReader类，加载图，将表中每一条记录解析为一个点，记录的第一列是起点，其他列为终点。
- 第 88 行：主程序（main函数），定义GraphJob，指定Vertex/GraphLoader等的实现，以及最大迭代次数（默认30），并指定输入输出表。

12.8.3. K-均值聚类

k-均值聚类（Kmeans）算法是非常基础且被大量使用的聚类算法。

算法基本原理：以空间中k个点为中心进行聚类，对最靠近它们的点进行归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。

将样本集分为k个类别的算法描述如下：

1. 适当选择k个类的初始中心。
2. 在第i次迭代中，对任意一个样本，求其到k个中心的距离，将该样本归到距离最短的中心所在的类。
3. 利用均值等方法更新该类的中心值。
4. 对于所有的k个聚类中心，如果利用上两步的迭代法更新后，值保持不变或者小于某个阈值，则迭代结束，否则继续迭代。

代码示例

K-均值聚类算法的代码，如下所示。

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.log4j.Logger;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
```

```
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
public class Kmeans {
  private final static Logger LOG = Logger.getLogger(Kmeans.class);
  public static class KmeansVertex extends
      Vertex<Text, Tuple, NullWritable, NullWritable> {
    @Override
    public void compute(
        ComputeContext<Text, Tuple, NullWritable, NullWritable> context,
        Iterable<NullWritable> messages) throws IOException {
      context.aggregate(getValue());
    }
  }
  public static class KmeansVertexReader extends
      GraphLoader<Text, Tuple, NullWritable, NullWritable> {
    @Override
    public void load(LongWritable recordNum, WritableRecord record,
        MutationContext<Text, Tuple, NullWritable, NullWritable> context)
        throws IOException {
      KmeansVertex vertex = new KmeansVertex();
      vertex.setId(new Text(String.valueOf(recordNum.get())));
      vertex.setValue(new Tuple(record.getAll()));
      context.addVertexRequest(vertex);
    }
  }
  public static class KmeansAggrValue implements Writable {
    Tuple centers = new Tuple();
    Tuple sums = new Tuple();
    Tuple counts = new Tuple();
    @Override
    public void write(DataOutput out) throws IOException {
      centers.write(out);
      sums.write(out);
      counts.write(out);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
      centers = new Tuple();
      centers.readFields(in);
      sums = new Tuple();
      sums.readFields(in);
      counts = new Tuple();
      counts.readFields(in);
    }
    @Override
    public String toString() {
      return "centers " + centers.toString() + ", sums " + sums.toString()
          + ", counts " + counts.toString();
    }
  }
  public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {
    @SuppressWarnings("rawtypes")
    @Override
    public KmeansAggrValue createInitialValue(WorkerContext context)
        throws IOException {
      KmeansAggrValue aggrVal = null;
      if (context.getSuperstep() == 0) {
        aggrVal = new KmeansAggrValue();
        aggrVal.centers = new Tuple();
        aggrVal.sums = new Tuple();
        aggrVal.counts = new Tuple();
        byte[] centers = context.readCacheFile("centers");
        String lines[] = new String(centers).split("\n");
        for (int i = 0; i < lines.length; i++) {
          String[] ss = lines[i].split(",");
          Tuple center = new Tuple();
          Tuple sum = new Tuple();
          for (int j = 0; j < ss.length; ++j) {
            center.append(new DoubleWritable(Double.valueOf(ss[j].trim())));
            sum.append(new DoubleWritable(0.0));
          }
```

```
        LongWritable count = new LongWritable(0);
        aggrVal.sums.append(sum);
        aggrVal.counts.append(count);
        aggrVal.centers.append(center);
      }
    } else {
      aggrVal = (KmeansAggrValue) context.getLastAggregatedValue(0);
    }
    return aggrVal;
  }
  @Override
  public void aggregate(KmeansAggrValue value, Object item) {
    int min = 0;
    double mindist = Double.MAX_VALUE;
    Tuple point = (Tuple) item;
    for (int i = 0; i < value.centers.size(); i++) {
      Tuple center = (Tuple) value.centers.get(i);
      // use Euclidean Distance, no need to calculate sqrt
      double dist = 0.0d;
      for (int j = 0; j < center.size(); j++) {
        double v = ((DoubleWritable) point.get(j)).get()
            - ((DoubleWritable) center.get(j)).get();
        dist += v * v;
      }
      if (dist < mindist) {
        mindist = dist;
        min = i;
      }
    }
    // update sum and count
    Tuple sum = (Tuple) value.sums.get(min);
    for (int i = 0; i < point.size(); i++) {
      DoubleWritable s = (DoubleWritable) sum.get(i);
      s.set(s.get() + ((DoubleWritable) point.get(i)).get());
    }
    LongWritable count = (LongWritable) value.counts.get(min);
    count.set(count.get() + 1);
  }
  @Override
  public void merge(KmeansAggrValue value, KmeansAggrValue partial) {
    for (int i = 0; i < value.sums.size(); i++) {
      Tuple sum = (Tuple) value.sums.get(i);
      Tuple that = (Tuple) partial.sums.get(i);
      for (int j = 0; j < sum.size(); j++) {
        DoubleWritable s = (DoubleWritable) sum.get(j);
        s.set(s.get() + ((DoubleWritable) that.get(j)).get());
      }
    }
    for (int i = 0; i < value.counts.size(); i++) {
      LongWritable count = (LongWritable) value.counts.get(i);
      count.set(count.get() + ((LongWritable) partial.counts.get(i)).get());
    }
  }
  @SuppressWarnings("rawtypes")
  @Override
  public boolean terminate(WorkerContext context, KmeansAggrValue value)
      throws IOException {
    // compute new centers
    Tuple newCenters = new Tuple(value.sums.size());
    for (int i = 0; i < value.sums.size(); i++) {
      Tuple sum = (Tuple) value.sums.get(i);
      Tuple newCenter = new Tuple(sum.size());
      LongWritable c = (LongWritable) value.counts.get(i);
      for (int j = 0; j < sum.size(); j++) {
        DoubleWritable s = (DoubleWritable) sum.get(j);
        double val = s.get() / c.get();
        newCenter.set(j, new DoubleWritable(val));
        // reset sum for next iteration
        s.set(0.0d);
      }
      // reset count for next iteration
      c.set(0);
      newCenters.set(i, newCenter);
    }
    // update centers
```

```
      Tuple oldCenters = value.centers;
      value.centers = newCenters;
      LOG.info("old centers: " + oldCenters + ", new centers: " + newCenters);
      // compare new/old centers
      boolean converged = true;
      for (int i = 0; i < value.centers.size() && converged; i++) {
        Tuple oldCenter = (Tuple) oldCenters.get(i);
        Tuple newCenter = (Tuple) newCenters.get(i);
        double sum = 0.0d;
        for (int j = 0; j < newCenter.size(); j++) {
          double v = ((DoubleWritable) newCenter.get(j)).get()
              - ((DoubleWritable) oldCenter.get(j)).get();
          sum += v * v;
        }
        double dist = Math.sqrt(sum);
        LOG.info("old center: " + oldCenter + ", new center: " + newCenter
            + ", dist: " + dist);
        // converge threshold for each center: 0.05
        converged = dist < 0.05d;
      }
      if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
        // converged or reach max iteration, output centers
        for (int i = 0; i < value.centers.size(); i++) {
          context.write(((Tuple) value.centers.get(i)).toArray());
        }
        // true means to terminate iteration
        return true;
      }
      // false means to continue iteration
      return false;
    }
  }
  private static void printUsage() {
    System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
    System.exit(-1);
  }
  public static void main(String[] args) throws IOException {
    if (args.length < 2)
      printUsage();
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(KmeansVertexReader.class);
    job.setRuntimePartitioning(false);
    job.setVertexClass(KmeansVertex.class);
    job.setAggregatorClass(KmeansAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
      job.setMaxIteration(Integer.parseInt(args[2]));
    long start = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
  }
}
```

上述代码说明如下：

- 第26行：定义 `KmeansVertex` 类， `compute()` 方法非常简单，只是调用上下文对象的 `aggregate` 方法，传入当前点的取值（Tuple类型，向量表示）。
- 第38行：定义 `KmeansVertexReader` 类，加载图，将表中每一条记录解析为一个点，点标识无关紧要，这里取传入的 `recordNum` 序号作为标识，点值为记录的所有列组成的Tuple。
- 第83行：定义 `KmeansAggregator` 类，这个类封装了Kmeans算法的主要逻辑，其中：
  - `createInitialValue` 为每一轮迭代创建的初始值（k类中心点）。如果是第一轮迭代（superstep=0），该取值为初始中心点，否则取值为上一轮结束时的新中心点。
  - `aggregate` 方法为每个点计算其到各个类中心的距离，并归为距离最短的类，并更新该类的 `sum` 和 `count` 。
  - `merge` 方法合并来自各个Worker收集的 `sum` 和 `count` 。
  - `terminate` 方法根据各个类的 `sum` 和 `count` 计算新的中心点。如果新中心点与之前的中心点距离小于某个阈值或者迭代次数到达最大迭代次数设置，则终止迭代（返回False），写最终的中心点到结果表。
- 第236行：主程序（ `main` 函数），定义 `GraphJob` 类，指定 `Vertex` 、 `GraphLoader` 、 `Aggregator` 等的实现，以及最大迭代次数（默认

30），并指定输入输出表。

- 第243行：`job.setRuntimePartitioning(false)`，对于Kmeans算法，加载图不需要进行点的分发。设置`RuntimePartitioning`为False，以提升加载图时的性能。

12.8.4. BiPartiteMatching

二分图（BiPartite）是指图的所有顶点可分为两个集合，每条边对应的两个顶点分别属于这两个集合。对于一个二分图G，M是它的一个子图。如果M的边集中任意两条边都不依附于同一个顶点，则称M为一个匹配。二分图匹配（BiPartiteMatching）常用于有明确供需关系场景（如交友网站等）下的信息匹配行为。

二分图匹配算法描述如下：

- 从左边第1个顶点开始，挑选未匹配点进行搜索，寻找增广路。
- 如果经过一个未匹配点，说明寻找成功。
- 更新路径信息，匹配边数+1，停止搜索。
- 如果一直没有找到增广路，则不再从这个点开始搜索。

代码示例

BiPartiteMatching算法的代码，如下所示。

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.Random;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
public class BipartiteMatching {
  private static final Text UNMATCHED = new Text("UNMATCHED");
  public static class TextPair implements Writable {
    public Text first;
    public Text second;
    public TextPair() {
      first = new Text();
      second = new Text();
    }
    public TextPair(Text first, Text second) {
      this.first = new Text(first);
      this.second = new Text(second);
    }
    @Override
    public void write(DataOutput out) throws IOException {
      first.write(out);
      second.write(out);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
      first = new Text();
      first.readFields(in);
      second = new Text();
      second.readFields(in);
    }
    @Override
    public String toString() {
      return first + ": " + second;
    }
  }
  public static class BipartiteMatchingVertexReader extends
      GraphLoader<Text, TextPair, NullWritable, Text> {
    @Override
    public void load(LongWritable recordNum, WritableRecord record,
        MutationContext<Text, TextPair, NullWritable, Text> context)
        throws IOException {
      BipartiteMatchingVertex vertex = new BipartiteMatchingVertex();
      vertex.setId((Text) record.get(0));
```

```
      vertex.setValue(new TextPair(UNMATCHED, (Text) record.get(1)));
      String[] adjs = record.get(2).toString().split(",");
      for (String adj : adjs) {
        vertex.addEdge(new Text(adj), null);
      }
      context.addVertexRequest(vertex);
    }
  }
  public static class BipartiteMatchingVertex extends
      Vertex<Text, TextPair, NullWritable, Text> {
    private static final Text LEFT = new Text("LEFT");
    private static final Text RIGHT = new Text("RIGHT");
    private static Random rand = new Random();
    @Override
    public void compute(
        ComputeContext<Text, TextPair, NullWritable, Text> context,
        Iterable<Text> messages) throws IOException {
      if (isMatched()) {
        voteToHalt();
        return;
      }
      switch ((int) context.getSuperstep() % 4) {
      case 0:
        if (isLeft()) {
          context.sendMessageToNeighbors(this, getId());
        }
        break;
      case 1:
        if (isRight()) {
          Text luckyLeft = null;
          for (Text message : messages) {
            if (luckyLeft == null) {
              luckyLeft = new Text(message);
            } else {
              if (rand.nextInt(1) == 0) {
                luckyLeft.set(message);
              }
            }
          }
          if (luckyLeft != null) {
            context.sendMessage(luckyLeft, getId());
          }
        }
        break;
      case 2:
        if (isLeft()) {
          Text luckyRight = null;
          for (Text msg : messages) {
            if (luckyRight == null) {
              luckyRight = new Text(msg);
            } else {
              if (rand.nextInt(1) == 0) {
                luckyRight.set(msg);
              }
            }
          }
          if (luckyRight != null) {
            setMatchVertex(luckyRight);
            context.sendMessage(luckyRight, getId());
          }
        }
        break;
      case 3:
        if (isRight()) {
          for (Text msg : messages) {
            setMatchVertex(msg);
          }
        }
        break;
      }
    }
    @Override
    public void cleanup(
        WorkerContext<Text, TextPair, NullWritable, Text> context)
        throws IOException {
```

```
      context.write(getId(), getValue().first);
    }
    private boolean isMatched() {
      return !getValue().first.equals(UNMATCHED);
    }
    private boolean isLeft() {
      return getValue().second.equals(LEFT);
    }
    private boolean isRight() {
      return getValue().second.equals(RIGHT);
    }
    private void setMatchVertex(Text matchVertex) {
      getValue().first.set(matchVertex);
    }
  }
  private static void printUsage() {
    System.err.println("BipartiteMatching <input> <output> [maxIteration]");
  }
  public static void main(String[] args) throws IOException {
    if (args.length < 2) {
      printUsage();
    }
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(BipartiteMatchingVertexReader.class);
    job.setVertexClass(BipartiteMatchingVertex.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    int maxIteration = 30;
    if (args.length > 2) {
      maxIteration = Integer.parseInt(args[2]);
    }
    job.setMaxIteration(maxIteration);
    job.run();
  }
}
```

12.8.5. 强连通分量

在有向图中，如果从任意一个顶点出发，都能通过图中的边到达图中的每一个顶点，则称之为强连通图。一张有向图的顶点数极大的强连通子图称为强连通分量。

这一算法示例基于parallel Coloring algorithm。每个顶点包含以下两个部分：

- colorID：在向前遍历过程中存储顶点v的颜色。在计算结束时，具有相同colorID的顶点属于一个强连通分量。
- transposeNeighbors：存储输入图的转置图中顶点v的邻居ID。

算法包含以下四个部分：

- 生成转置图：包含两个超步，首先每个顶点发送ID到其出边对应的邻居，这些ID在第二个超步中会存为transposeNeighbors值。
- 修剪：一个超步，每个只有一个入边或出边的顶点，将其colorID设为自身ID，状态设为不活跃，后面传给该顶点的信号被忽略。
- 向前遍历：顶点包括两个子过程（超步），启动和休眠。在启动阶段，每个顶点将其colorID设置为自身ID，同时将其ID传给出边对应的邻居。休眠阶段，顶点使用其收到的最大colorID更新自身colorID，并传播其colorID，直到colorID收敛。当colorID收敛，master进程将全局对象设置为向后遍历。
- 向后遍历：同样包含两个子过程，启动和休眠。启动阶段，每一个ID等于colorID的顶点将其ID传递给其转置图邻居顶点，同时将自身状态设置为不活跃，后面传给该顶点的信号可忽略。在每一个休眠步，每个顶点接收到与其colorID匹配的信号，并将其colorID在转置图中传播，随后设置自身状态为不活跃。该步结束后如果仍有活跃顶点，则回到修剪步。

代码示例

强连通分量的代码，如下所示。

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.IntWritable;
import com.aliyun.odps.io.LongWritable;
```

```
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
 * Definition from Wikipedia:
 * In the mathematical theory of directed graphs, a graph is said
 * to be strongly connected if every vertex is reachable from every
 * other vertex. The strongly connected components of an arbitrary
 * directed graph form a partition into subgraphs that are themselves
 * strongly connected.
 *
 * Algorithms with four phases as follows.
 * 1. Transpose Graph Formation: Requires two supersteps. In the first
 * superstep, each vertex sends a message with its ID to all its outgoing
 * neighbors, which in the second superstep are stored in transposeNeighbors.
 *
 * 2. Trimming: Takes one superstep. Every vertex with only in-coming or
 *  only outgoing edges (or neither) sets its colorID to its own ID and
 *  becomes inactive. Messages subsequently sent to the vertex are ignored.
 *
 * 3. Forward-Traversal: There are two sub phases: Start and Rest. In the
 * Start phase, each vertex sets its colorID to its own ID and propagates
 * its ID to its outgoing neighbors. In the Rest phase, vertices update
 * their own colorIDs with the minimum colorID they have seen, and propagate
 * their colorIDs, if updated, until the colorIDs converge.
 * Set the phase to Backward-Traversal when the colorIDs converge.
 *
 * 4. Backward-Traversal: We again break the phase into Start and Rest.
 * In Start, every vertex whose ID equals its colorID propagates its ID to
 * the vertices in transposeNeighbors and sets itself inactive. Messages
 * subsequently sent to the vertex are ignored. In each of the Rest phase supersteps,
 * each vertex receiving a message that matches its colorID: (1) propagates
 * its colorID in the transpose graph; (2) sets itself inactive. Messages
 * subsequently sent to the vertex are ignored. Set the phase back to Trimming
 * if not all vertex are inactive.
 *
 * http://ilpubs.stanford.edu:8090/1077/3/p535-salihoglu.pdf
 */
public class StronglyConnectedComponents {
  public final static int STAGE_TRANSPOSE_1 = 0;
  public final static int STAGE_TRANSPOSE_2 = 1;
  public final static int STAGE_TRIMMING = 2;
  public final static int STAGE_FW_START = 3;
  public final static int STAGE_FW_REST = 4;
  public final static int STAGE_BW_START = 5;
  public final static int STAGE_BW_REST = 6;
  /**
   * The value is composed of component id, incoming neighbors,
   * active status and updated status.
   */
  public static class MyValue implements Writable {
    LongWritable sccID;// strongly connected component id
    Tuple inNeighbors; // transpose neighbors
    BooleanWritable active; // vertex is active or not
    BooleanWritable updated; // sccID is updated or not
    public MyValue() {
      this.sccID = new LongWritable(Long.MAX_VALUE);
      this.inNeighbors = new Tuple();
      this.active = new BooleanWritable(true);
      this.updated = new BooleanWritable(false);
    }
    public void setSccID(LongWritable sccID) {
      this.sccID = sccID;
    }
    public LongWritable getSccID() {
      return this.sccID;
    }
    public void setInNeighbors(Tuple inNeighbors) {
      this.inNeighbors = inNeighbors;
    }
    public Tuple getInNeighbors() {
      return this.inNeighbors;
    }
    public void addInNeighbor(LongWritable neighbor) {
```

```
      this.inNeighbors.append(new LongWritable(neighbor.get()));
    }
    public boolean isActive() {
      return this.active.get();
    }
    public void setActive(boolean status) {
      this.active.set(status);
    }
    public boolean isUpdated() {
      return this.updated.get();
    }
    public void setUpdated(boolean update) {
      this.updated.set(update);
    }
    @Override
    public void write(DataOutput out) throws IOException {
      this.sccID.write(out);
      this.inNeighbors.write(out);
      this.active.write(out);
      this.updated.write(out);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
      this.sccID.readFields(in);
      this.inNeighbors.readFields(in);
      this.active.readFields(in);
      this.updated.readFields(in);
    }
    @Override
    public String toString() {
      StringBuilder sb = new StringBuilder();
      sb.append("sccID: " + sccID.get());
      sb.append(" inNeighbores: " + inNeighbors.toDelimitedString(','));
      sb.append(" active: " + active.get());
      sb.append(" updated: " + updated.get());
      return sb.toString();
    }
  }
  public static class SCCVertex extends
  Vertex<LongWritable, MyValue, NullWritable, LongWritable> {
    public SCCVertex() {
      this.setValue(new MyValue());
    }
    @Override
    public void compute(
        ComputeContext<LongWritable, MyValue, NullWritable, LongWritable> context,
        Iterable<LongWritable> msgs) throws IOException {
      // Messages sent to inactive vertex are ignored.
      if (!this.getValue().isActive()) {
        this.voteToHalt();
        return;
      }
      int stage = ((SCCAggrValue)context.getLastAggregatedValue(0)).getStage();
      switch (stage) {
      case STAGE_TRANSPOSE_1:
        context.sendMessageToNeighbors(this, this.getId());
        break;
      case STAGE_TRANSPOSE_2:
        for (LongWritable msg: msgs) {
          this.getValue().addInNeighbor(msg);
        }
      case STAGE_TRIMMING:
        this.getValue().setSccID(getId());
        if (this.getValue().getInNeighbors().size() == 0 ||
            this.getNumEdges() == 0) {
          this.getValue().setActive(false);
        }
        break;
      case STAGE_FW_START:
        this.getValue().setSccID(getId());
        context.sendMessageToNeighbors(this, this.getValue().getSccID());
        break;
      case STAGE_FW_REST:
        long minSccID = Long.MAX_VALUE;
        for (LongWritable msg : msgs) {
```

```
          if (msg.get() < minSccID) {
            minSccID = msg.get();
          }
        }
        if (minSccID < this.getValue().getSccID().get()) {
          this.getValue().setSccID(new LongWritable(minSccID));
          context.sendMessageToNeighbors(this, this.getValue().getSccID());
          this.getValue().setUpdated(true);
        } else {
          this.getValue().setUpdated(false);
        }
        break;
      case STAGE_BW_START:
        if (this.getId().equals(this.getValue().getSccID())) {
          for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
            context.sendMessage((LongWritable)neighbor, this.getValue().getSccID());
          }
          this.getValue().setActive(false);
        }
        break;
      case STAGE_BW_REST:
        this.getValue().setUpdated(false);
        for (LongWritable msg : msgs) {
          if (msg.equals(this.getValue().getSccID())) {
            for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
              context.sendMessage((LongWritable)neighbor, this.getValue().getSccID());
            }
            this.getValue().setActive(false);
            this.getValue().setUpdated(true);
            break;
          }
        }
        break;
      }
      context.aggregate(0, getValue());
    }
    @Override
    public void cleanup(
        WorkerContext<LongWritable, MyValue, NullWritable, LongWritable> context)
        throws IOException {
      context.write(getId(), getValue().getSccID());
    }
  }
  /**
   * The SCCAggrValue maintains global stage and graph updated and active status.
   * updated is true only if one vertex is updated.
   * active is true only if one vertex is active.
   */
  public static class SCCAggrValue implements Writable {
    IntWritable stage = new IntWritable(STAGE_TRANSPOSE_1);
    BooleanWritable updated = new BooleanWritable(false);
    BooleanWritable active = new BooleanWritable(false);
    public void setStage(int stage) {
      this.stage.set(stage);
    }
    public int getStage() {
      return this.stage.get();
    }
    public void setUpdated(boolean updated) {
      this.updated.set(updated);
    }
    public boolean getUpdated() {
      return this.updated.get();
    }
    public void setActive(boolean active) {
      this.active.set(active);
    }
    public boolean getActive() {
      return this.active.get();
    }
    @Override
    public void write(DataOutput out) throws IOException {
      this.stage.write(out);
      this.updated.write(out);
      this.active.write(out);
    }
```

```
  }
  @Override
  public void readFields(DataInput in) throws IOException {
    this.stage.readFields(in);
    this.updated.readFields(in);
    this.active.readFields(in);
  }
}
/**
 * The job of SCCAggregator is to schedule global stage in every superstep.
 */
public static class SCCAggregator extends Aggregator<SCCAggrValue> {
  @SuppressWarnings("rawtypes")
  @Override
  public SCCAggrValue createStartupValue(WorkerContext context) throws IOException {
    return new SCCAggrValue();
  }
  @SuppressWarnings("rawtypes")
  @Override
  public SCCAggrValue createInitialValue(WorkerContext context)
      throws IOException {
    return (SCCAggrValue) context.getLastAggregatedValue(0);
  }
  @Override
  public void aggregate(SCCAggrValue value, Object item) throws IOException {
    MyValue v = (MyValue)item;
    if ((value.getStage() == STAGE_FW_REST || value.getStage() == STAGE_BW_REST)
        && v.isUpdated()) {
      value.setUpdated(true);
    }
    // only active vertex invoke aggregate()
    value.setActive(true);
  }
  @Override
  public void merge(SCCAggrValue value, SCCAggrValue partial)
      throws IOException {
    boolean updated = value.getUpdated() || partial.getUpdated();
    value.setUpdated(updated);
    boolean active = value.getActive() || partial.getActive();
    value.setActive(active);
  }
  @SuppressWarnings("rawtypes")
  @Override
  public boolean terminate(WorkerContext context, SCCAggrValue value)
      throws IOException {
    // If all vertices is inactive, job is over.
    if (!value.getActive()) {
      return true;
    }
    // state machine
    switch (value.getStage()) {
    case STAGE_TRANSPOSE_1:
      value.setStage(STAGE_TRANSPOSE_2);
      break;
    case STAGE_TRANSPOSE_2:
      value.setStage(STAGE_TRIMMING);
      break;
    case STAGE_TRIMMING:
      value.setStage(STAGE_FW_START);
      break;
    case STAGE_FW_START:
      value.setStage(STAGE_FW_REST);
      break;
    case STAGE_FW_REST:
      if (value.getUpdated()) {
        value.setStage(STAGE_FW_REST);
      } else {
        value.setStage(STAGE_BW_START);
      }
      break;
    case STAGE_BW_START:
      value.setStage(STAGE_BW_REST);
      break;
    case STAGE_BW_REST:
      if (value.getUpdated()) {
        value.setStage(STAGE_BW_REST);
```

```
        } else {
          value.setStage(STAGE_TRIMMING);
        }
        break;
      }
      value.setActive(false);
      value.setUpdated(false);
      return false;
    }
  }
  public static class SCCVertexReader extends
  GraphLoader<LongWritable, MyValue, NullWritable, LongWritable> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, MyValue, NullWritable, LongWritable> context)
    throws IOException {
      SCCVertex vertex = new SCCVertex();
      vertex.setId((LongWritable) record.get(0));
      String[] edges = record.get(1).toString().split(",");
      for (int i = 0; i < edges.length; i++) {
        try {
          long destID = Long.parseLong(edges[i]);
          vertex.addEdge(new LongWritable(destID), NullWritable.get());
        } catch(NumberFormatException nfe) {
          System.err.println("Ignore " + nfe);
        }
      }
      context.addVertexRequest(vertex);
    }
  }
  public static void main(String[] args) throws IOException {
    if (args.length < 2) {
      System.out.println("Usage: <input> <output>");
      System.exit(-1);
    }
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(SCCVertexReader.class);
    job.setVertexClass(SCCVertex.class);
    job.setAggregatorClass(SCCAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
  }
}
```

12.8.6. 连通分量

如果两个顶点之间存在路径，则称两个顶点为连通的。如果无向图G中任意两个顶点都是连通的，则为连通图，否则称为非连通图。其顶点个数极大的连通子图称为连通分量。

本算法计算每个点的连通分量成员，最后输出顶点值中包含最小顶点ID的连通分量。将最小顶点ID沿着边传播到连通分量的所有顶点。

代码示例

连通分量的代码，如下所示。

```
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.examples.SSSP.MinLongCombiner;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.WritableRecord;
/**
```

```
 * Compute the connected component membership of each vertex and output
 * each vertex which's value containing the smallest id in the connected
 * component containing that vertex.
 *
 * Algorithm: propagate the smallest vertex id along the edges to all
 * vertices of a connected component.
 *
 */
public class ConnectedComponents {
  public static class CCVertex extends
    Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context,
        Iterable<LongWritable> msgs) throws IOException {
      if (context.getSuperstep() == 0L) {
        this.setValue(getId());
        context.sendMessageToNeighbors(this, getValue());
        return;
      }
      long minID = Long.MAX_VALUE;
      for (LongWritable id : msgs) {
        if (id.get() < minID) {
          minID = id.get();
        }
      }
      if (minID < this.getValue().get()) {
        this.setValue(new LongWritable(minID));
        context.sendMessageToNeighbors(this, getValue());
      } else {
        this.voteToHalt();
      }
    }
    /**
     * Output Table Description:
     * +-----------------+----------------------------------------+
     * | Field | Type    | Comment                                |
     * +-----------------+----------------------------------------+
     * | v     | bigint  | vertex id                              |
     * | minID | bigint  | smallest id in the connected component |
     * +-----------------+----------------------------------------+
     */
    @Override
    public void cleanup(
        WorkerContext<LongWritable, LongWritable, NullWritable, LongWritable> context)
        throws IOException {
      context.write(getId(), getValue());
    }
  }
  /**
   * Input Table Description:
   * +-----------------+-------------------------------------------------------+
   * | Field | Type    | Comment                                               |
   * +-----------------+-------------------------------------------------------+
   * | v     | bigint  | vertex id                                             |
   * | es    | string  | comma separated target vertex id of outgoing edges |
   * +-----------------+-------------------------------------------------------+
   *
   * Example:
   * For graph:
   *       1 ----- 2
   *       |       |
   *       3 ----- 4
   * Input table:
   * +-----------+
   * | v  | es   |
   * +-----------+
   * | 1  | 2,3  |
   * | 2  | 1,4  |
   * | 3  | 1,4  |
   * | 4  | 2,3  |
   * +-----------+
   */
  public static class CCVertexReader extends
    GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
```

```
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context)
    throws IOException {
      CCVertex vertex = new CCVertex();
      vertex.setId((LongWritable) record.get(0));
      String[] edges = record.get(1).toString().split(",");
      for (int i = 0; i < edges.length; i++) {
        long destID = Long.parseLong(edges[i]);
        vertex.addEdge(new LongWritable(destID), NullWritable.get());
      }
      context.addVertexRequest(vertex);
    }
  }
  public static void main(String[] args) throws IOException {
    if (args.length < 2) {
      System.out.println("Usage: <input> <output>");
      System.exit(-1);
    }
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(CCVertexReader.class);
    job.setVertexClass(CCVertex.class);
    job.setCombinerClass(MinLongCombiner.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
  }
}
```

12.8.7. 拓扑排序

对于有向边（u,v），定义所有满足u<v的顶点序列为拓扑序列。拓扑排序是求一个有向图的拓扑序列的算法。

算法步骤如下：

1. 从图中找到一个没有入边的顶点，并输出。
2. 从图中删除该点，及其所有出边。
3. 重复以上步骤，直到所有没有入边的顶点都已输出。

代码示例

拓扑排序算法的代码，如下所示。

```
import java.io.IOException;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.WritableRecord;
public class TopologySort {
  private final static Log LOG = LogFactory.getLog(TopologySort.class);
  public static class TopologySortVertex extends
      Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context,
        Iterable<LongWritable> messages) throws IOException {
      // in superstep 0, each vertex sends message whose value is 1 to its
      // neighbors
      if (context.getSuperstep() == 0) {
```

```
    if (context.getSuperstep() == 0) {
      if (hasEdges()) {
        context.sendMessageToNeighbors(this, new LongWritable(1L));
      }
    } else if (context.getSuperstep() >= 1) {
      // compute each vertex's indegree
      long indegree = getValue().get();
      for (LongWritable msg : messages) {
        indegree += msg.get();
      }
      setValue(new LongWritable(indegree));
      if (indegree == 0) {
        voteToHalt();
        if (hasEdges()) {
          context.sendMessageToNeighbors(this, new LongWritable(-1L));
        }
        context.write(new LongWritable(context.getSuperstep()), getId());
        LOG.info("vertex: " + getId());
      }
      context.aggregate(new LongWritable(indegree));
    }
  }
}
public static class TopologySortVertexReader extends
    GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
  @Override
  public void load(
      LongWritable recordNum,
      WritableRecord record,
      MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context)
      throws IOException {
    TopologySortVertex vertex = new TopologySortVertex();
    vertex.setId((LongWritable) record.get(0));
    vertex.setValue(new LongWritable(0));
    String[] edges = record.get(1).toString().split(",");
    for (int i = 0; i < edges.length; i++) {
      long edge = Long.parseLong(edges[i]);
      if (edge >= 0) {
        vertex.addEdge(new LongWritable(Long.parseLong(edges[i])),
            NullWritable.get());
      }
    }
    LOG.info(record.toString());
    context.addVertexRequest(vertex);
  }
}
public static class LongSumCombiner extends
    Combiner<LongWritable, LongWritable> {
  @Override
  public void combine(LongWritable vertexId, LongWritable combinedMessage,
      LongWritable messageToCombine) throws IOException {
    combinedMessage.set(combinedMessage.get() + messageToCombine.get());
  }
}
public static class TopologySortAggregator extends
    Aggregator<BooleanWritable> {
  @SuppressWarnings("rawtypes")
  @Override
  public BooleanWritable createInitialValue(WorkerContext context)
      throws IOException {
    return new BooleanWritable(true);
  }
  @Override
  public void aggregate(BooleanWritable value, Object item)
      throws IOException {
    boolean hasCycle = value.get();
    boolean inDegreeNotZero = ((LongWritable) item).get() == 0 ? false : true;
    value.set(hasCycle && inDegreeNotZero);
  }
  @Override
  public void merge(BooleanWritable value, BooleanWritable partial)
      throws IOException {
    value.set(value.get() && partial.get());
  }
  @SuppressWarnings("rawtypes")
  @Override
```

```
    public boolean terminate(WorkerContext context, BooleanWritable value)
        throws IOException {
      if (context.getSuperstep() == 0) {
        // since the initial aggregator value is true, and in superstep we don't
        // do aggregate
        return false;
      }
      return value.get();
    }
  }
  public static void main(String[] args) throws IOException {
    if (args.length != 2) {
      System.out.println("Usage : <inputTable> <outputTable>");
      System.exit(-1);
    }
    // 输入表形式如下。
    // 0 1，2
    // 1 3
    // 2 3
    // 3 -1
    // 第一列为vertexid，第二列为该点边的destination vertexid。如果值为-1，表示该点无出边。
    // 输出表形式如下。
    // 0 0
    // 1 1
    // 1 2
    // 2 3
    // 第一列为supstep值，隐含了拓扑顺序，第二列为vertexid。
    // TopologySortAggregator用来判断图中是否有环。
    // 若输入的图有环，则当图中active的点入度都不为0时，迭代结束。
    // 用户可以通过输入表和输出表的记录数来判断一个有向图是否有环。
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(TopologySortVertexReader.class);
    job.setVertexClass(TopologySortVertex.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    job.setCombinerClass(LongSumCombiner.class);
    job.setAggregatorClass(TopologySortAggregator.class);
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
  }
}
```

12.8.8. 线性回归

在统计学中，线性回归是用来确定两种或两种以上变量间的相互依赖关系的统计分析方法，与分类算法处理离散预测不同。

回归算法可对连续值类型进行预测。线性回归算法定义损失函数为样本集的最小平方误差之和，通过最小化损失函数求解权重矢量。

常用的解法是梯度下降法，流程如下：

1. 初始化权重矢量，给定下降速率以及迭代次数（或者迭代收敛条件）。
2. 对每个样本，计算最小平方误差。
3. 对最小平方误差求和，根据下降速率更新权重。
4. 重复迭代直到收敛。

代码示例

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
```

```
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
 * LineRegression input: y,x1,x2,x3,......
 **/
public class LinearRegression {
  public static class GradientWritable implements Writable {
    Tuple lastTheta;
    Tuple currentTheta;
    Tuple tmpGradient;
    LongWritable count;
    DoubleWritable lost;
    @Override
    public void readFields(DataInput in) throws IOException {
      lastTheta = new Tuple();
      lastTheta.readFields(in);
      currentTheta = new Tuple();
      currentTheta.readFields(in);
      tmpGradient = new Tuple();
      tmpGradient.readFields(in);
      count = new LongWritable();
      count.readFields(in);
      /* update 1: add a variable to store lost at every iteration */
      lost = new DoubleWritable();
      lost.readFields(in);
    }
    @Override
    public void write(DataOutput out) throws IOException {
      lastTheta.write(out);
      currentTheta.write(out);
      tmpGradient.write(out);
      count.write(out);
      lost.write(out);
    }
  }
  public static class LinearRegressionVertex extends
      Vertex<LongWritable, Tuple, NullWritable, NullWritable> {
    @Override
    public void compute(
        ComputeContext<LongWritable, Tuple, NullWritable, NullWritable> context,
        Iterable<NullWritable> messages) throws IOException {
      context.aggregate(getValue());
    }
  }
  public static class LinearRegressionVertexReader extends
      GraphLoader<LongWritable, Tuple, NullWritable, NullWritable> {
    @Override
    public void load(LongWritable recordNum, WritableRecord record,
        MutationContext<LongWritable, Tuple, NullWritable, NullWritable> context)
        throws IOException {
      LinearRegressionVertex vertex = new LinearRegressionVertex();
      vertex.setId(recordNum);
      vertex.setValue(new Tuple(record.getAll()));
      context.addVertexRequest(vertex);
    }
  }
  public static class LinearRegressionAggregator extends
      Aggregator<GradientWritable> {
    @SuppressWarnings("rawtypes")
    @Override
    public GradientWritable createInitialValue(WorkerContext context)
        throws IOException {
      if (context.getSuperstep() == 0) {
        /* set initial value, all 0 */
        GradientWritable grad = new GradientWritable();
        grad.lastTheta = new Tuple();
        grad.currentTheta = new Tuple();
        grad.tmpGradient = new Tuple();
        grad.count = new LongWritable(1);
        grad.lost = new DoubleWritable(0.0);
        int n = (int) Long.parseLong(context.getConfiguration()
            .get("Dimension"));
        for (int i = 0; i < n; i++) {
          grad.lastTheta.append(new DoubleWritable(0));
          grad.currentTheta.append(new DoubleWritable(0));
```

```
        grad.currentTheta.append(new DoubleWritable(0));
        grad.tmpGradient.append(new DoubleWritable(0));
      }
      return grad;
    } else
      return (GradientWritable) context.getLastAggregatedValue(0);
  }
  public static double vecMul(Tuple value, Tuple theta) {
    /* perform this partial computing: y(i)-hθ(x(i)) for each sample */
    /* value denote a piece of sample and value(0) is y */
    double sum = 0.0;
    for (int j = 1; j < value.size(); j++)
      sum += Double.parseDouble(value.get(j).toString())
          * Double.parseDouble(theta.get(j).toString());
    Double tmp = Double.parseDouble(theta.get(0).toString()) + sum
        - Double.parseDouble(value.get(0).toString());
    return tmp;
  }
  @Override
  public void aggregate(GradientWritable gradient, Object value)
      throws IOException {
    /*
     * perform on each vertex--each sample i: set theta(j) for each sample i
     * for each dimension
     */
    double tmpVar = vecMul((Tuple) value, gradient.currentTheta);
    /*
     * update 2:local worker aggregate(), perform like merge() below. This
     * means the variable gradient denotes the previous aggregated value
     */
    gradient.tmpGradient.set(0, new DoubleWritable(
        ((DoubleWritable) gradient.tmpGradient.get(0)).get() + tmpVar));
    gradient.lost.set(Math.pow(tmpVar, 2));
    /*
     * calculate (y(i)-hθ(x(i))) x(i)(j) for each sample i for each
     * dimension j
     */
    for (int j = 1; j < gradient.tmpGradient.size(); j++)
      gradient.tmpGradient.set(j, new DoubleWritable(
          ((DoubleWritable) gradient.tmpGradient.get(j)).get() + tmpVar
              * Double.parseDouble(((Tuple) value).get(j).toString())));
  }
  @Override
  public void merge(GradientWritable gradient, GradientWritable partial)
      throws IOException {
    /* perform SumAll on each dimension for all samples. */
    Tuple master = (Tuple) gradient.tmpGradient;
    Tuple part = (Tuple) partial.tmpGradient;
    for (int j = 0; j < gradient.tmpGradient.size(); j++) {
      DoubleWritable s = (DoubleWritable) master.get(j);
      s.set(s.get() + ((DoubleWritable) part.get(j)).get());
    }
    gradient.lost.set(gradient.lost.get() + partial.lost.get());
  }
  @SuppressWarnings("rawtypes")
  @Override
  public boolean terminate(WorkerContext context, GradientWritable gradient)
      throws IOException {
    /*
     * 1. calculate new theta 2. judge the diff between last step and this
     * step, if smaller than the threshold, stop iteration
     */
    gradient.lost = new DoubleWritable(gradient.lost.get()
        / (2 * context.getTotalNumVertices()));
    /*
     * we can calculate lost in order to make sure the algorithm is running on
     * the right direction (for debug)
     */
    System.out.println(gradient.count + " lost:" + gradient.lost);
    Tuple tmpGradient = gradient.tmpGradient;
    System.out.println("tmpGra" + tmpGradient);
    Tuple lastTheta = gradient.lastTheta;
    Tuple tmpCurrentTheta = new Tuple(gradient.currentTheta.size());
    System.out.println(gradient.count + " terminate_start_last:" + lastTheta);
    double alpha = 0.07; // learning rate
    // alpha =
```

```
      // Double.parseDouble(context.getConfiguration().get("Alpha"));
      /* perform theta(j) = theta(j)-alpha*tmpGradient */
      long M = context.getTotalNumVertices();
      /*
       * update 3: add (/M) on the code. The original code forget this step
       */
      for (int j = 0; j < lastTheta.size(); j++) {
        tmpCurrentTheta
            .set(
                j,
                new DoubleWritable(Double.parseDouble(lastTheta.get(j)
                    .toString())
                    - alpha
                    / M
                    * Double.parseDouble(tmpGradient.get(j).toString())));
      }
      System.out.println(gradient.count + " terminate_start_current:"
          + tmpCurrentTheta);
      // judge if convergence is happening.
      double diff = 0.00d;
      for (int j = 0; j < gradient.currentTheta.size(); j++)
        diff += Math.pow(((DoubleWritable) tmpCurrentTheta.get(j)).get()
            - ((DoubleWritable) lastTheta.get(j)).get(), 2);
      if (/*
           * Math.sqrt(diff) < 0.00000000005d ||
           */Long.parseLong(context.getConfiguration().get("Max_Iter_Num")) == gradient.count
          .get()) {
        context.write(gradient.currentTheta.toArray());
        return true;
      }
      gradient.lastTheta = tmpCurrentTheta;
      gradient.currentTheta = tmpCurrentTheta;
      gradient.count.set(gradient.count.get() + 1);
      int n = (int) Long.parseLong(context.getConfiguration().get("Dimension"));
      /*
       * update 4: Important!!! Remember this step. Graph won't reset the
       * initial value for global variables at the beginning of each iteration
       */
      for (int i = 0; i < n; i++) {
        gradient.tmpGradient.set(i, new DoubleWritable(0));
      }
      return false;
    }
  }
  public static void main(String[] args) throws IOException {
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(LinearRegressionVertexReader.class);
    job.setRuntimePartitioning(false);
    job.setNumWorkers(3);
    job.setVertexClass(LinearRegressionVertex.class);
    job.setAggregatorClass(LinearRegressionAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    job.setMaxIteration(Integer.parseInt(args[2])); // Numbers of Iteration
    job.setInt("Max_Iter_Num", Integer.parseInt(args[2]));
    job.setInt("Dimension", Integer.parseInt(args[3])); // Dimension
    job.setFloat("Alpha", Float.parseFloat(args[4])); // Learning rate
    long start = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
  }
}
```

12.8.9. 三角形计数

三角形计数算法用于计算通过每个顶点的三角形个数。

算法实现的流程如下：

1. 每个顶点将其ID发送给所有出边邻居。
2. 存储入边和出边邻居并发送给出边邻居。
3. 对每条边计算其终点的交集数量，并求和，结果输出到表。

4. 将表中的输出结果求和并除以三，即得到三角形个数。

代码示例

三角形计数算法的代码，如下所示。

```
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
 * Compute the number of triangles passing through each vertex.
 *
 * The algorithm can be computed in three supersteps:
 * I. Each vertex sends a message with its ID to all its outgoing
 * neighbors.
 * II. The incoming neighbors and outgoing neighbors are stored and
 * send to outgoing neighbors.
 * III. For each edge compute the intersection of the sets at destination
 * vertex and sum them, then output to table.
 *
 * The triangle count is the sum of output table and divide by three since
 * each triangle is counted three times.
 *
 **/
public class TriangleCount {
  public static class TCVertex extends
    Vertex<LongWritable, Tuple, NullWritable, Tuple> {
    @Override
    public void setup(
        WorkerContext<LongWritable, Tuple, NullWritable, Tuple> context)
      throws IOException {
      // collect the outgoing neighbors
      Tuple t = new Tuple();
      if (this.hasEdges()) {
        for (Edge<LongWritable, NullWritable> edge : this.getEdges()) {
          t.append(edge.getDestVertexId());
        }
      }
      this.setValue(t);
    }
    @Override
    public void compute(
        ComputeContext<LongWritable, Tuple, NullWritable, Tuple> context,
        Iterable<Tuple> msgs) throws IOException {
      if (context.getSuperstep() == 0L) {
        // sends a message with its ID to all its outgoing neighbors
        Tuple t = new Tuple();
        t.append(getId());
        context.sendMessageToNeighbors(this, t);
      } else if (context.getSuperstep() == 1L) {
        // store the incoming neighbors
        for (Tuple msg : msgs) {
          for (Writable item : msg.getAll()) {
            if (!this.getValue().getAll().contains((LongWritable)item)) {
              this.getValue().append((LongWritable)item);
            }
          }
        }
        // send both incoming and outgoing neighbors to all outgoing neighbors
        context.sendMessageToNeighbors(this, getValue());
      } else if (context.getSuperstep() == 2L) {
        // count the sum of intersection at each edge
        long count = 0;
        for (Tuple msg : msgs) {
          for (Writable id : msg.getAll()) {
```

```
          if (getValue().getAll().contains(id)) {
            count ++;
          }
        }
      }
      // output to table
      context.write(getId(), new LongWritable(count));
      this.voteToHalt();
    }
  }
}
public static class TCVertexReader extends
GraphLoader<LongWritable, Tuple, NullWritable, Tuple> {
  @Override
  public void load(
      LongWritable recordNum,
      WritableRecord record,
      MutationContext<LongWritable, Tuple, NullWritable, Tuple> context)
  throws IOException {
    TCVertex vertex = new TCVertex();
    vertex.setId((LongWritable) record.get(0));
    String[] edges = record.get(1).toString().split(",");
    for (int i = 0; i < edges.length; i++) {
      try {
        long destID = Long.parseLong(edges[i]);
        vertex.addEdge(new LongWritable(destID), NullWritable.get());
      } catch(NumberFormatException nfe) {
        System.err.println("Ignore " + nfe);
      }
    }
    context.addVertexRequest(vertex);
  }
}
public static void main(String[] args) throws IOException {
  if (args.length < 2) {
    System.out.println("Usage: <input> <output>");
    System.exit(-1);
  }
  GraphJob job = new GraphJob();
  job.setGraphLoaderClass(TCVertexReader.class);
  job.setVertexClass(TCVertex.class);
  job.addInput(TableInfo.builder().tableName(args[0]).build());
  job.addOutput(TableInfo.builder().tableName(args[1]).build());
  long startTime = System.currentTimeMillis();
  job.run();
  System.out.println("Job Finished in "
      + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}
}
```

12.8.10. 输入点表

本文为您提供输入点表的代码。

输入点表的代码，如下所示。

```
import java.io.IOException;
import com.aliyun.odps.conf.Configuration;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.VertexResolver;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableComparable;
import com.aliyun.odps.io.WritableRecord;
/**
 * 本示例是用于展示，对于不同类型的数据类型，如何编写图作业程序载入数据。主要展示GraphLoader和VertexResolver配合完成图的构建。
 * ODPS Graph的作业输入都为ODPS的Table，假设作业输入有两张表，一张存储点的信息，一张存储边的信息。
 * 存储点信息的表的格式，如下。
```

```
 * +------------------------+
 * | VertexID | VertexValue |
 * +------------------------+
 * |       id0|            9|
 * +------------------------+
 * |       id1|            7|
 * +------------------------+
 * |       id2|            8|
 * +------------------------+
 *
 * 存储边信息的表的格式如下。
 * +-----------------------------------+
 * | VertexID | DestVertexID| EdgeValue|
 * +-----------------------------------+
 * |       id0|          id1|         1|
 * +-----------------------------------+
 * |       id0|          id2|         2|
 * +-----------------------------------+
 * |       id2|          id1|         3|
 * +-----------------------------------+
 *
 * 结合两张表的数据，表示id0有两条出边，分别指向id1和id2；id2有一条出边，指向id1；id1没有出边。
 *
 * 对于此种类型的数据，在GraphLoader::load(LongWritable, Record, MutationContext)
 * ，可以使用MutationContext#addVertexRequest(Vertex)向图中请求添加点，使用
 * link MutationContext#addEdgeRequest(WritableComparable, Edge)向图中请求添加边，然后，在
 * link VertexResolver#resolve(WritableComparable, Vertex, VertexChanges, boolean)
 * 中，将load 方法中添加的点和边合并到一个Vertex对象中，作为返回值，添加到最后参与计算的图中。
 *
 **/
public class VertexInputFormat {
  private final static String EDGE_TABLE = "edge.table";
  /**
   * 将Record解释为Vertex和Edge，每个Record根据其来源，表示一个Vertex或者一条Edge。
   *
   * 类似于com.aliyun.odps.mapreduce.Mapper#map，输入Record，生成键值对，此处的键是Vertex的ID，
   * 值是Vertex或Edge，通过上下文Context写出，这些键值对会在LoadingVertexResolver出根据Vertex的ID汇总。
   *
   * 注意：此处添加的点或边只是根据Record内容发出的请求，并不是最后参与计算的点或边，只有在随后的VertexResolver
   * 中添加的点或边才参与计算。
   **/
  public static class VertexInputLoader extends
      GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
    private boolean isEdgeData;
    /**
     * 配置VertexInputLoader。
     *
     * @param conf
     *          作业的配置参数，是在main中使用GraphJob配置的，或者是在console中set的。
     * @param workerId
     *          当前工作的worker的序号，从0开始，可以用于构造唯一的vertex id。
     * @param inputTableInfo
     *          当前worker载入的输入表信息，可以用于确定当前输入是哪种类型的数据，即Record的格式。
     **/
    @Override
    public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
      isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableName());
    }
    /**
     * 根据Record中的内容，解析为对应的边，并请求添加到图中。
     *
     * @param recordNum
     *          记录序列号，从1开始，每个worker上单独计数。
     * @param record
     *          输入表中的记录，三列，分别表示初点、终点、边的权重。
     * @param context
     *          上下文，请求将解释后的边添加到图中。
     **/
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
        throws IOException {
      if (isEdgeData) {
```

```
        /**
         * 数据来源于存储边信息的表。
         *
         * 1、第一列表示初始点的ID。
         **/
        LongWritable sourceVertexID = (LongWritable) record.get(0);
        /**
         * 2、第二列表示终点的ID。
         **/
        LongWritable destinationVertexID = (LongWritable) record.get(1);
        /**
         * 3、第三列表示边的权重。
         **/
        LongWritable edgeValue = (LongWritable) record.get(2);
        /**
         * 4、创建边，由终点ID和边的权重组成。
         **/
        Edge<LongWritable, LongWritable> edge = new Edge<LongWritable, LongWritable>(
            destinationVertexID, edgeValue);
        /**
         * 5、请求给初始点添加边。
         **/
        context.addEdgeRequest(sourceVertexID, edge);
        /**
         * 6、如果每条Record表示双向边，重复4与5 。
          * Edge<LongWritable, LongWritable> edge2 = new
         * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
         * context.addEdgeRequest(destinationVertexID, edge2);
         **/
      } else {
        /**
         * 数据来源于存储点信息的表。
         *
         * 1、第一列表示点的ID。
         **/
        LongWritable vertexID = (LongWritable) record.get(0);
        /**
         * 2、第二列表示点的值。
         **/
        LongWritable vertexValue = (LongWritable) record.get(1);
        /**
         * 3、创建点，由点的ID和点的值组成。
         **/
        MyVertex vertex = new MyVertex();
        /**
         * 4、初始化点。
         **/
        vertex.setId(vertexID);
        vertex.setValue(vertexValue);
        /**
         * 5、请求添加点。
         **/
        context.addVertexRequest(vertex);
      }
    }
  }
  /**
   * 汇总GraphLoader::load(LongWritable, Record, MutationContext)生成的键值对，类似于
   * com.aliyun.odps.mapreduce.Reducer中的reduce。对于唯一的Vertex ID，所有关于这个ID上
   * 添加\删除、点\边的行为都会放在VertexChanges中。
   *
   * 注意：此处并不只针对load方法中添加的有冲突的点或边才调用（冲突是指添加多个相同Vertex对象，添加重复边等），
   * 所有在load方法中请求生成的ID都会在此处被调用。
   **/
  public static class LoadingResolver extends
      VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
    /**
     * 处理关于一个ID的添加或删除、点或边的请求。
     *
     * VertexChanges有四个接口，分别与MutationContext的四个接口对应：
     * VertexChanges::getAddedVertexList()与
     * MutationContext::addVertexRequest(Vertex)对应，
     * 在load方法中，请求添加的ID相同的Vertex对象，会被汇总在返回的List中。
     * VertexChanges::getAddedEdgeList()与
     * MutationContext::addEdgeRequest(WritableComparable, Edge)
```

```
     * 对应，请求添加的初始点ID相同的Edge对象，会被汇总在返回的List中。
     * VertexChanges::getRemovedVertexCount()与
     * MutationContext::removeVertexRequest(WritableComparable)
     * 对应，请求删除的ID相同的Vertex，汇总的请求删除的次数作为返回值。
     * VertexChanges#getRemovedEdgeList()与
     * MutationContext#removeEdgeRequest(WritableComparable, WritableComparable)
     * 对应，请求删除的初始点ID相同的Edge对象，会被汇总在返回的List中。
     *
     * 用户通过处理关于这个ID的变化，通过返回值声明此ID是否参与计算，如果返回的Vertex不为null，
     * 则此ID会参与随后的计算，如果返回null，则不会参与计算。
     *
     * @param vertexId
     *          请求添加的点的ID，或请求添加的边的初点ID。
     * @param vertex
     *          已存在的Vertex对象，数据载入阶段，始终为null。
     * @param vertexChanges
     *          此ID上的请求添加\删除、点\边的集合。
     * @param hasMessages
     *          此ID是否有输入消息，数据载入阶段，始终为false。
     **/
    @Override
    public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve(
        LongWritable vertexId,
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
        VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable> vertexChanges,
        boolean hasMessages) throws IOException {
      /**
       * 1、获取Vertex对象，作为参与计算的点。
       **/
      MyVertex computeVertex = null;
      if (vertexChanges.getAddedVertexList() == null
          || vertexChanges.getAddedVertexList().isEmpty()) {
        computeVertex = new MyVertex();
        computeVertex.setId(vertexId);
      } else {
        /**
         * 此处假设存储点信息的表中，每个Record表示唯一的点。
         **/
        computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);
      }
      /**
       * 2、将请求给此点添加的边，添加到Vertex对象中，如果数据有重复的可能，根据算法需要决定是否去重。
       **/
      if (vertexChanges.getAddedEdgeList() != null) {
        for (Edge<LongWritable, LongWritable> edge : vertexChanges
            .getAddedEdgeList()) {
          computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
        }
      }
      /**
       * 3、将Vertex对象返回，添加到最终的图中参与计算。
       **/
      return computeVertex;
    }
  }
  /**
   * 确定参与计算的Vertex的行为。
   *
   **/
  public static class MyVertex extends
      Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
    /**
     * 将vertex的边，按照输入表的格式再写到结果表。输入表与输出表的格式和数据都相同。
     *
     * @param context
     *          运行时上下文。
     * @param messages
     *          输入消息。
     **/
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
        Iterable<LongWritable> messages) throws IOException {
      /**
       * 将点的ID和值，写到存储点的结果表。
```

```
       **/
      context.write("vertex", getId(), getValue());
      /**
       * 将点的边，写到存储边的结果表。
       **/
      if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
          context.write("edge", getId(), edge.getDestVertexId(),
              edge.getValue());
        }
      }
      /**
       * 只迭代一轮。
       **/
      voteToHalt();
    }
  }
  /**
   * @param args
   * @throws IOException
   */
  public static void main(String[] args) throws IOException {
    if (args.length < 4) {
      throw new IOException(
          "Usage: VertexInputFormat <vertex input> <edge input> <vertex output> <edge output>");
    }
    /**
     * GraphJob用于对Graph作业进行配置。
     */
    GraphJob job = new GraphJob();
    /**
     * 1、指定输入的图数据，并指定存储边数据所在的表。
     */
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addInput(TableInfo.builder().tableName(args[1]).build());
    job.set(EDGE_TABLE, args[1]);
    /**
     * 2、指定载入数据的方式，将Record解释为Edge，此处类似于map，生成的key为vertex的ID，value为edge。
     */
    job.setGraphLoaderClass(VertexInputLoader.class);
    /**
     * 3、指定载入数据阶段，生成参与计算的vertex。此处类似于reduce，将map生成的edge合并成一个vertex。
     */
    job.setLoadingVertexResolverClass(LoadingResolver.class);
    /**
     * 4、指定参与计算的vertex的行为。每轮迭代执行vertex.compute方法。
     */
    job.setVertexClass(MyVertex.class);
    /**
     * 5、指定图作业的输出表，将计算生成的结果写到结果表中。
     */
    job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
    job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());
    /**
     * 6、提交作业执行。
     */
    job.run();
  }
}
```

12.8.11. 输入边表

本文为您提供输入边表的代码。

示例代码如下。

```
import java.io.IOException;
import com.aliyun.odps.conf.Configuration;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.Vertex;
```

```
import com.aliyun.odps.graph.VertexResolver;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableComparable;
import com.aliyun.odps.io.WritableRecord;
/**
 * 本示例是用于展示，对于不同类型的数据类型，如何编写图作业程序载入数据。主要展示GraphLoader和
 * VertexResolver配合完成图的构建。
 *
 * ODPS Graph的作业输入都为ODPS的Table，假设作业输入有两张表，一张存储点的信息，一张存储边的信息。
 * 存储点信息的表的格式，如下所示。
 * +------------------------+
 * | VertexID | VertexValue |
 * +------------------------+
 * |       id0|            9|
 * +------------------------+
 * |       id1|            7|
 * +------------------------+
 * |       id2|            8|
 * +------------------------+
 *
 * 存储边信息的表的格式，举例如下。
 * +-----------------------------------+
 * | VertexID | DestVertexID| EdgeValue|
 * +-----------------------------------+
 * |       id0|          id1|         1|
 * +-----------------------------------+
 * |       id0|          id2|         2|
 * +-----------------------------------+
 * |       id2|          id1|         3|
 * +-----------------------------------+
 *
 * 结合两张表的数据，表示id0有两条出边，分别指向id1和id2；id2有一条出边，指向id1；id1没有出边。
 *
 * 对于此种类型的数据，在GraphLoader::load(LongWritable, Record, MutationContext)，可以使用MutationContext#addVertexRequest(Vertex)向
图中请求添加点，使用
 * link MutationContext#addEdgeRequest(WritableComparable, Edge)向图中请求添加边，然后，在
 * link VertexResolver#resolve(WritableComparable, Vertex, VertexChanges, boolean)
 * 中，将load方法中添加的点和边合并到一个Vertex对象中，作为返回值，添加到最后参与计算的图中。
 *
 **/
public class VertexInputFormat {
  private final static String EDGE_TABLE = "edge.table";
  /**
   * 将Record解释为Vertex和Edge，每个Record根据其来源，表示一个Vertex或者一条Edge。
   * <p>
   * 类似于com.aliyun.odps.mapreduce.Mapper#map，
   * 输入Record，生成键值对，此处的键是Vertex的ID，
   * 值是Vertex或Edge，通过上下文Context写出，这些键值对会在LoadingVertexResolver出根据Vertex的ID汇总。
   *
   * 注意：此处添加的点或边只是根据Record内容发出的请求，并不是最后参与计算的点或边，只有在随后的VertexResolver
   * 中添加的点或边才参与计算。
   **/
  public static class VertexInputLoader extends
      GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
    private boolean isEdgeData;
    /**
     * 配置VertexInputLoader。
     *
     * @param conf
     *          作业的配置参数，在main中使用GraphJob配置的，或者在客户端中使用set命令配置。
     * @param workerId
     *          当前工作的worker的序号，从0开始，可以用于构造唯一的vertex id。
     * @param inputTableInfo
     *          当前worker载入的输入表信息，可以用于确定当前输入是哪种类型的数据，即Record的格式。
     **/
    @Override
    public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
      isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableName());
    }
    /**
     * 根据Record中的内容，解析为对应的边，并请求添加到图中。
     *
     * @param recordNum
```

```
     * @param recordNum
     *          记录序列号，从1开始，每个worker上单独计数。
     * @param record
     *          输入表中的记录，三列分别表示初点、终点、边的权重。
     * @param context
     *          上下文，请求将解释后的边添加到图中。
     **/
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
        throws IOException {
      if (isEdgeData) {
        /**
         * 数据来源于存储边信息的表。
         *
         * 1、第一列表示初始点的ID。
         **/
        LongWritable sourceVertexID = (LongWritable) record.get(0);
        /**
         * 2、第二列表示终点的ID。
         **/
        LongWritable destinationVertexID = (LongWritable) record.get(1);
        /**
         * 3、第三列表示边的权重。
         **/
        LongWritable edgeValue = (LongWritable) record.get(2);
        /**
         * 4、创建边，由终点ID和边的权重组成。
         **/
        Edge<LongWritable, LongWritable> edge = new Edge<LongWritable, LongWritable>(
            destinationVertexID, edgeValue);
        /**
         * 5、请求给初始点添加边。
         **/
        context.addEdgeRequest(sourceVertexID, edge);
        /**
         * 6、如果每条Record表示双向边，重复4与5。
          * Edge<LongWritable, LongWritable> edge2 = new
         * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
         * context.addEdgeRequest(destinationVertexID, edge2);
         **/
      } else {
        /**
         * 数据来源于存储点信息的表。
         *
         * 1、第一列表示点的ID。
         **/
        LongWritable vertexID = (LongWritable) record.get(0);
        /**
         * 2、第二列表示点的值。
         **/
        LongWritable vertexValue = (LongWritable) record.get(1);
        /**
         * 3、创建点，由点的ID和点的值组成。
         **/
        MyVertex vertex = new MyVertex();
        /**
         * 4、初始化点。
         **/
        vertex.setId(vertexID);
        vertex.setValue(vertexValue);
        /**
         * 5、请求添加点。
         **/
        context.addVertexRequest(vertex);
      }
    }
  }
  /**
   * 汇总GraphLoader::load(LongWritable, Record, MutationContext)生成的键值对，类似于
   * com.aliyun.odps.mapreduce.Reducer中的reduce。对于唯一的Vertex ID，所有关于这个ID上
   * 添加\删除、点\边的行为都会放在VertexChanges中。
   *
   * 注意：此处并不只针对load方法中添加的有冲突的点或边才调用（冲突是指添加多个相同Vertex对象，添加重复边等）。
```

```
    注意：此处开始计算load方法中添加的有序关的点或边方调用（序关是指添加多个相同Vertex对象，添加重复边等），
  * 所有在load方法中请求生成的ID都会在此处被调用。
  **/
 public static class LoadingResolver extends
     VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
   /**
    * 处理关于一个ID的添加或删除、点或边的请求。
    *
    * VertexChanges有四个接口，分别与MutationContext的四个接口对应：
    * VertexChanges::getAddedVertexList()与
    * MutationContext::addVertexRequest(Vertex)对应，
    * 在load方法中，请求添加的ID相同的Vertex对象，会被汇总在返回的List中。
    * VertexChanges::getAddedEdgeList()与
    * MutationContext::addEdgeRequest(WritableComparable, Edge)
    * 对应，请求添加的初始点ID相同的Edge对象，会被汇总在返回的List中。
    * VertexChanges::getRemovedVertexCount()与
    * MutationContext::removeVertexRequest(WritableComparable)
    * 对应，请求删除的ID相同的Vertex，汇总的请求删除的次数作为返回值。
    * VertexChanges#getRemovedEdgeList()与
    * MutationContext#removeEdgeRequest(WritableComparable, WritableComparable)
    * 对应，请求删除的初始点ID相同的Edge对象，会被汇总在返回的List中。
    *
    * 用户通过处理关于这个ID的变化，通过返回值声明此ID是否参与计算，如果返回的Vertex不为null，
    * 则此ID会参与随后的计算，如果返回null，则不会参与计算。
    *
    * @param vertexId
    *          请求添加的点的ID，或请求添加的边的初点ID。
    * @param vertex
    *          已存在的Vertex对象，数据载入阶段，始终为null。
    * @param vertexChanges
    *          此ID上的请求添加\删除、点\边的集合。
    * @param hasMessages
    *          此ID是否有输入消息，数据载入阶段，始终为false。
    **/
   @Override
   public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve(
       LongWritable vertexId,
       Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
       VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable> vertexChanges,
       boolean hasMessages) throws IOException {
     /**
      * 1、获取Vertex对象，作为参与计算的点。
      **/
     MyVertex computeVertex = null;
     if (vertexChanges.getAddedVertexList() == null
         || vertexChanges.getAddedVertexList().isEmpty()) {
       computeVertex = new MyVertex();
       computeVertex.setId(vertexId);
     } else {
       /**
        * 此处假设存储点信息的表中，每个Record表示唯一的点。
        **/
       computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);
     }
     /**
      * 2、将请求给此点添加的边添加到Vertex对象中，如果数据有重复的可能，根据算法需要决定是否去重。
      **/
     if (vertexChanges.getAddedEdgeList() != null) {
       for (Edge<LongWritable, LongWritable> edge : vertexChanges
           .getAddedEdgeList()) {
         computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
       }
     }
     /**
      * 3、将Vertex对象返回添加到最终的图中参与计算。
      **/
     return computeVertex;
   }
 }
 /**
  * 确定参与计算的Vertex的行为。
  *
  **/
 public static class MyVertex extends
     Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
   /**
```

```
     * 将vertex的边，按照输入表的格式再写到结果表。输入表与输出表的格式和数据都相同。
     *
     * @param context
     *          运行时上下文。
     * @param messages
     *          输入消息。
     **/
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
        Iterable<LongWritable> messages) throws IOException {
      /**
       * 将点的ID和值写到存储点的结果表。
       **/
      context.write("vertex", getId(), getValue());
      /**
       * 将点的边写到存储边的结果表。
       **/
      if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
          context.write("edge", getId(), edge.getDestVertexId(),
              edge.getValue());
        }
      }
      /**
       * 只迭代一轮。
       **/
      voteToHalt();
    }
  }
  /**
   * @param args
   * @throws IOException
   */
  public static void main(String[] args) throws IOException {
    if (args.length < 4) {
      throw new IOException(
          "Usage: VertexInputFormat <vertex input> <edge input> <vertex output> <edge output>");
    }
    /**
     * GraphJob用于对Graph作业进行配置。
     */
    GraphJob job = new GraphJob();
    /**
     * 1、指定输入的图数据，并指定存储边数据所在的表。
     */
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addInput(TableInfo.builder().tableName(args[1]).build());
    job.set(EDGE_TABLE, args[1]);
    /**
     * 2、指定载入数据的方式，将Record解释为Edge，此处类似于map，生成的key为vertex的ID，value为edge。
     */
    job.setGraphLoaderClass(VertexInputLoader.class);
    /**
     * 3、指定载入数据阶段，生成参与计算的vertex。此处类似于reduce，将map生成的edge合并成一个vertex。
     */
    job.setLoadingVertexResolverClass(LoadingResolver.class);
    /**
     * 4、指定参与计算的vertex的行为。每轮迭代执行vertex.compute方法。
     */
    job.setVertexClass(MyVertex.class);
    /**
     * 5、指定图作业的输出表，将计算生成的结果写到结果表中。
     */
    job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
    job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());
    /**
     * 6、提交作业执行。
     */
    job.run();
  }
}
```

13.查看作业运行信息

13.1. 使用Logview查看作业运行信息

Logview是MaxCompute Job提交后查看和Debug任务的工具。

通过Logview可看到一个Job的如下内容：

- 任务的运行状态。
- 任务的运行结果。
- 任务的细节和每个步骤的进度。

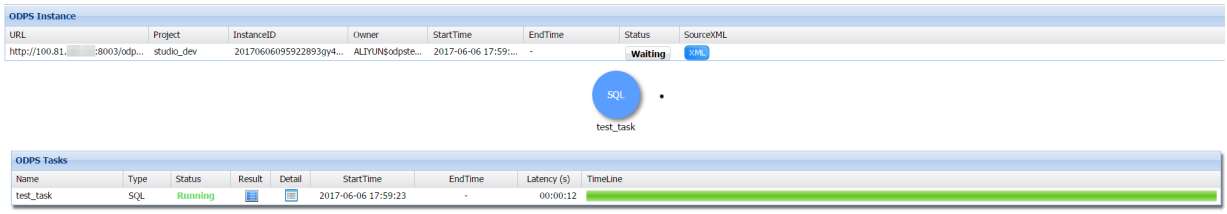
Job提交到MaxCompute后，会生成Logview的链接。您可以直接在浏览器上打开Logview链接，进入查看Job的信息。

```
odps@ test_workshop001>select * from result_test;

ID = 20190917055240226ga6e5mim
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcomp
m/api&p=test_workshop001&i=20190917055240226ga6e5mim&token=YkxhUzJQ
1NIamRWNDBBU3UJPSxPRFBTX09CTzoxMDc5OTI2ODk2OTk5NDIxLDE1NjkzMDQzNjAs
nQiOlt7IkFjdG1vbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ciLCJSZXNv
3M6b2RwczoqOnByb2plY3RzL3R1c3Rfd29ya3Nob3AwMDEvaW5zdGFuY2UvLzIwMTkw
jI2Z2E2ZTUtaW0iXX1dLCJWZXJzaW9uIjoiMSJ9
Job Queueing...
Summary:

+------------+------------+
| education  | num        |
+------------+------------+
```

> **说明** 每个Job的Logview页面的有效期为一周。

Logview功能组件

下面结合具体的Logview Web UI界面，为您介绍每个组件的含义。



Logview的首页分成上下两部分：

- Instance信息
- Task信息

Instance信息

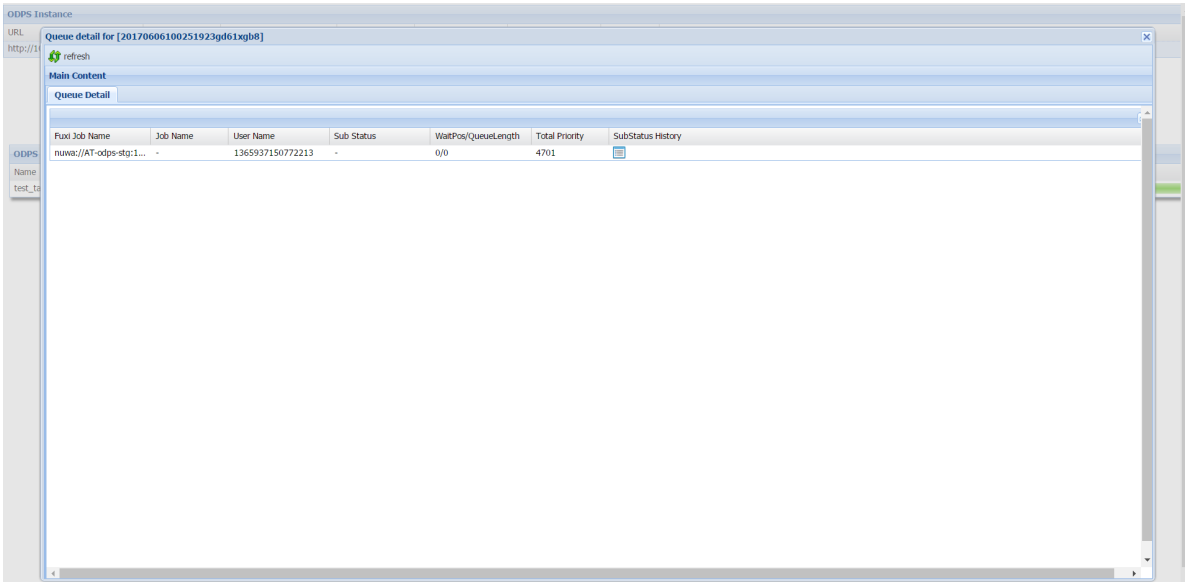
在Logview页面中，上半部分是您提交的SQL任务对应的MaxCompute Instance信息，包含URL链接、项目、InstanceID、提交人、开始时间、结束时间和状态（Status）等。

- Instance信息部分，您可以单击Status查看当前队列的Status详细信息，包含四种状态：
  - Waiting：说明当前作业正在MaxCompute中处理，并没有提交到分布式调度系统（Fuxi）中运行。
  - Waiting List：n：说明作业已提交至分布式调度系统（Fuxi）排队等待运行，当前在队列中处于第n位。
  - Running：作业在分布式调度系统（Fuxi）中运行。

  > **说明** 处于Terminated状态的Instance由于作业已处理完成，因此不存在队列信息。

- 单击Status查看队列详细信息：
  - Sub Status：表示当前子状态信息。
  - WaitPos：表示排队位置，如果是 `0` 表示正在运行，如果为 `-` 表示尚未到Fuxi。
  - QueueLength：表示Fuxi中总的队列长度。
  - Total Priority：表示作业运行时经过系统判断后授予的优先级。

- SubStatus History：单击后，可以查看作业执行的详细历史状态，包含状态码、状态描述、开始时间、持续时间等（某些版本暂时无历史信息）。

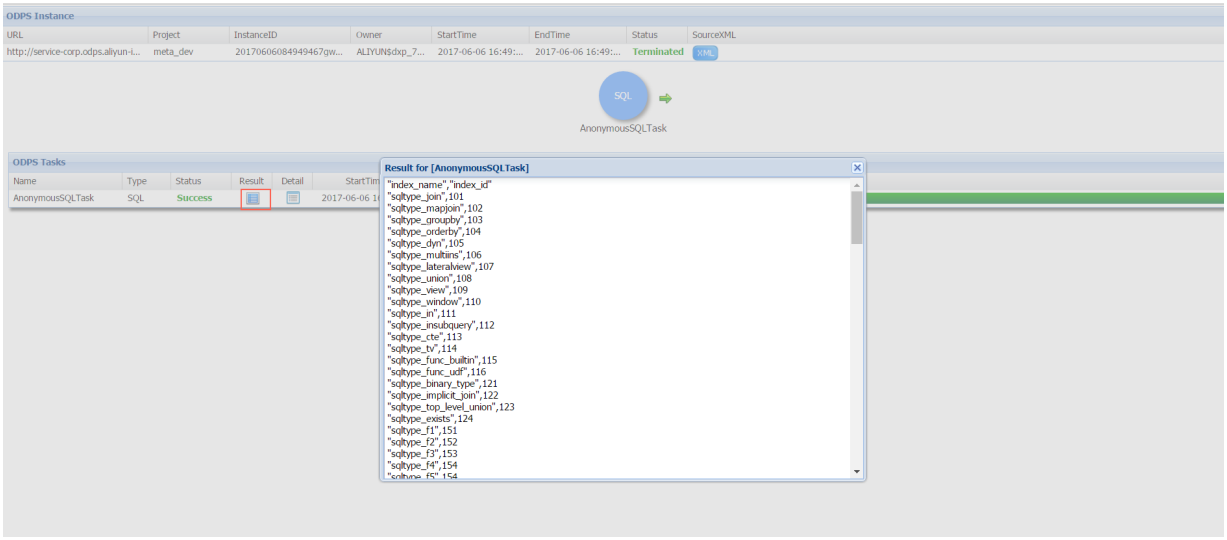


Task 信息

在Logview页面中，下半部分为该Instance对应的task信息，包括任务名称、类型、状态、运行结果（Result）、详细信息（Detail）、开始时间、结束时间、总运行时间（Latency）、进度。其中，Latency指运行总共消耗的时间，与其他页面的Latency含义类似。

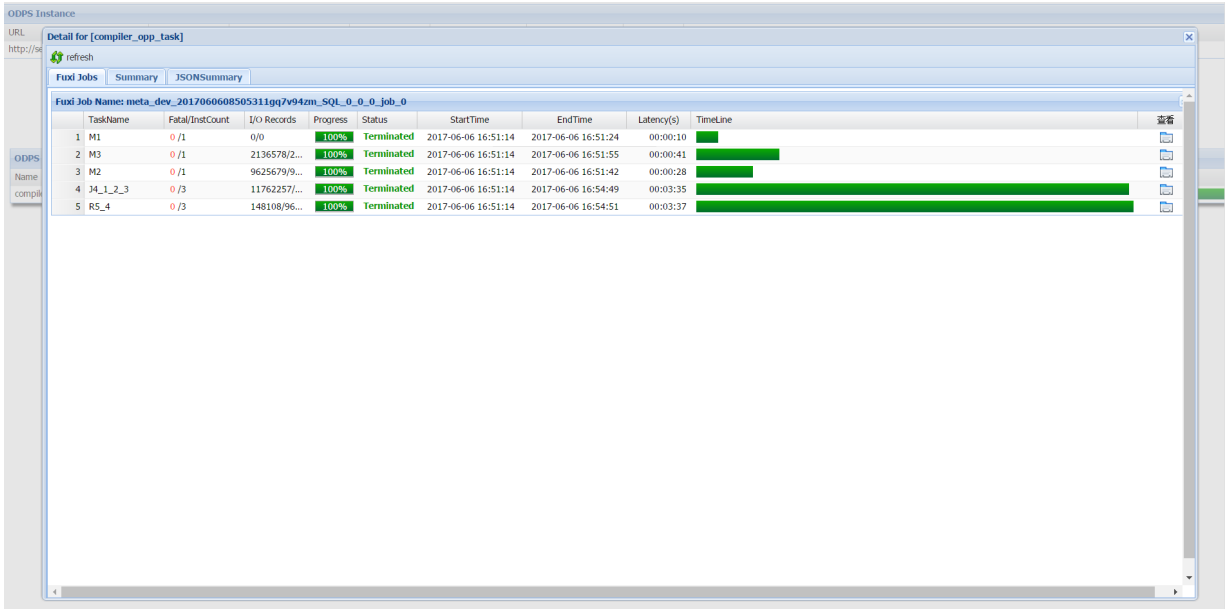
Result：

在Job运行结束后，可以看到运行结果。例如一条select SQL的结果如下图所示。



Detail：

一个Job在运行中和结束后，均可以单击**Detail**来查看任务运行的具体情况。



在MaxCompute Task详细信息对话框中，您可以查看以下信息：

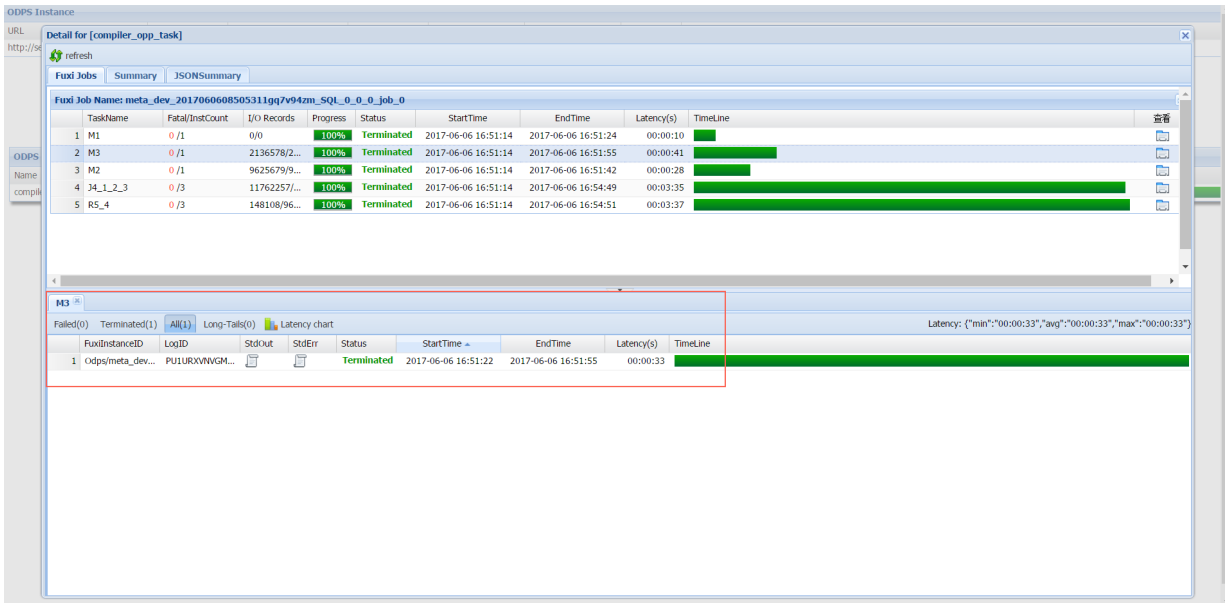
- 一个MaxCompute Task可以由一个或多个Fuxi Job组成。例如，如果您提交的SQL任务较为复杂，MaxCompute将自动向分布式调度系统（Fuxi）提交多个Fuxi Job。
- 每个Fuxi Job可以由一个或多个Fuxi Task组成。例如，简单的MapReduce任务通常会产生两个Fuxi Task，即Map Task（M1）和Reduce Task（R2）。而一个比较复杂的SQL任务也可能产生多个Fuxi Task。
- 查看所产生的Fuxi Task的任务名称。任务名称一般由字母和数字组成，其中字母表示任务类型（例如，M代表Map Task），数字标识任务编号以及依赖关系（例如，R5_4表示该Reduce Task需要J4任务执行结束后才会开始执行；J4_1_2_3则表示该Join Task需要M1、M2、M3这三个任务都执行完成后才会开始执行）。

  > **说明** I/O Records表示该Fuxi Task的输入和输出的记录数。

- I/O Records表示这个task的输入和输出的records数。

单击Fuxi Task最右侧的查看列中的按钮或双击该Fuxi Task，您可以查看该Fuxi Task对应的Instance信息。

> **说明** 每个Fuxi Task由一个或多个Fuxi Instance组成。当Fuxi Task的输入数据量持续增大时，MaxCompute将自动为该Task启动更多的节点来处理数据，每个节点即对应一个Fuxi Instance。



在页面下方的Fuxi Instance信息部分，Logview将处于不同阶段的Instance进行分组展示。例如，您可以选择Failed组，查看运行出错的节点。同时，您可以单击StdOut和StdErr列中的按钮查看标准输出信息和标准错误信息。

> **说明** 您在所提交的MaxCompute任务中编写的打印信息也将在标准输出信息和标准错误信息中展示。

使用Logview排查问题

- **出错的任务**

  当有任务出错时，您可以在Logview页面的Result中看到错误的提示信息，也可以在Detail页面中通过Fuxi Instance的stderr，查看具体某个Instance出错的信息。

- **数据倾斜**

  运行缓慢有时是由于在某个Fuxi Task的所有Fuxi Instance中，有个别Instance形成长尾造成的。长尾的现象由同一个Task内任务分配不均导致。这时可以在任务运行完后，在Summay标签页中看运行结果。在每个Task中都可以看到如下输出：

```
output records:
R2_1_Stg1: 199998999  (min: 22552459, max: 177446540, avg: 99999499)
```

  此处如果min和max值相差较大，不在一个数量级，说明在这一阶段出现了数据倾斜。例如在执行字段Join时，某个字段中有一个值出现的比例很高，则在这一字段上执行Join就会出现数据倾斜。

13.2. 使用Logview 2.0查看Job运行信息

本文为您介绍Logview 2.0的入口及功能。您可以通过Logview 2.0 URL查看作业运行信息。

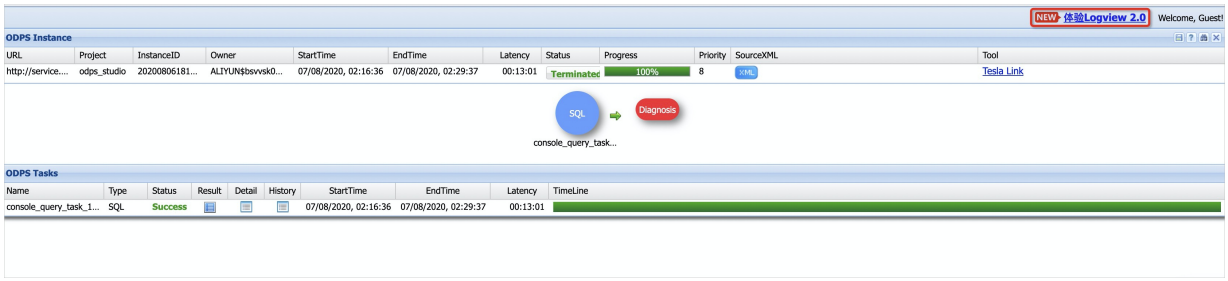
概述

Logview 2.0在Logview基础上，进行了全新的页面设计，优化了加载速度，并新增了如下功能：

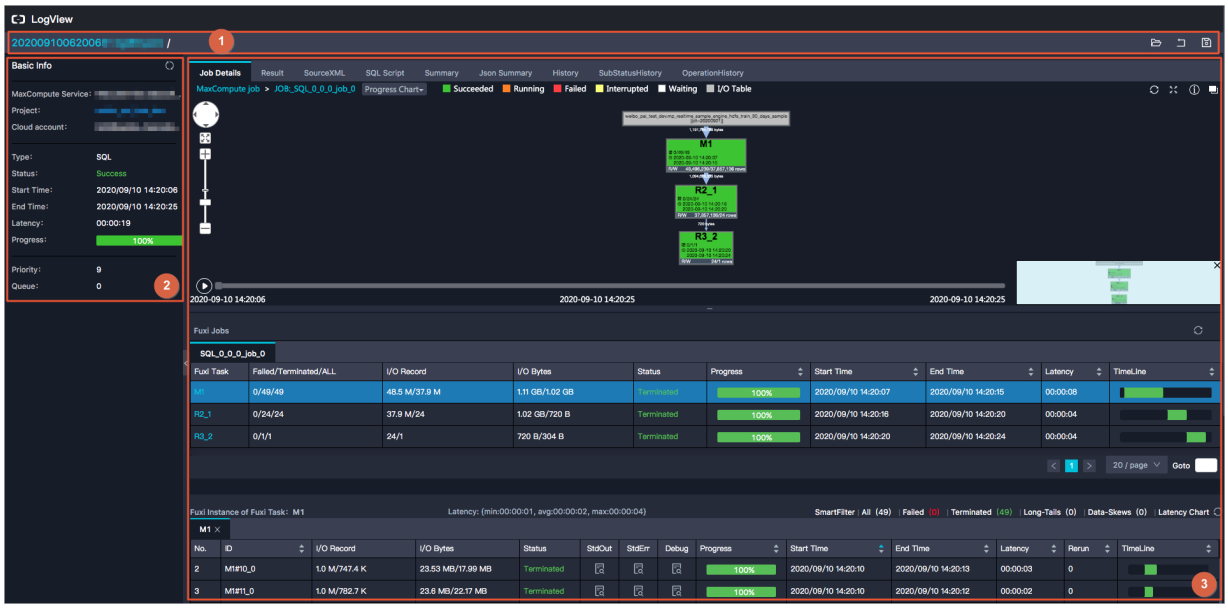
- 支持以交互式DAG图展示作业处理逻辑架构，您还可以查看相应的Operation层级。
- 支持回放作业运行过程。
- 支持通过Fuxi Sensor查看内存及CPU使用情况。

入口

您使用MaxCompute客户端（odpscmd）提交作业后，系统会生成Logview URL。在搜索引擎中打开Logview URL界面，单击**体验Logview 2.0**即可进入Logview 2.0界面。



Logview 2.0界面如下。



序号	区域
①	标题与功能区，详情请参见标题和功能区。
②	Basic Info，详情请参见Basic Info。
③	作业详情，详情请参见作业详情。

标题和功能区

标题与功能区包含您提交MaxCompute作业时生成的唯一作业ID及您自定义的作业名称（通过SDK方式提交的作业包含作业名称信息）。您还可以执行如下操作。

图标	功能
	打开本地保存的作业详情文件Logview_detail.txt。
	返回Logview旧版界面。
	将作业详情文件保存至本地设备。

Basic Info

Basic Info区域展示的作业基本信息如下。

参数名	描述
MaxCompute Service	作业使用的MaxCompute服务的Endpoint。Endpoint详情请参见Endpoint。
Project	作业所属的MaxCompute项目名称。
Cloud account	提交作业的阿里云账号信息。
Type	作业的类型。例如SQL、SQLRT、LOT、XLib、CUPID、AlgoTask和Graph。
Status	作业的状态。状态取值如下： • Success：作业执行成功。 • Failed：作业执行失败。 • Canceled：作业执行取消。 • Waiting：作业正在MaxCompute中处理，并没有提交至Fuxi中运行。 • Running：作业正在Fuxi中处理。 • Terminated：作业已执行结束。
Start Time	作业提交时间。
End Time	作业执行结束时间。
Latency	作业执行消耗的时长。
Progress	作业执行进度。
Priority	作业优先级。
Queue	作业在资源配额组内的排队位置。

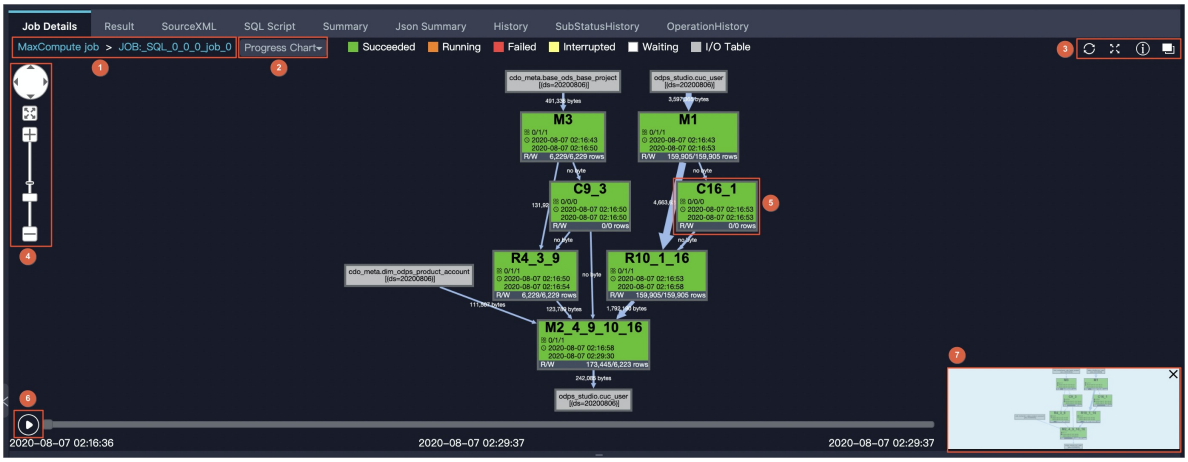
作业详情

您可以通过作业详情区域全方位了解作业，作业详情区域包含如下功能区：

- **Job Details**

- 作业执行图

  Job Detail页签的上半部分为作业执行图。执行图以可视化方式展示三个维度的子任务依赖关系：Fuxi Job层、Fuxi Task层和Operation层，同时提供一系列辅助工具，为排查问题提供帮助。界面构成如下。

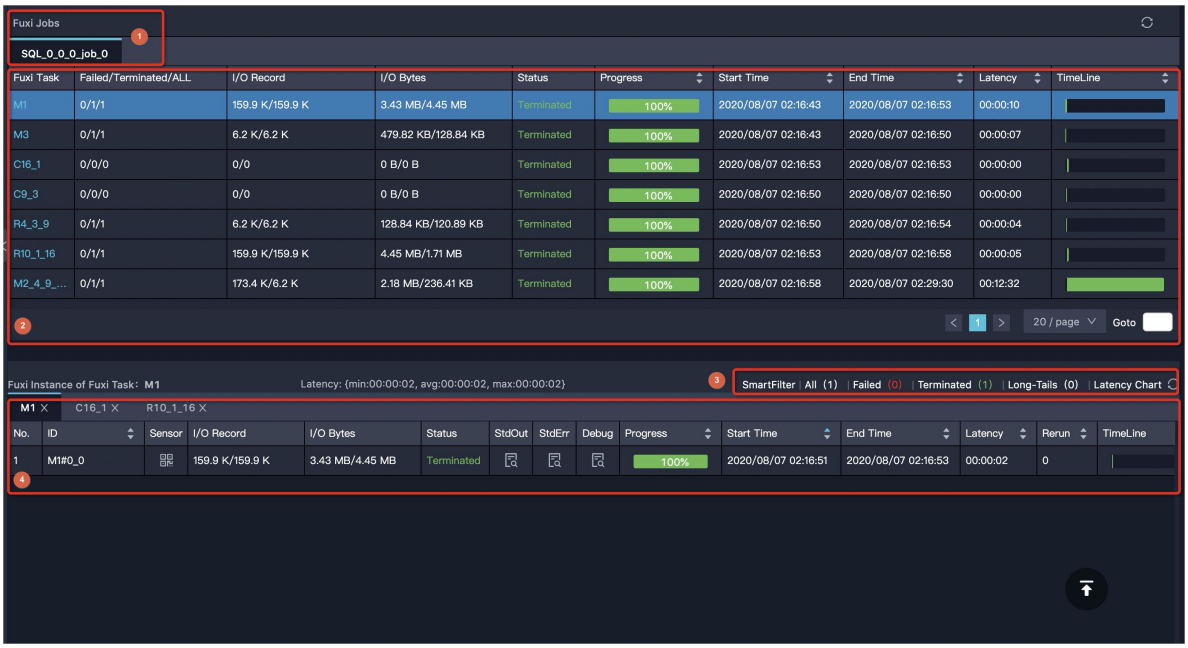


序号	描述
①	面包屑导航，用于切换Fuxi Job。JOB:_SQL_0_0_0_job_0为Fuxi Job名称。
②	排查问题辅助工具。包含Progress Chart、Input Heat Chart、Output Heat Chart、TaskTime Heart Chart和InstanceTime Heart Chart。
③	您可以在此区域刷新作业执行状态（）、全屏或缩放显示作业执行图（）、获取MaxCompute Studio文档（）和切换至任务上一层级（）。
④	缩放辅助工具。
⑤	Fuxi Task。一个MaxCompute作业由一个或多个Fuxi Job组成。每个Fuxi Job由一个或多个Fuxi Task组成。每个Fuxi Task由一个或者多个Fuxi Instance组成，当用户的输入数据量变大时，MaxCompute会在每个Task启动更多的节点来处理数据，每个节点就是一个Fuxi Instance。例如，简单的MapReduce通常会产生两个Fuxi Task，一个是Map一个是Reduce，两个Fuxi Task的名称分别为M1和R2，当SQL比较复杂时，可能会产生多个Fuxi Task。 您可以在执行界面上看到每个Fuxi Task的名称。例如M1，表示一个Map Task；R4_3_9的3、9表示它依赖M3、C9_3执行结束才能开始执行。同理，M2_4_9_10_16表示M2要依赖R4_3_9、C9_3、R10_1_16、C16_1四个Task执行结束后才能开始执行。R/W表示Task读取和写的行数。 单击某个Task或在Task上单击右键可以查看Task对应的Operator算子依赖关系及全部Operator算子图。 Logview 2.0增加了表依赖关系，您可以快速查看输入和输出表。
⑥	Fuxi Task回放。单击即可开始播放，再次单击则暂停。您也可以手动拖动进度条。进度条左边为开始时间，中间为播放时间，右边为结束时间。
⑦	鹰眼。

> **说明**
> - 不支持回放Running状态的Fuxi Task。
> - AlgoTask类型的作业（例如PAI机器学习），由于只有一个Fuxi Task，故不提供作业执行图。
> - 非SQL类型作业，仅能展示Fuxi Job和Fuxi Task层，不支持展示Operation层。
> - 如果只有一个Fuxi Job，作业执行图默认展示Fuxi Task层依赖关系；否则，默认展示Fuxi Job层依赖关系。

- 作业运行情况

  Job Detail页签的下半部分为作业运行详细信息。界面构成如下。



序号	描述
①	Fuxi Job页签，您可以在此切换Fuxi Job。
②	Fuxi Job对应的Fuxi Task详情。在某个Fuxi Job下，单击任一Fuxi Task，在下方会展开该Fuxi Task对应的Fuxi Instance信息。默认展开第一个Fuxi Job的第一个Fuxi Task的Fuxi Instance信息。 对于AlgoTask类型作业，此区域会提供**Sensor**列，您可以单击Fuxi Task对应的Sensor查看Fuxi Instance的CPU及内存信息。Fuxi Sensor详情请参见Fuxi Sensor。 **说明** Fuxi Sensor功能在西南1（成都）、华南1（深圳）、华东2（上海）、华东1（杭州）、华北3（张家口）和华北2（北京）区域已开放。
③	Logview为不同阶段的Instance进行了分组。您可以单击Failed快速查看出错的节点。
④	Fuxi Instance。 ■ StdOut和StdErr。您可以查看输出和错误信息，您自己打印的信息也可以在这里查看，同时提供下载功能。 ■ Debug。调测并排除错误。

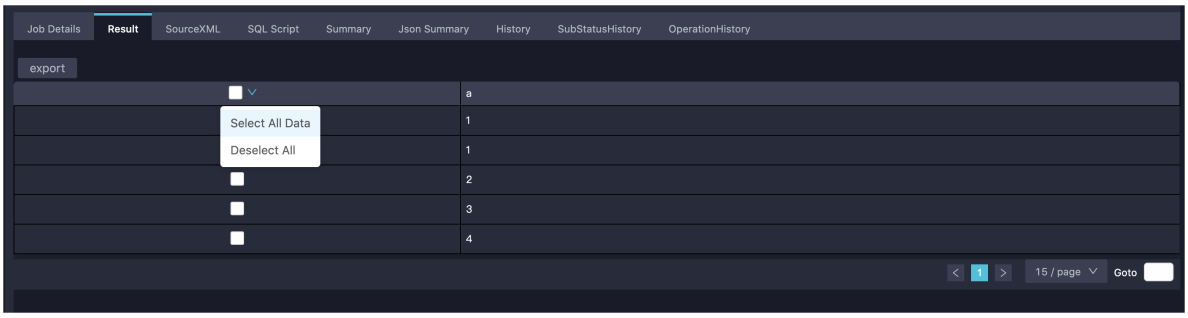
- **Result**

  Result页签用于显示作业运行结果，如果作业执行失败，系统默认会打开该页签显示失败原因。

  说明 MaxCompute支持您在项目级别通过 `odps.forbid.fetch.result.by.bearertoken` 参数，设置是否禁止Logview的**Result**页签显示作业运行结果。更多参数解释，请参见项目空间操作。

  系统会根据MaxCompute设置的返回格式，显示为表格或纯文本，您可以通过MaxCompute客户端配置odps.sql.select.output.format参数设置显示格式，如果取值为HumanReadable表示纯文本，否则默认是CSV格式。

- 表格

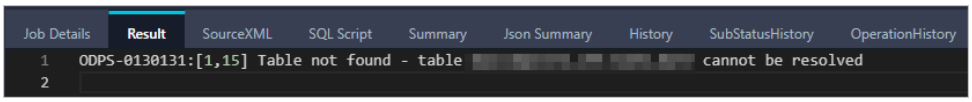


> **说明** 表格中NULL会表示为 `\N` ，既可以和NULL字符串区分，还兼容了Logview 1.0的展示方式。

您可以通过该界面执行如下操作：

- 选中某一行可以导出该行数据。
- 选中标题栏可以导出当前界面的数据。
- 在标题栏单击，选择**Select All Data**或**Deselect All**可以全选或取消全选全量数据。
- 单击**export**即可导出CSV格式的数据。CSV仅仅为文件后缀，您可以使用文本文件或Sublime Text工具打开。

- 纯文本



- **SourceXML**

  您提交至MaxCompute的作业的SourceXML信息。
- **SQL Script**

  您提交至MaxCompute的作业的SQL脚本信息。
- **Summary**

  您提交至MaxCompute的作业运行概要信息。同时可查看作业占用的内存及CPU信息。
- **Json Summary**

  以JSON格式显示的作业运行概要信息。
- **History**

  如果Fuxi Instance重新执行就会产生History信息。
- **Substatus History**

  您可以查看作业执行的详细历史状态，包含状态码、状态描述、开始时间、持续时间和结束时间。

Fuxi Sensor

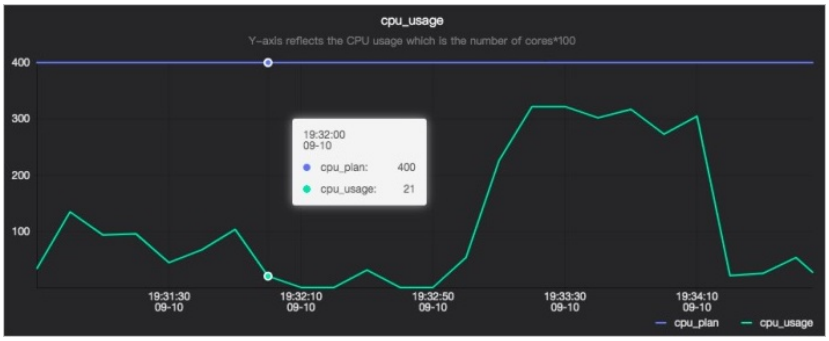
Fuxi Sensor是展示MaxCompute作业全维度的资源视图。您可以通过Fuxi Sensor查看Fuxi Instance的CPU及内存消耗量。Fuxi Sensor是作业问题定位以及作业运行质量分析的重要工具。例如以下场景的作业问题可借助Fuxi Sensor进行分析：

- 作业内存溢出时，分析实际使用的内存量。
- 对比作业申请的资源量与实际使用资源量，优化资源申请。

例如，您可以通过Fuxi Sensor查看机器学习作业Fuxi Instance的以下资源使用情况：

- CPU使用量

CPU指标包括两条线，一条显示申请的CPU量（cpu_plan），另一条显示实际的CPU使用量（cpu_usage）。纵坐标中的400，表示4个Processor。

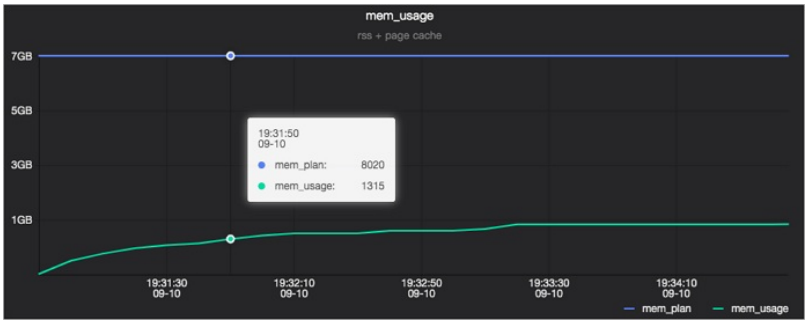


- 内存使用量

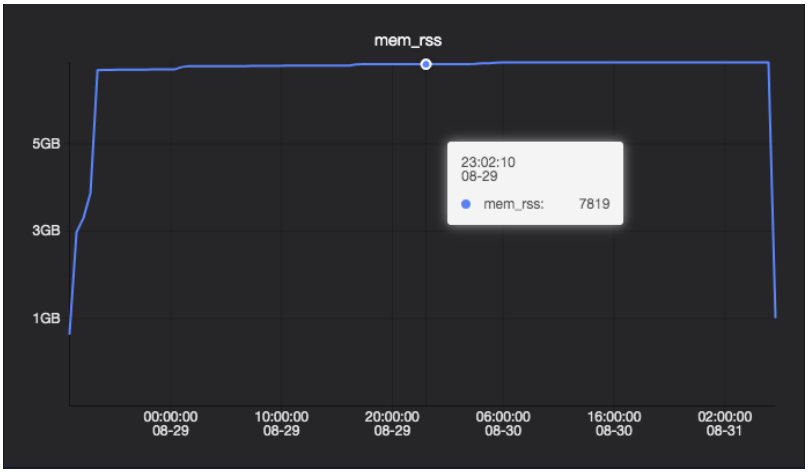
  内存指标包括两条线，一条显示申请的内存量（mem_plan），另一条显示实际的内存使用量（mem_usage）。

  内存使用量（mem_usage）包括两部分信息RSS（Resident Set Size）和PageCache。RSS是Malloc（非文件映射）发生缺页之后的内存，该部分内存，在内存紧张时无法被回收。PageCache是内核缓存文件占用的内存，例如读写日志文件，PageCache在内存紧张时通常是可以被回收。

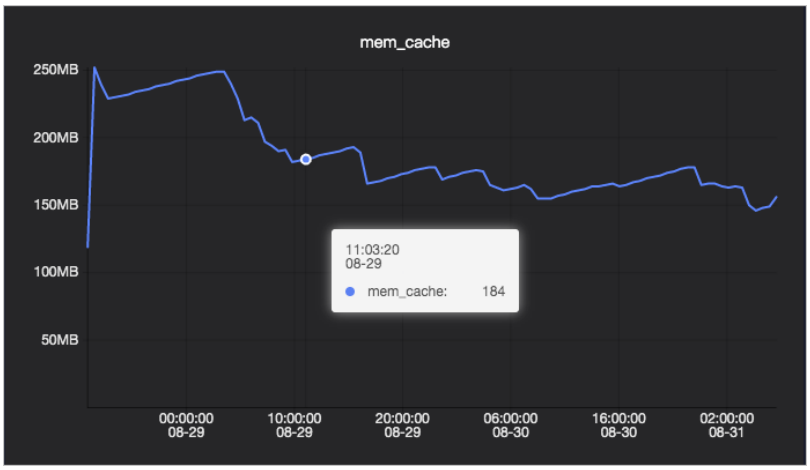
  - 内存详情

  - RSS内存使用量

- PageCache内存使用量



13.3. 使用MaxCompute编译器检查脚本

MaxCompute编译器基于MaxCompute 2.0新一代的SQL引擎，显著提升了SQL语言编译过程的易用性与语言的表达能力。本文将为您介绍如何利用MaxCompute编译器的报错和告警进行排障。

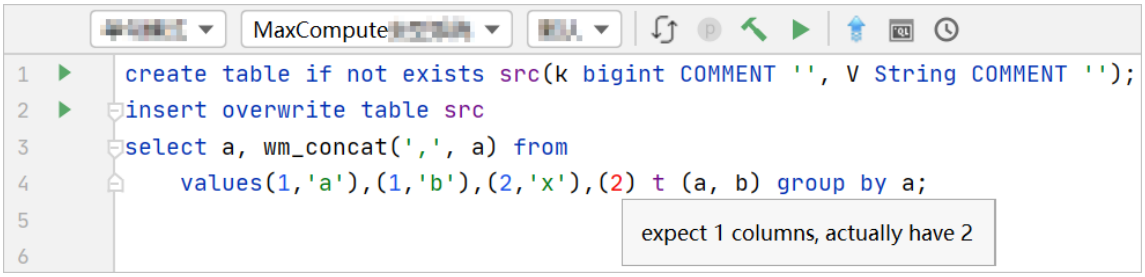
背景信息

- 为了充分展示MaxCompute编译器的易用性，推荐您使用MaxCompute Studio。如何安装请参见安装 MaxCompute Studio。
- 建议您在提交任何脚本之前，使用MaxCompute编译器对脚本进行静态编译检查。推荐您在**MaxCompute SQL**配置选项选中**Enable syntax coloring**，启动语法高亮功能。如何配置请参见MaxCompute SQL配置选项。

报错示例

- 编译器显示

  鼠标悬停如下图所示红色报错标记处，编译器显示**expect 1 columns, actually have 2**。



- 运行结果

```
1> 2021-10-14 11:33:37 Status: FAILED
1> ODPS-0130071:[3,36] Semantic analysis exception - expect 1 columns, actually have 2
```

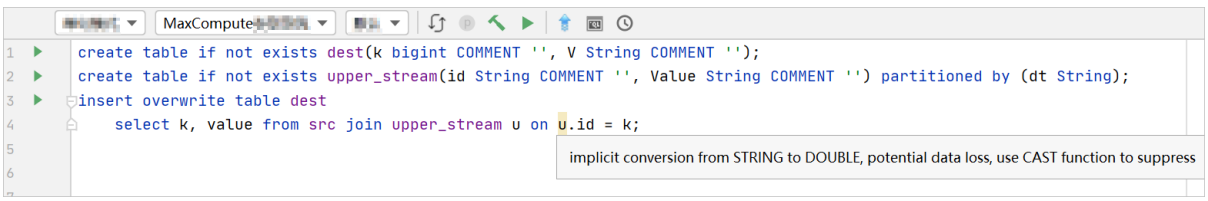
- 报错说明

  由于src表定义参数为两列，而利用insert语句插入数据时，values中有一组数据缺少一列参数，故编译器报错。

告警示例

- 编译器显示

  鼠标悬停如下图所示告警标记处，编译器显示**implicit conversion from STRING to DOUBLE, potential data loss, use CAST function to suppress**。



- 告警说明

  从STRING到DOUBLE的隐式转换，有可能造成数据丢失，需要使用**CAST**函数处理。数据转换详情请参见数据类型转换。

说明

- 在运行脚本之前建议修改所有的告警，避免浪费时间和资源。
- 提交有错误的脚本会扣您的计算健康分，会导致以后提交任务的优先级下调，未来没有修改的告警也会被纳入到健康分体系。所以，充分利用MaxCompute编译器的错误和告警提示，可以避免降低优先级。

14.通过DataWorks开发调度节点

14.1. 创建ODPS SQL节点

ODPS SQL采用类似SQL的语法，适用于海量数据（TB级）但实时性要求不高的分布式处理场景。

前提条件

您在**工作空间配置**页面添加**MaxCompute**计算引擎实例后，当前页面才会显示**MaxCompute**目录。详情请参见配置工作空间。

背景信息

因为每个作业从前期准备到提交等阶段都需要花费较长时间，因此如果要求处理几千至数万笔事务的业务，您可以使用ODPS SQL顺利完成。ODPS SQL是主要面向吞吐量的OLAP应用，详情请参见与标准SQL的主要区别及解决方法。

使用限制

ODPS SQL节点的使用限制如下：

- ODPS SQL不支持单独使用set、use语句，必须和具体的SQL语句一起执行，示例如下。

```
set a=b;
create table name(id string);
```

- ODPS SQL不支持关键字（set、use）语句后单独加注释，示例如下。

```
create table name(id string);
set a=b; --注释    //ODPS SQL不支持在set语句后添加"--注释"。
create table name1(id string);
```

- ODPS SQL不支持在已完结的语句结尾加注释，示例如下。

> **说明** SQL语句后添加英文分号（;），表示语句已完结。

```
select *   --注释    //"select *"语句未完结，因此"--注释"这个注释可以添加。
from dual; --注释    //"from dual;"语句已完结，因此"--注释"这个注释不支持添加。
show tables;
```

- 数据开发与调度运行的区别如下：
  - 数据开发：合并当前任务代码内所有的关键字（set、use）语句，作为所有SQL的前置语句。
  - 调度运行：按照顺序执行。

```
set a=b;
create table name1(id string);
set c=d;
create table name2(id string);
```

运行结果如下表所示。

执行SQL	数据开发	调度运行

第一条SQL语句	`set a=b;`
	`set c=d;`
	`create table name1(id string);`
		`set a=b;`
		`create table name1(id string);`

第二条SQL语句	`set a=b;`
	`set c=d;`
	`create table name2(id string);`
		`set c=d;`
		`create table name2(id string);`

- 调度参数配置必须是 `key=value` 的格式，且（=）前后不支持空格，示例如下。

```
time={yyyymmdd hh:mm:ss} //错误
a =b //错误
```

- 如果设置bizdate、date等关键字作为调度参数变量，格式必须是yyyymmdd。如果需要其它格式，请使用其它变量名称，避免冲突，示例如下。

```
bizdate=201908 //错误，不支持。
```

- 数据开发需要查询结果，仅支持select和with起始的SQL语句，否则无结果输出。
- 执行多条SQL语句时，请用分号（;）分隔，且需要换行。

- 错误示例

```
create table1;create table2
```

- 正确示例

```
create table1;
create table2;
```

- MaxCompute 2.0扩展函数使用到新数据类型时，您需要在该函数的SQL语句前加 `set odps.sql.type.system.odps2=true;` ，并与SQL一起提交运行，以便正常使用新数据类型。
- SQL语句中添加注释时，不支持在注释中使用英文分号（;）。

  错误示例：

```
create table1;  //创建表格table1;再创建表格table2
create table2;
```

- 使用ODPS SQL节点进行SQL任务开发时，SQL代码大小不能超过200KB，SQL命令条数不能超过200条。

操作步骤

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 鼠标悬停至+新建图标，单击**MaxCompute > ODPS SQL**。

   您也可以找到相应的业务流程，右键单击**MaxCompute**，选择**新建 > ODPS SQL**。
3. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。

   > **说明** 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。

4. 单击**提交**。
5. 在节点的编辑页面，编辑并运行代码。

   新建节点成功后，编写符合语法的ODPS SQL代码，SQL语法请参见SQL概述。

   > **说明** 由于国际标准化组织发布的中国时区信息调整，通过DataWorks执行相关SQL时，日期显示某些时间段会存在时间差异：1900~1928年的日期时间差异5分52秒，1900年之前的日期时间差异9秒。

   DataWorks不允许节点代码中仅包含 `set` 语句。如果您需要运行SET语句，可以和其它SQL语句一起执行，如下所示。

```
set odps.sql.allow.fullscan=true;
select 1;
```

   SET语句的详情请参见SET操作。

   以创建一张表并向表中插入数据，查询结果为例，操作如下：

   i. 创建一张表test1。

```
CREATE TABLE IF NOT EXISTS test1
( id BIGINT COMMENT '' ,
  name STRING COMMENT '' ,
  age BIGINT COMMENT '' ,
  sex STRING COMMENT '');
```

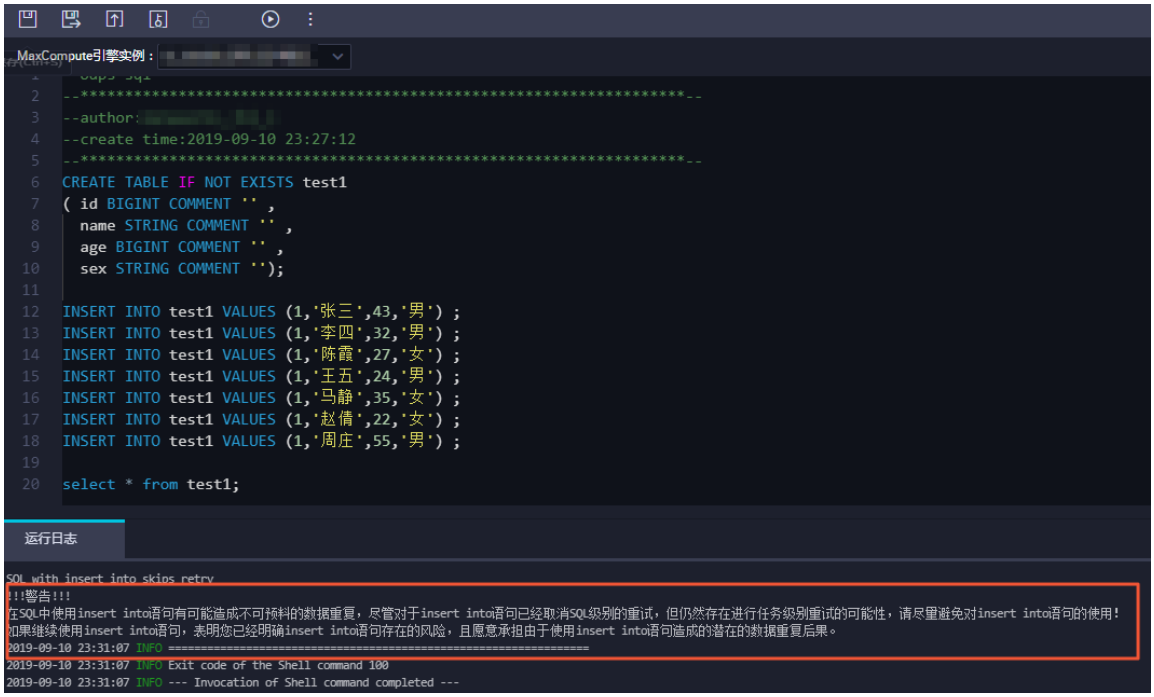
   ii. 插入准备好的数据。

```
INSERT INTO test1 VALUES (1,'张三',43,'男') ;
INSERT INTO test1 VALUES (1,'李四',32,'男') ;
INSERT INTO test1 VALUES (1,'陈霞',27,'女') ;
INSERT INTO test1 VALUES (1,'王五',24,'男') ;
INSERT INTO test1 VALUES (1,'马静',35,'女') ;
INSERT INTO test1 VALUES (1,'赵倩',22,'女') ;
INSERT INTO test1 VALUES (1,'周庄',55,'男') ;
```

   iii. 查询表数据。

```
select * from test1;
```

iv. SQL语句编辑完成后，单击工具栏中的图标，系统会按照从上往下的顺序执行SQL语句，并打印日志。



> **说明**
>
> - 如果当前工作空间绑定多个MaxCompute计算引擎，请选择需要的MaxCompute引擎实例后，再运行查询语句。
> - 如果您选中的MaxCompute引擎实例使用的是按量计费默认资源组，则可以在运行语句前，单击工具栏中的图标，预估此次运行产生的费用（实际费用请以账单为准）。

您在SQL中使用 `insert into` 语句有可能造成不可预料的数据重复。虽然已经对 `insert into` 语句取消SQL级别的重试，但仍然存在进行任务级别重试的可能性，请尽量避免使用 `insert into` 语句。如果使用，运行日志中会出现如下提示。

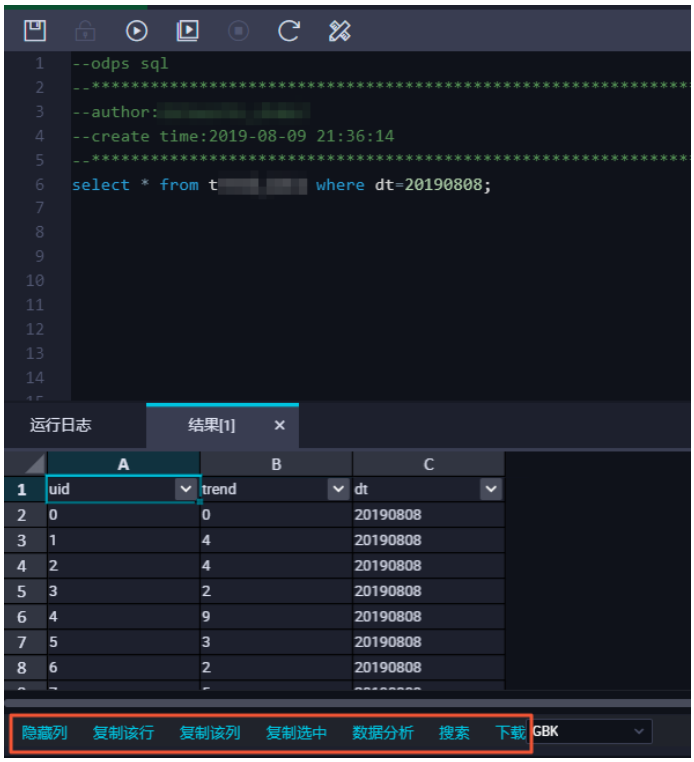
```
在SQL中使用insert into语句有可能造成不可预料的数据重复，尽管对于insert into语句已经取消SQL级别的重试，但仍然存在进行任务级别重试的可能性，请尽量避免对insert into语句的使用！
如果继续使用insert into语句，表明您已经明确insert into语句存在的风险，且愿意承担由于使用insert into语句造成的潜在的数据重复后果。
```

v. 执行无误后，单击工具栏中的图标，保存当前SQL代码。

vi. 查看执行结果。

DataWorks的查询结果接入了电子表格功能，方便您对数据结果进行操作。

查询的结果，会直接以电子表格的形式展示。您可以在DataWorks中执行操作，或者在电子表格中打开，也可以自由复制内容粘贴至本地Excel中。



操作	描述
隐藏列	选中需要隐藏的一列或多列后，单击**隐藏列**。
复制该行	左侧选中需要复制的一行或多行后，单击**复制该行**。
复制该列	顶部选中需要复制的一列或多列后，单击**复制该列**。
复制选中	选中需要复制的内容后，单击**复制选中**。
数据分析	单击**数据分析**，即可跳转至**数据分析**页面。
搜索	单击**搜索**后，在查询结果的右上角会出现搜索框，方便对表中的数据进行搜索。
下载	支持下载GBK和UTF-8两种格式。

6. 单击节点编辑区域右侧的**调度配置**，配置节点的调度属性。详情请参见配置基础属性。
7. 提交节点。

> **注意** 您需要设置节点的**重跑属性**和**依赖的上游节点**，才可以提交节点。

i. 单击工具栏中的图标。

ii. 在**提交新版本**对话框中，输入**变更描述**并选中**我确认继续执行提交操作**。

如果出现**输入输出和代码血缘分析不匹配**的告警，请确认是否忽略该告警，或是否需要调整依赖关系。详情请参见同周期调度依赖逻辑说明。

iii. 单击**确认**。

如果您使用的是标准模式的工作空间，提交成功后，请单击右上角的**发布**。具体操作请参见发布任务。

8. 测试节点，详情请参见查看并管理周期任务。

14.2. 创建SQL组件节点

SQL组件是一种带有多个输入参数和输出参数的SQL代码模板。使用SQL代码处理数据表时，通过过滤、连接和聚合源数据表，获取结果表。

前提条件

- 您需要购买DataWorks标准版及以上版本，才可以使用SQL组件节点功能。
- 您在**工作空间配置**页面添加**MaxCompute**计算引擎实例后，当前页面才会显示**MaxCompute**目录。详情请参见配置工作空间。
- 请确保您已准备好需要使用的组件，详情请参见创建组件。

背景信息

在组件的开发者发布新版本后，组件的使用者可以选择是否升级现有组件的使用实例至最新版本。

组件的版本机制支持开发者对组件不断升级，提升流程的执行效率并优化业务效果。

例如，用户A使用用户B旧版本的组件时，用户B升级组件至新版本。用户A收到组件的更新提醒后，可以根据自身需求决定是否升级。

升级根据组件模板开发的SQL组件节点时，需要单击**更新代码版本**，然后确认新版本SQL组件节点的参数配置是否继续有效。根据组件新版本的说明进行调整后，即可和普通SQL节点开发的流程一致，保存提交并进入发布流程。

操作步骤

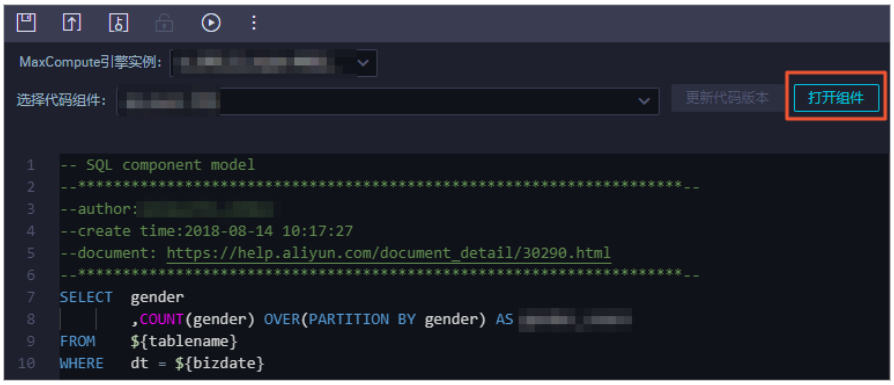
1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 鼠标悬停至**+新建**图标，单击**MaxCompute > SQL组件节点**。

   您也可以找到相应的业务流程，右键单击**MaxCompute**，选择**新建 > SQL组件节点**。
3. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。

   > **说明** 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。
4. 单击**提交**。
5. 在SQL组件节点的编辑页面，**选择代码组件**。

   如果当前工作空间绑定了多个MaxCompute计算引擎，请选择**MaxCompute引擎实例**。

   选中组件后，您可以单击**打开组件**，进入组件的详情页面。

   为提高开发效率，数据任务的开发者可以使用工作空间成员和租户成员贡献的组件，来新建数据处理节点：
   - 本工作空间成员创建的组件在**组件**下。
   - 租户成员创建的组件在**公共组件**下。
6. 单击节点编辑页面右侧的**参数配置**，为选择的组件指定参数。
7. 保存并提交节点。

   > **注意** 您需要设置节点的**重跑属性**和**依赖的上游节点**，才可以提交节点。

   i. 单击工具栏中的图标，保存节点。
   ii. 单击工具栏中的图标。
   iii. 在**提交新版本**对话框中，输入**变更描述**。
   iv. 单击**确认**。

   如果您使用的是标准模式的工作空间，提交成功后，请单击右上方的**发布**。具体操作请参见发布任务。
8. 测试节点，详情请参见查看并管理周期任务。

14.3. 创建ODPS Spark节点

ODPS Spark节点支持使用Java和Python处理数据，本文为您介绍如何新建和配置ODPS Spark节点。

背景信息

Python资源是针对Python UDF进行开发，能够获取的可以直接依赖的三方包较为有限，因此使用Python资源的局限性较大。如果您需要使用Python资源中未支持的三方包，请参见在MaxCompute UDF中运行Scipy。

PyODPS 2和PyODPS 3节点对Python资源的支持性更强大，详情请参见创建PyODPS 2节点和创建PyODPS 3节点。

本文分别为您介绍如何创建JAR资源和Python资源，并在创建ODPS Spark节点后加载不同类型的资源，您可以根据业务需求进行操作。

创建JAR资源

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 鼠标悬停至**+新建**图标，单击**MaxCompute > 资源 > JAR**。

   您也可以找到相应的业务流程，右键单击**MaxCompute**，选择**新建 > 资源 > JAR**。
3. 在**新建资源**对话框中，输入**资源名称**，并选择**目标文件夹**。

   > **说明**
   > - 如果绑定多个实例，则需要选择**MaxCompute引擎实例**。
   > - 如果该JAR包已经在MaxCompute（ODPS）客户端上传过，则需要取消勾选**上传为ODPS资源**，否则上传会报错。
   > - 资源名称无需与上传的文件名保持一致。
   > - 资源名称命名规范：1~128个字符，字母、数字、下划线、小数点，大小写不敏感，JAR资源的后缀为.jar，Python资源的后缀为.py。
4. 单击**点击上传**，选择相应的文件进行上传。

   WordCount的示例代码请参见WordCount。
5. 单击**确定**。
6. 分别单击工具栏中的和图标。
7. 在**提交新版本**对话框中，输入**变更描述**，单击**确认**。

创建Python资源

1. 在**数据开发**页面，鼠标悬停至**+新建**图标，单击**MaxCompute > 资源 > Python**。

   您也可以找到相应的业务流程，右键单击**MaxCompute**，选择**新建 > 资源 > Python**。
2. 在**新建资源**对话框中，输入**资源名称**，并选择**目标文件夹**。

   > **说明**
   > - 如果绑定多个实例，则需要选择**MaxCompute引擎实例**。
   > - 资源名称只能包含中文、字母、数字、点、下划线（_）、减号（-），且必须加后缀名.py。
3. 单击**新建**。
4. 在节点的编辑页面，输入Python代码。

   代码示例如下，仅进行校检数值判断，非数据业务处理逻辑。

```
# -*- coding: utf-8 -*-
import sys
from pyspark.sql import SparkSession
try:
    # for python 2
    reload(sys)
    sys.setdefaultencoding('utf8')
except:
    # python 3 not needed
    pass
if __name__ == '__main__':
    spark = SparkSession.builder\
        .appName("spark sql")\
        .config("spark.sql.broadcastTimeout", 20 * 60)\
        .config("spark.sql.crossJoin.enabled", True)\
        .config("odps.exec.dynamic.partition.mode", "nonstrict")\
        .config("spark.sql.catalogImplementation", "odps")\
        .getOrCreate()
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        pass
    try:
        import unicodedata
        unicodedata.numeric(s)
        return True
    except (TypeError, ValueError):
        pass
    return False
print(is_number('foo'))
print(is_number('1'))
print(is_number('1.3'))
print(is_number('-1.37'))
print(is_number('1e3'))
```

5. 分别单击工具栏中的和图标。
6. 在**提交新版本**对话框中，输入**变更描述**，单击**确认**。

创建ODPS Spark节点

1. 在**DataStudio（数据开发）**页面，鼠标悬停至+新建图标，单击**MaxCompute > ODPS Spark**。

   您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > ODPS Spark**。
2. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。

   > **说明** 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。
3. 单击**提交**。
4. 在ODPS Spark编辑页面，配置各项参数。ODPS Spark的详情请参见概述。

   ODPS Spark节点支持两种**spark版本**和**语言**。选择不同的**语言**，会显示相应不同的配置，您可以根据界面提示进行配置：
   - 选择**语言**为**Java/Scala**，配置如下。



参数	描述
spark版本	包括Spark1.x和Spark2.x两个版本。
语言	此处选择Java/Scala。
选择主jar资源	从下拉列表中选择您已上传的JAR资源。
配置项	单击**添加一条**，即可配置key和value。
Main Class	选择类名称。
参数	添加参数，多个参数之间用空格分隔。支持使用调度参数，调度参数使用方式请参考文档调度参数概述。 **说明** 您需要在配置调度参数后，再在编辑页面配置节点参数，系统会顺序执行。
选择jar资源	ODPS Spark节点根据上传的文件类型自动过滤，选择下拉框中显示的您已上传的JAR资源。
选择file资源	ODPS Spark节点根据上传的文件类型自动过滤，选择下拉框中显示的您已上传的File资源。
选择archives资源	ODPS Spark节点根据上传的文件类型自动过滤，选择下拉框中显示的您已上传的Archives资源，仅展示压缩类型的资源。

- 选择**语言**为**Python**，配置如下。



参数	描述
spark版本	包括Spark1.x和Spark2.x两个版本。
语言	此处选择Python。
选择主python资源	从下拉列表中选择您已创建的Python资源。
配置项	单击**添加一条**，即可配置key和value。

参数	描述
参数	添加参数，多个参数之间用空格分隔。
选择python资源	ODPS Spark节点根据上传的文件类型自动过滤，选择下拉框中显示的您已上传的Python资源。
选择file资源	ODPS Spark节点根据上传的文件类型自动过滤，选择下拉框中显示的您已上传的File资源。
选择archives资源	ODPS Spark节点根据上传的文件类型自动过滤，选择下拉框中显示的您已上传的Archives资源，仅展示压缩类型的资源。

5. 单击节点编辑区域右侧的**调度配置**，配置节点的调度属性，详情请参见配置基础属性。
6. 保存并提交节点。

   注意 您需要设置节点的**重跑属性**和**依赖的上游节点**，才可以提交节点。

   i. 单击工具栏中的图标，保存节点。

   ii. 单击工具栏中的图标。

   iii. 在**提交新版本**对话框中，输入**变更描述**。

   iv. 单击**确认**。

   如果您使用的是标准模式的工作空间，提交成功后，请单击右上方的**发布**。具体操作请参见发布任务。
7. 测试节点，详情请参见查看并管理周期任务。

14.4. 创建PyODPS 2节点

DataWorks提供PyODPS 2节点类型，集成了MaxCompute的Python SDK。您可以在DataWorks的PyODPS 2节点上，直接编辑Python代码，用于操作MaxCompute。

背景信息

MaxCompute提供了Python SDK方法说明，您可以使用Python的SDK来操作MaxCompute。

说明

- PyODPS 2节点底层的Python版本为2.7。
- 推荐通过SQL或者Dataframe的方式处理数据，详情请参见DataFrame概述。不建议您直接调用pandas等第三方包来处理数据。
- PyODPS 2节点获取到本地处理的数据不能超过50 MB，节点运行时占用的内存不能超过1 GB，否则节点任务会结束运行。请避免在PyODPS 2节点中写入过多的数据处理代码。
- Hints参数的详情请参见SET操作。

PyODPS 2节点主要针对MaxCompute的Python SDK应用。对于纯Python代码的执行，您可以使用Shell节点执行上传至DataWorks的Python脚本。如果您需要在PyODPS 2节点中调用第三方包，请参见在PyODPS节点中调用第三方包。

PyODPS操作实践请参见使用MaxCompute分析IP来源最佳实践和PyODPS节点实现结巴中文分词，更多信息请参见PyODPS文档。

新建PyODPS 2节点

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 鼠标悬停至+新建图标，单击**MaxCompute > PyODPS 2**。

   您也可以展开**业务流程**目录下的目标业务流程，右键单击**MaxCompute**，选择**新建 > PyODPS 2**。

   如果您需要创建业务流程，请参见创建业务流程。
3. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。

   说明 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。
4. 单击**提交**。
5. 编辑PyODPS 2节点。

i. 进入ODPS入口。

DataWorks的PyODPS 2节点中，将会包含一个全局的变量odps或o，即ODPS入口，您无需手动定义ODPS入口。

```
print(odps.exist_table('PyODPS_iris'))
```

ii. 执行SQL。

PyODPS 2支持ODPS SQL的查询，并可以读取执行的结果。execute_sql或run_sql方法的返回值是运行实例。

并非所有在MaxCompute客户端中可以执行的命令，都是PyODPS 2支持的SQL语句。调用非DDL或非DML语句时，请使用其它方法。

例如，执行GRANT、REVOKE等语句时，请使用run_security_query方法。PAI命令请使用run_xflow或execute_xflow方法。

```
o.execute_sql('select * from dual')  # 同步的方式执行，会阻塞直到SQL执行完成。
instance = o.run_sql('select * from dual')  # 异步的方式执行。
print(instance.get_logview_address())  # 获取logview地址。
instance.wait_for_success()  # 阻塞直到完成。
```

iii. 设置运行参数。

您可以通过设置hints参数，来设置运行时的参数，参数类型是dict。

```
o.execute_sql('select * from PyODPS_iris', hints={'odps.sql.mapper.split.size': 16})
```

对全局配置设置sql.settings后，每次运行时，都需要添加相关的运行时的参数。

```
from odps import options
options.sql.settings = {'odps.sql.mapper.split.size': 16}
o.execute_sql('select * from PyODPS_iris')  # 根据全局配置添加hints。
```

iv. 读取SQL执行结果。

运行SQL的实例能够直接执行open_reader的操作，有以下两种情况：

- SQL返回了结构化的数据。

```
with o.execute_sql('select * from dual').open_reader() as reader:
for record in reader:  # 处理每一个record。
```

- 可能执行的是desc等SQL语句，通过reader.raw属性，获取到原始的SQL执行结果。

```
with o.execute_sql('desc dual').open_reader() as reader:
print(reader.raw)
```

> **说明** 如果使用了自定义调度参数，页面上直接触发运行PyODPS 2节点时，需要写死时间，PyODPS节点无法直接替换。

6. 单击节点编辑区域右侧的**调度配置**，配置节点的调度属性，详情请参见配置基础属性。

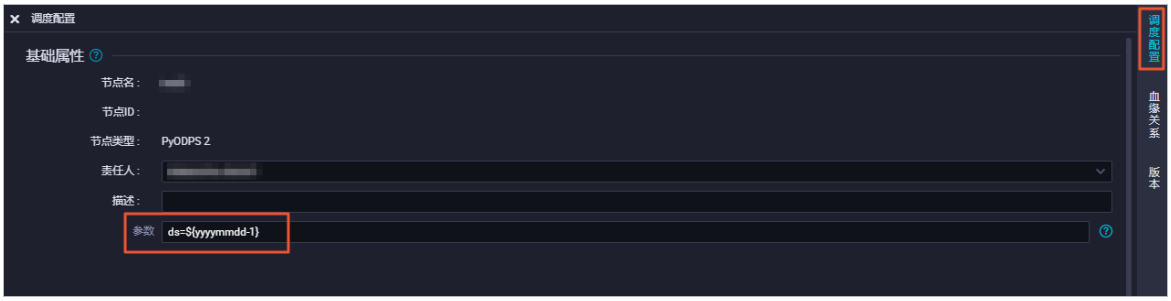
PyODPS 2节点可以使用系统定义的调度参数和自定义参数：

- 如果PyODPS 2使用系统定义的调度参数，可以直接在页面赋值。

```
a='${bdp.system.bizdate}'
print(format(a))
b='${bdp.system.cyctime}'
print(format(b))
```

> **说明** 由于公共资源组无法直接访问外网环境，建议您有公网访问需求时，使用自定义资源组或独享调度资源。仅DataWorks专业版提供自定义资源组，任意版本都可以购买独享调度资源，详情请参见DataWorks独享资源组。

- 您也可以在**调度配置 > 基础属性**区域，配置自定义参数。



> **说明** 自定义参数需要使用args['参数名']的形式调用，例如 `print (args['ds'])` 。

7. 提交节点。

   > **注意** 提交节点前，您需要设置**重跑属性**和**依赖的上游节点**。

   i. 单击工具栏中的图标。

   ii. 在**提交新版本**对话框中，输入**备注**。

   iii. 单击**确认**。

   如果您使用的是标准模式的工作空间，提交节点后，请单击右上角的**发布**。详情请参见发布任务。

8. 测试节点，详情请参见查看并管理周期任务。

PyODPS节点预装模块列表

PyODPS节点包括以下预装模块：

- setuptools
- cython
- psutil
- pytz
- dateutil
- requests
- pyDes
- numpy
- pandas
- scipy
- scikit_learn
- greenlet
- six
- 其它Python 2.7内置已安装的模块，如smtplib等。

14.5. 创建PyODPS 3节点

本文为您介绍如何创建PyODPS 3节点，以及在DataWorks使用PyODPS 3的限制。

使用限制

- 当Python 3的子版本号不同（例如Python 3.8和Python 3.7）时，字节码的定义有所不同。

  目前MaxCompute使用的Python 3版本为3.7，当使用其它版本Python 3中的部分语法（例如Python 3.8中的finally block）时，执行会报错，建议您选择Python 3.7。

- PyODPS 3节点获取到本地处理的数据不能超过50 MB，节点运行时占用的内存不能超过1 GB，否则节点任务会结束运行。请避免在PyODPS 3节点中写入过多的数据处理代码。
- PyODPS 3支持运行在公共资源组和2020年4月之后购买的独享调度资源组上。如果您的独享调度资源组的创建时间较早，请提交工单升级资源组。

创建PyODPS 3节点

1. 进入**数据开发**页面。

   i. 登录DataWorks控制台。

   ii. 在左侧导航栏，单击**工作空间列表**。

   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。

2. 鼠标悬停至 +新建 图标，单击**MaxCompute > PyODPS 3**。

   您也可以展开**业务流程**目录下的目标业务流程，右键单击**MaxCompute**，选择**新建 > PyODPS 3**。

   如果您需要创建业务流程，请参见创建业务流程。

3. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。

   > **说明** 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。

4. 单击**提交**。
5. 编辑并执行PyODPS 3节点。

   例如，您在使用execute_sql接口时，需要手动设置SQL运行参数。详情请参见执行SQL。

   ```
   hints={'odps.sql.python.version': 'cp37', 'odps.isolation.session.enable': True}
   ```

   当您使用DataFrame自定义函数（df.map、df.map_reduce、df.apply和df.agg）时，请进行如下设置。

```
hints={'odps.isolation.session.enable': True}
```

PyODPS会根据客户端使用的Python版本决定UDF的运行环境，提交SQL查询语句。例如，通过公共Python UDF执行DataFrame，在客户端使用Python 3时，会根据Python 3进行解释。如果相应的UDF使用print语句等Python 2特有的语法或库，执行语句会报ScriptError的错误。如果您需要在PyODPS 2节点中调用第三方包，请参见在PyODPS节点中调用第三方包。

6. 单击节点编辑区域右侧的**调度配置**，配置节点的调度属性，详情请参见配置基础属性。
7. 保存并提交节点。

> **注意** 您需要设置节点的**重跑属性**和**依赖的上游节点**，才可以提交节点。

i. 单击工具栏中的图标，保存节点。

ii. 单击工具栏中的图标。

iii. 在**提交新版本**对话框中，输入**变更描述**。

iv. 单击**确认**。

如果您使用的是标准模式的工作空间，提交成功后，请单击右上方的**发布**。具体操作请参见发布任务。

8. 测试节点，详情请参见查看并管理周期任务。

14.6. 创建ODPS Script节点

ODPS Script节点的SQL开发模式是MaxCompute基于2.0的SQL引擎提供的脚本开发模式。

背景信息

编译脚本时，ODPS Script节点可以将一个多语句的SQL脚本文件作为一个整体进行编译，无需逐条语句进行编译。将其作为一个整体提交运行，可以保证一个执行计划一次排队、一次执行，充分利用MaxCompute的资源。

Script Mode的SQL编译较为简单，您只需要按照业务逻辑，用类似于普通编程语言的方式进行编译，无需考虑如何组织语句。

```
--SET语句
set odps.sql.type.system.odps2=true;
[set odps.stage.reducer.num=***;]
[...]
--DDL语句
create table table1 xxx;
[create table table2 xxx;]
[...]
--DML语句
@var1 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
    FROM table3
    [WHERE where_condition];
@var2 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
    FROM table4
    [WHERE where_condition];
@var3 := SELECT [ALL | DISTINCT] var1.select_expr, var2.select_expr, ...
    FROM @var1 join @var2 on ...;
INSERT OVERWRITE|INTO TABLE [PARTITION (partcol1=val1, partcol2=val2 ...)]
    SELECT [ALL | DISTINCT] select_expr, select_expr, ...
    FROM @var3;
[@var4 := SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr, ... FROM @var1
    UNION ALL | UNION
    SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr, ... FROM @var2;
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
    AS
    SELECT [ALL | DISTINCT] select_expr, select_expr, ...
    FROM var4;]
```

ODPS Script节点的使用限制如下：

- 脚本模式支持SET语句、部分DDL语句（结果是屏显类型的语句除外，例如 `desc` 、 `show` ）和DML语句。
- 一个脚本的完整形式是SET语句>DDL语句>DML语句。每种类型语句都可以有0到多个语句，但不同类型的语句不能混合。
- 多个语句以@开始，表示变量连接。
- 一个脚本，目前最多支持一个屏幕显示结果的语句（例如单独的Select语句），否则会报错。不建议您在脚本中执行屏幕显示的Select语句。
- 一个脚本，目前最多支持一个 `Create table as` 语句，并且必须是最后一句。建议将建表语句和Insert语句分开写。
- 脚本模式下，如果有一个语句失败，整个脚本的语句都不会执行成功。
- 脚本模式下，只有所有输入的数据都准备完成，才会生成一个作业进行数据处理。
- 脚本模式下，如果一个表被写入后又被读取，会报错。

```
insert overwrite table src2 select * from src where key > 0;
@a := select * from src2;
select * from @a;
```

为避免先写后读，可以进行如下修改。

```
@a := select * from src where key > 0;
insert overwrite table src2 select * from @a;
select * from @a;
```

示例如下。

```
create table if not exists dest(key string , value bigint) partitioned by (d string);
create table if not exists dest2(key string,value bigint ) partitioned by (d string);
@a := select * from src where value >0;
@b := select * from src2 where key is not null;
@c := select * from src3 where value is not null;
@d := select a.key,b.value from @a left outer join @b on a.key=b.key and b.value>0;
@e := select a.key,c.value from @a inner join @c on a.key=c.key;
@f := select * from @d union select * from @e union select * from @a;
insert overwrite table dest partition (d='20171111') select * from @f;
@g := select e.key,c.value  from @e join @c on e.key=c.key;
insert overwrite table dest2 partition (d='20171111') SELECT * from @g;
```

脚本模式适用于以下场景：

- 脚本模式更适合用于改写需要层层嵌套子查询的单个语句，或因为脚本复杂性而不得不拆成多个语句的脚本。
- 多个输入的数据源数据准备完成的时间相差很大（例如一个凌晨1点可以准备好，另一个上午7点可以准备好），不适合通过table variable衔接，可以拼接为一个大的脚本模式SQL。

操作步骤

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 鼠标悬停至+新建图标，单击**MaxCompute > ODPS Script**。

   您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > ODPS Script**。
3. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。

   > **说明** 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。
4. 单击**提交**。
5. 在节点的编辑页面编辑脚本，详情请参见开发及提交SQL脚本。
6. 单击节点编辑区域右侧的**调度配置**，配置节点的调度属性，详情请参见配置基础属性。
7. 保存并提交节点。

   > **注意** 您需要设置节点的**重跑属性**和**依赖的上游节点**，才可以提交节点。

   i. 单击工具栏中的图标，保存节点。
   ii. 单击工具栏中的图标。
   iii. 在**提交新版本**对话框中，输入**变更描述**。
   iv. 单击**确认**。

   如果您使用的是标准模式的工作空间，提交成功后，请单击右上方的**发布**。具体操作请参见发布任务。
8. 测试节点，详情请参见查看并管理周期任务。

14.7. 创建ODPS MR节点

MaxCompute提供MapReduce编程接口。您可以通过创建ODPS MR类型节点并提交任务调度，使用MapReduce Java API编写MapReduce程序来处理MaxCompute中的数据。

前提条件

您需要上传并提交、发布使用的资源后，再创建ODPS MR节点。

ODPS MR类型节点的编辑和使用方法，请参见WordCount示例。

操作步骤

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 创建JAR资源。
   i. 鼠标悬停至+新建图标，单击**MaxCompute > 资源 > JAR**。

   您也可以找到相应的业务流程，右键单击**MaxCompute**，选择**新建 > 资源 > JAR**。
   ii. 在**新建资源**对话框中，输入**资源名称**，并选择**目标文件夹**。

   > **说明**
   > - 如果绑定多个实例，则需要选择**MaxCompute引擎实例**。
   > - 如果该JAR包已经在MaxCompute（ODPS）客户端上传过，则需要取消勾选**上传为ODPS资源**，否则上传会报错。
   > - 资源名称无需与上传的文件名保持一致。
   > - 资源名称命名规范：1~128个字符，字母、数字、下划线、小数点，大小写不敏感，JAR资源的后缀为.jar，Python资源的后缀为.py。

   iii. 单击**点击上传**，在本地选择相应文件后，单击**打开**。

   本文以mapreduce_example.jar为例。
   iv. 在**新建资源**对话框中，单击**确定**。
   v. 单击工具栏中的和图标，保存并提交资源至调度开发服务器端。
3. 创建ODPS MR节点。
   i. 鼠标悬停至+新建图标，单击**MaxCompute > ODPS MR**。

   您也可以找到相应的业务流程，右键单击**MaxCompute**，选择**新建 > ODPS MR**。
   ii. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。

   > **说明** 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。

   iii. 单击**提交**。
4. 在节点的编辑页面输入代码，示例如下。

```
--创建输入表。
CREATE TABLE if not exists jingyan_wc_in (key STRING, value STRING);
--创建输出表。
CREATE TABLE if not exists jingyan_wc_out (key STRING, cnt BIGINT);
    ---创建系统dual。
    drop table if exists dual;
    create table dual(id bigint); --如果工作空间不存在该伪表，则需要创建并初始化数据。
    ---向系统伪表初始化数据。
    insert overwrite table dual select count(*)from dual;
    ---向输入表wc_in插入示例数据。
    insert overwrite table jingyan_wc_in select * from (
    select 'project','val_pro' from dual
    union all
    select 'problem','val_pro' from dual
    union all
    select 'package','val_a' from dual
    union all
    select 'pad','val_a' from dual
      ) b;
-- 引用刚刚上传的JAR包资源，可以在资源管理栏中找到该资源，右键引用资源。
--@resource_reference{"mapreduce-examples.jar"}
jar -resources mapreduce-examples.jar -classpath ./mapreduce-examples.jar com.aliyun.odps.mapred.open.example.WordCount jingy
an_wc_in jingyan_wc_out
```

代码说明如下：

- `--@resource_reference`：您可以右键单击资源名称，选择**引用资源**，即可自动产生该条语句。
- `-resources`：引用到的JAR资源文件名。
- `-classpath`：JAR包的路径。由于已经引用了资源，此处路径统一为./下的JAR包。
- `com.aliyun.odps.mapred.open.example.WordCount`：执行过程调用JAR中的主类，需要和JAR中的主类名称保持一致。

- `jingyan_wc_in`：MR的输入表名称，已在上述代码中提前创建。
- `jingyan_wc_out`：MR的输出表名称，已在上述代码中提前创建。
- 一个MR调用多个JAR资源时，classpath写法为`-classpath ./xxxx1.jar,./xxxx2.jar`，即两个路径之间用英文逗号（,）分隔。

5. 单击节点编辑区域右侧的**调度配置**，配置节点的调度属性，详情请参见配置基础属性。
6. 保存并提交节点。

> **注意** 您需要设置节点的**重跑属性**和**依赖的上游节点**，才可以提交节点。

i. 单击工具栏中的图标，保存节点。

ii. 单击工具栏中的图标。

iii. 在**提交新版本**对话框中，输入**变更描述**。

iv. 单击**确认**。

如果您使用的是标准模式的工作空间，提交成功后，请单击右上方的**发布**。具体操作请参见发布任务。

7. 测试节点，详情请参见查看并管理周期任务。

14.8. 创建MaxCompute表

本文为您介绍如何创建MaxCompute表。

前提条件

您在**工作空间配置**页面添加MaxCompute引擎后，当前页面才会显示MaxCompute目录。详情请参见配置工作空间。

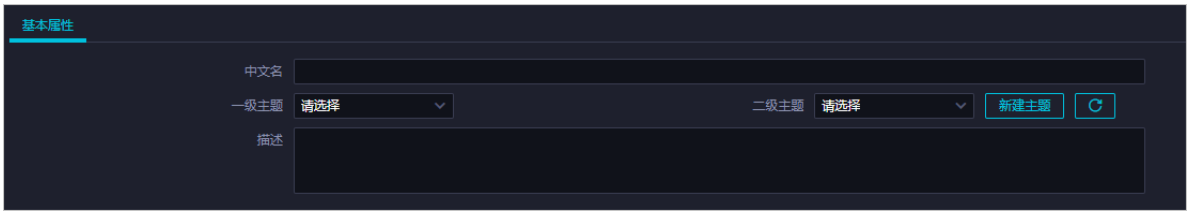
操作步骤

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 鼠标悬停至+新建图标，单击**MaxCompute > 表**。

   您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > 表**。
3. 在**新建表**对话框中，输入**表名**，单击**提交**，进入表的编辑页面。

> **注意**
> - 表名不能超过64个字符，且必须以字母开头，不能包含中文或特殊字符。
> - 如果绑定多个实例，则需要选择**MaxCompute引擎实例**。

4. 在**基本属性**区域，配置各项参数。



名称	描述
中文名	MaxCompute表的中文名称。
一级主题	新建表所处的一级目标文件夹名称。 **说明** 一级、二级主题仅仅是DataWorks上文件夹的摆放形式，目的是为了您能更好地管理您的表。
二级主题	新建表所处的二级目标文件夹名称。
新建主题	单击**新建主题**，进入**主题管理**页面，您可以在该页面创建一级主题、二级主题。 新建主题后，单击图标，即可同步新建的主题。

名称	描述
描述	对新建表进行简单描述。

5. 创建表。

   您可以通过以下两种方式创建表：

   - 使用DDL模式创建表。

     单击工具栏中的**DDL模式**，在对话框中输入建表语句，单击**生成表结构**，即可自动填充**物理模型设计**、**表结构设计**中的相关内容。建表语句的详情请参见标准的建表语句。

   - 使用图形界面创建表。

     如果不适用于DDL模式建表，您也可以使用图形界面直接建表。



分类	参数	描述
物理模型设计	**分区类型**	包括**分区表**和**非分区表**。
	生命周期	MaxCompute的生命周期功能。如果选中生命周期，请在**选择生命周期（日）**中输入一个数字表示天数，该表（或分区）超过设置的天数，会清除未更新的数据。
	层级	通常分为ODS、CDM和ADS三个层级，您可以自定义层级名称。 关于物理层级的相关信息请参见数仓分层。
	物理分类	包括基础业务层、高级业务层和其它，您可以自定义分类名称。 单击**新建层级**，进入**层级管理**页面，即可新增表层级和表物理分类。 **说明** 物理分类仅为方便您的管理，不涉及底层实现。
表结构设计	**添加字段**	单击**添加字段**，配置字段信息后，单击**保存**，即可新增一个字段。
	上移	调整未创建的表的字段顺序。如果为已经创建的表调整字段顺序，会要求删除当前已经创建的表，再新建一张同名表。生产环境中禁止该操作。
	下移	
	字段英文名	字段英文名，由字母、数字和下划线（_）组成。
	字段中文名	字段的中文名称。
	字段类型	MaxCompute数据类型，支持TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、VARCHAR、CHAR、STRING、BINARY、DATETIME、DATE、TIMESTAMP、BOOLEAN、ARRAY、MAP和STRUCT。详情请参见数据类型。
	长度/设置	当选择的字段类型需要设置长度时，请在文本框中进行配置。
	描述	对字段进行描述。
	主键	勾选表示该字段是主键，或者是联合主键的其中一个字段。
	编辑	单击已保存字段后的**编辑**，修改当前字段的配置，并单击**保存**。
	删除	删除已经创建的字段。 **说明** 已经创建的表，删除字段重新提交时，会要求删除当前表，再去建一张同名表，在生产环境中禁止该操作。

分类	参数	描述
分区字段设计 说明 当物理模型设计选择分区表后才显示分区字段设计。	添加分区	如果您在**物理模型设计**区域，设置**分区类型**为**分区表**，则需要配置分区。 您可以为当前表新建一个分区。如果为已经创建的表添加分区，会要求删除当前已经创建的表，再新建一张同名表。生产环境中禁止该操作。
	字段类型	建议统一采用STRING类型。
	日期分区格式	如果该分区字段是日期含义（尽管数据类型可能是STRING），则一个或自填一个日期格式，常用格式为*yyyymmdd*、*yyyy-mm-dd*。
	日期分区粒度	支持的分区粒度包括秒、分、时、日、月、季度和年。创建分区的粒度根据需要填写，如果需要填写多个分区粒度，则默认粒度越大，分区等级越高。例如，同时存在日、时、月三个分区，多级分区关系是一级分区（月），二级分区（日），三级分区（时）。
	删除	可以删除一个分区。如果删除已创建的表的分区，会要求删除当前已经创建的表，再新建一张同名表。生产环境中禁止该操作。

6. 分别单击**提交到开发环境**和**提交到生产环境**。

   如果您使用的是简单模式的工作空间，仅需要单击**提交到生产环境**。

名称	描述
从开发环境加载	如果该表已经提交到开发环境，该按钮会高亮显示。单击后，开发环境已经创建的表信息会覆盖当前的页面信息。 说明 仅MaxCompute表支持该功能。
提交到开发环境	请确认当前编辑页面的必填项是否已经填写完整。如果有遗漏会禁止提交。
从生产环境加载	已经提交到生产环境的表的详细信息会覆盖当前页面。 说明 仅MaxCompute表支持该功能。
提交到生产环境	提交后，会在生产环境的工作空间创建该表。

后续步骤

新建表成功后，您可以进行查询、修改和删除等操作，详情请参见管理表。

14.9. 创建MaxCompute资源

本文为您介绍如何创建JAR和Python类型的资源，以及如何引用和下载资源。

前提条件

您在**工作空间配置**页面添加MaxCompute引擎后，当前页面才会显示MaxCompute目录。详情请参见配置工作空间。

背景信息

如果您的代码或函数中需要使用.jar等资源文件，您可以先上传资源至该工作空间，再进行引用。

如果现有的系统内置函数无法满足您的需求，DataWorks支持创建自定义函数，实现个性化处理逻辑。将实现逻辑的JAR包上传至工作空间下，便可以在创建自定义函数时进行引用。

说明
- 您可以在**函数列表**面板查看系统内置的函数，详情请参见查看函数列表。
- 您可以在**MaxCompute函数**面板查看在DataWorks提交或发布的函数，详情请参见MaxCompute函数。

您可以将文本文件、Python代码以及.zip、.tgz、.tar.gz、.tar、.jar等压缩包作为不同类型的资源上传至MaxCompute，在UDF及MapReduce的运行过程中读取、使用资源。

MaxCompute为您提供读取、使用资源的接口。目前资源包括以下类型：

- **Python**：您编写的Python代码，用于注册Python UDF函数。
- **JAR**：编译好的Java JAR包。
- **Archive**：通过资源名称中的后缀识别压缩类型，支持的压缩文件类型包括.zip、.tgz、.tar.gz、.tar和.jar。
- **File**：仅支持.zip、.so和.jar类型的File资源。

JAR和File类型的资源，区别如下：

- JAR资源是您在线下Java环境编辑Java代码，打包为JAR包上传至DataWorks。
- File类型的小文件资源可以直接在DataWorks上编辑。
- 新建File类型资源时，选中**大文件**，可以上传超过500 KB的本地资源文件。

> **说明** 目前支持最大可以上传50 MB资源。超过50 MB的资源，您可以通过MaxCompute客户端上传，并使用MaxCompute资源提交至DataWorks。详情请参见MaxCompute资源。

创建JAR资源

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 鼠标悬停至+新建图标，单击**MaxCompute > 资源 > JAR**。

   您也可以展开**业务流程**目录下的目标业务流程，右键单击**MaxCompute**，选择**新建 > 资源 > JAR**。

   如果您需要创建业务流程，请参见创建业务流程。
3. 在**新建资源**对话框中，输入**资源名称**，并选择**目标文件夹**。

   > **说明**
   > - 如果该JAR包已经在MaxCompute（ODPS）客户端上传过，则需要取消勾选**上传为ODPS资源**，否则上传会报错。
   > - 资源名称无需与上传的文件名保持一致。
   > - 资源名称命名规范：1~128个字符，字母、数字、下划线、小数点，大小写不敏感，JAR资源的后缀为.jar，Python资源的后缀为.py。
4. 单击**点击上传**，选择相应的文件进行上传。
5. 单击**确定**。
6. 单击工具栏中的图标，提交资源至调度开发服务器端。

   如果您使用的是标准模式的工作空间，提交成功后，请单击右上方的发布。具体操作请参见发布任务

创建Python资源并注册函数

1. 创建Python资源。
   i. 在DataStudio（**数据开发**）页面，鼠标悬停至+新建图标，单击**MaxCompute > 资源 > Python**。

   您也可以展开**业务流程**目录下的目标业务流程，右键单击**MaxCompute**，选择**新建 > 资源 > Python**。
   ii. 在**新建资源**对话框中，输入**资源名称**，并选择**目标文件夹**。

   > **注意** 资源名称只能包含中文、字母、数字、点、下划线（_）、减号（-），且必须加后缀名.py。
   iii. 单击**确定**。
   iv. 在您新建的Python资源内编写Python资源代码，示例如下。

```
from odps.udf import annotate
@annotate("string->bigint")
class ipint(object):
    def evaluate(self, ip):
        try:
            return reduce(lambda x, y: (x << 8) + y, map(int, ip.split('.')))
        except:
            return 0
```

   v. 单击工具栏中的图标，提交节点。

   如果您使用的是标准模式的工作空间，提交成功后，请单击右上方的发布。具体操作请参见发布任务。
2. 注册函数。
   i. 在DataStudio（**数据开发**）页面，鼠标悬停至+新建图标，单击**MaxCompute > 函数**。

   您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > 函数**。
   ii. 在**新建函数**对话框中，输入**函数名称**，并选择**目标文件夹**。
   iii. 单击**提交**。
   iv. 在**注册函数**对话框中，输入函数的类名（示例为 `ipint.ipint` ），在资源列表输入提交的资源名称，单击工具栏中的图标。

v. 验证ipint函数是否生效并满足预期值。您可以在DataWorks上新建一个ODPS SQL类型节点运行SQL语句查询。

您也可以在本地创建ipint.py文件，使用MaxCompute客户端上传资源，详情请参见MaxCompute客户端。

```
odps@ MaxCompute_DOC>add py D:/ipint.py;
OK: Resource 'ipint.py' have been created.
```

```
odps@ MaxCompute_DOC>create function ipint as ipint.ipint using ipint.py;
Success: Function 'ipint' have been created.
```

完成上传后，使用客户端直接注册函数，详情请参见注册函数。完成注册后，即可正常使用该函数。

引用和下载资源

- 在函数中引用资源请参见注册函数。
- 在节点中引用资源请参见创建ODPS MR节点。

如果您需要下载资源，请双击**资源**选择您需要的资源，单击**下载**。通过MaxCompute客户端下载资源的详情请参见资源操作。

其他操作

资源创建成功后，您还可以在对应业务流程下，选择**MaxCompute > 资源**，右键单击对应资源，进行资源的重命名、引用及删除等操作。删除资源的具体操作请参见数据开发与运行。

14.10. 注册MaxCompute函数

DataWorks支持Python和Java两种语言接口，本文为您介绍如何注册函数。

前提条件

您需要先上传资源，才可以注册函数。

操作步骤

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 创建业务流程，详情请参见创建业务流程。
3. 在本地Java环境编辑程序并生成JAR包后，再新建JAR资源，并提交发布。详情请参见创建MaxCompute资源。
4. 新建函数。
   i. 打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > 函数**。
   ii. 在**新建函数**对话框中，输入**函数名称**，并选择**目标文件夹**。
   iii. 单击**提交**。

iv. 在**注册函数**对话框中，配置各项参数。



参数	描述
函数类型	选择函数类型，包括**数学运算函数**、**聚合函数**、**字符串处理函数**、**日期函数**、**窗口函数**和**其他函数**。
MaxCompute引擎实例	默认不可以修改。
函数名	UDF函数名，即SQL中引用该函数所使用的名称。需要全局唯一，且注册函数后不支持修改。
责任人	默认显示。
类名	实现UDF的主类名，必填。 **说明** 当资源类型为Python时，类名格式为Python资源名称.类名（资源名称中的.py无需填写）。
资源列表	支持模糊匹配查找本工作空间中已添加的资源，必填。
描述	针对当前UDF作用的简单描述。
命令格式	该UDF的具体使用方法示例，例如`test`。
参数说明	支持输入的参数类型以及返回参数类型的具体说明。
返回值	返回值，例如1，非必填项。
示例	函数中的示例，非必填项。

5. 单击工具栏中的图标。
6. 提交函数。
   i. 单击工具栏中的图标。
   ii. 在**提交新版本**对话框中，输入**变更描述**。
   iii. 单击**确认**。

15.优化器信息收集

MaxCompute的优化器是基于代价的优化器，需要基于数据的一些特征（即元数据），例如行数、字符串平均长度，准确估算代价。本文为您介绍MaxCompute收集元数据的方法，为优化查询性能提供帮助。

背景信息

如果获取不到准确的元数据，优化器会对代价产生误判，生成不良的执行计划，因此元数据对于优化器至关重要。表的元数据主要是通过对其数据收集统计信息（Column stats）来获取，该元数据是推算其它元数据的基础。

MaxCompute提供了如下两种收集方式：

- 异步收集框架（Analyze）：用户异步通过 `analyze` 命令收集。需要用户主动收集。

  > **说明** MaxCompute客户端版本要求在0.35以上。

- 同步收集框架（Freeride）：在数据生成的同时，自动收集Column stats，更加自动化，但对查询时延有影响。

MaxCompute对不同数据类型的数据收集的Column stats指标如下。

Column stats指标/数据类型	数值类型（TINYINT、SMALLINT、INT、BIGINT、DOUBLE、DECIMAL、NUMERIC）	字符类型（STRING、VARCHAR、CHAR）	二进制类型（BINARY）	布尔类型（BOOLEAN）	日期类型（TIMESTAMP、DATE、INTERVAL）	复杂类型（MAP、STRUCT、ARRAY）
min（最小值）	Y	N	N	N	Y	N
max（最大值）	Y	N	N	N	Y	N
nNulls（空值个数）	Y	Y	Y	Y	Y	Y
avgColLen（平均列长度）	N	Y	Y	N	N	N
maxColLen（最大列长度）	N	Y	Y	N	N	N
ndv（不同值个数）	Y	Y	Y	Y	Y	N
topK（出现频率最高的前K个值）	Y	Y	Y	Y	Y	N

> **说明** Y表示支持，N表示不支持。

使用场景

Column stats指标的使用场景如下：

Column stats指标	功能	场景	说明
min（最小值）或max（最大值）	获取最小值或最大值提升性能优化准确率。	场景1：估算输出记录数。	只提供数据类型时，值域很大。当提供了最大最小值时，优化器可以对过滤条件的选择度有更合理的估计，从而提供更优的执行计划。
		场景2：将过滤条件下推至存储层，减少读取的数据量。	在MaxCompute中，过滤条件 `a < -90` 可以被下推至存储层，以减少数据的读取。而 `a + 100 < 10` 则无法完成下推。如果考虑a的溢出，这两个过滤条件是不等价的。假设a有最大值，转换就变成等价的。因此，如果有最大值最小值，可以对更多的过滤条件进行下推，减少读取的数据量，节省费用。
nNulls（空值个数）	根据是否为空值信息提高判断效率。	场景1：运行作业时减少NULL判断。	在运行作业时，对于任何类型数据都需要判断是否为NULL，如果能准确得到nNulls=0，此判断逻辑可以被忽略以提高计算性能。

Column stats指标	功能	场景	说明
		场景2：基于过滤条件裁剪数据。	如果整列数据值都为NULL，一般的过滤条件可以直接变成 `always false` 条件而被裁剪，提高剪裁效率。
avgColLen（平均列长度）或maxColLen（最大列长度）	获取列长度信息，预估资源消耗，减少Shuffle。	场景1：Hash聚簇表内存估计。	例如，根据avgColLen，可以估计变长字段的内存消耗，得到Record的内存消耗。从而可以选择性进行Auto Mapjoin，即建立Hash聚簇表Broadcast处理机制，减少一次Shuffle操作。对于输入为大表的场景，减少Shuffle的代价非常明显，能有效提升性能。
		场景2：减少Shuffle的数据量。	无。
ndv（不同值个数）	根据基数信息提高执行计划的质量。	场景1：JOIN的输出记录数推算。	• 数据膨胀：当两个表Join key的ndv都比行数小很多时，证明数据大量重复，大概率存在数据膨胀，优化器可以采取相关措施来规避数据膨胀带来的问题。 • 数据过滤：当小表的ndv比大表小很多时，说明执行JOIN操作后，大表的数据会被大量过滤。优化器可以根据这些判断给出相关优化决策。
		场景2：JOIN排序。	基于估算的输出记录数，优化器还可以自动调整JOIN顺序。例如把会有数据过滤的JOIN操作往前调，把有数据膨胀的JOIN操作往后调。
topK（出现频率最高的前K个值）	估算数据分布减少数据倾斜带来的性能影响。	场景1：倾斜数据进行JOIN的优化处理。	当JOIN的输入均较大，且无法通过Mapjoin方式将非大表全装载至内存时，在一路中某些数据存在倾斜状态，而其它路比较有限。MaxCompute可以自动转换成Skew Ddata使用MAP JOIN处理，非倾斜数据使用MERGE JOIN进行处理，最后再合并两部分的计算结果。此功能对于大数据量JOIN，收益非常明显，降低失败后的人工处理成本。
		场景2：估算输出记录数。	利用ndv、min、max进行输出记录数的估算是基于数据“平均”的假设。在用户数据存在明显倾斜时，基于前面假设的推论会存在“失真”。需要对倾斜数据进行特殊处理，而其它数据利用平均假设更合适。

Analyze使用说明

以分区表和非分区表为例介绍Analyze使用方法。

- **非分区表**

  支持对指定的列或全部列收集Column Stats。

  i. 通过MaxCompute客户端创建一张非分区表analyze2_test，命令示例如下：

  ```
  create table if not exists analyze2_test (tinyint1 tinyint, smallint1 smallint, int1 int, bigint1 bigint, double1 double, decimal1 decimal, decimal2 decimal(20,10), string1 string, varchar1 varchar(10), boolean1 boolean, timestamp1 timestamp, datetime1 datetime ) lifecycle 30;
  ```

  ii. 向表中插入数据，命令示例如下：

  ```
  insert overwrite table analyze2_test select * from values (1Y, 20S, 4, 8L, 123452.3, 12.4, 52.5, 'str1', 'str21', false, timestamp '2018-09-17 00:00:00', datetime '2018-09-17 00:59:59') ,(10Y, 2S, 7, 11111118L, 67892.3, 22.4, 42.5, 'str12', 'str200', true, timestamp '2018-09-17 00:00:00', datetime '2018-09-16 00:59:59') ,(20Y, 7S, 4, 2222228L, 12.3, 2.4, 2.57, 'str123', 'str2', false, timestamp '2018-09-18 00:00:00', datetime '2018-09-17 00:59:59') ,(null, null, null, null, null, null, null, null, null, null, null , null) as t(tinyint1, smallint1, int1, bigint1, double1, decimal1, decimal2, string1, varchar1, boolean1, timestamp1, datetime1);
  ```

  iii. 执行 `analyze` 命令收集某一列、多列或全部列的Column Stats，命令示例如下：

```
--收集tinyint1列的Column Stats。
analyze table analyze2_test compute statistics for columns (tinyint1);
--收集smallint1、string1、boolean1和timestamp1列的Column Stats。
analyze table analyze2_test compute statistics for columns (smallint1, string1, boolean1, timestamp1);
--收集全部列的Column Stats。
analyze table analyze2_test compute statistics for columns;
```

iv. 执行 `show statistic` 命令测试Column Stats收集结果，命令示例如下：

```
--测试tinyint1列的Column Stats收集结果。
show statistic analyze2_test columns (tinyint1);
--测试smallint1、string1、boolean1和timestamp1列的Column Stats收集结果。
show statistic analyze2_test columns (smallint1, string1, boolean1, timestamp1);
--测试全部列的Column Stats收集结果。
show statistic analyze2_test columns;
```

返回结果如下：

```
--tinyint1列的Column Stats收集结果。
ID = 20201126085225150gnqo****
tinyint1:MaxValue:      20                  --对应max。
tinyint1:DistinctNum:   4.0                 --对应ndv。
tinyint1:MinValue:      1                   --对应min。
tinyint1:NullNum:       1.0                 --对应nNulls。
tinyint1:TopK:  {1=1.0, 10=1.0, 20=1.0}     --对应topK。10=1.0表示列值10出现的频次为1。topK最多显示前20个最高频次的值。
--smallint1、string1、boolean1和timestamp1列的Column Stats收集结果。
ID = 20201126091636149gxgf****
smallint1:MaxValue:     20
smallint1:DistinctNum:  4.0
smallint1:MinValue:     2
smallint1:NullNum:      1.0
smallint1:TopK:         {2=1.0, 7=1.0, 20=1.0}
string1:MaxLength       6.0                 --对应maxColLen。
string1:AvgLength:      3.0                 --对应avgColLen。
string1:DistinctNum:    4.0
string1:NullNum:        1.0
string1:TopK:   {str1=1.0, str12=1.0, str123=1.0}
boolean1:DistinctNum:   3.0
boolean1:NullNum:       1.0
boolean1:TopK:  {false=2.0, true=1.0}
timestamp1:DistinctNum:         3.0
timestamp1:NullNum:     1.0
timestamp1:TopK:        {2018-09-17 00:00:00.0=2.0, 2018-09-18 00:00:00.0=1.0}
--全部列的Column Stats收集结果。
ID = 20201126092022636gzm1****
tinyint1:MaxValue:      20
tinyint1:DistinctNum:   4.0
tinyint1:MinValue:      1
tinyint1:NullNum:       1.0
tinyint1:TopK:  {1=1.0, 10=1.0, 20=1.0}
smallint1:MaxValue:     20
smallint1:DistinctNum:  4.0
smallint1:MinValue:     2
smallint1:NullNum:      1.0
smallint1:TopK:         {2=1.0, 7=1.0, 20=1.0}
int1:MaxValue:  7
int1:DistinctNum:       3.0
int1:MinValue:  4
int1:NullNum:   1.0
int1:TopK:      {4=2.0, 7=1.0}
bigint1:MaxValue:       11111118
bigint1:DistinctNum:    4.0
bigint1:MinValue:       8
bigint1:NullNum:        1.0
bigint1:TopK:   {8=1.0, 2222228=1.0, 11111118=1.0}
double1:MaxValue:       123452.3
double1:DistinctNum:    4.0
double1:MinValue:       12.3
double1:NullNum:        1.0
double1:TopK:   {12.3=1.0, 67892.3=1.0, 123452.3=1.0}
decimal1:MaxValue:      22.4
decimal1:DistinctNum:   4.0
decimal1:MinValue:      2.4
decimal1:NullNum:       1.0
decimal1:TopK:  {2.4=1.0, 12.4=1.0, 22.4=1.0}
decimal2:MaxValue:      52.5
```

```
decimal2:MaxValue:      52.5
decimal2:DistinctNum:   4.0
decimal2:MinValue:      2.57
decimal2:NullNum:       1.0
decimal2:TopK:  {2.57=1.0, 42.5=1.0, 52.5=1.0}
string1:MaxLength       6.0
string1:AvgLength:      3.0
string1:DistinctNum:    4.0
string1:NullNum:        1.0
string1:TopK:   {str1=1.0, str12=1.0, str123=1.0}
varchar1:MaxLength      6.0
varchar1:AvgLength:     3.0
varchar1:DistinctNum:   4.0
varchar1:NullNum:       1.0
varchar1:TopK:  {str2=1.0, str200=1.0, str21=1.0}
boolean1:DistinctNum:   3.0
boolean1:NullNum:       1.0
boolean1:TopK:  {false=2.0, true=1.0}
timestamp1:DistinctNum:         3.0
timestamp1:NullNum:     1.0
timestamp1:TopK:        {2018-09-17 00:00:00.0=2.0, 2018-09-18 00:00:00.0=1.0}
datetime1:DistinctNum:  3.0
datetime1:NullNum:      1.0
datetime1:TopK:         {1537117199000=2.0, 1537030799000=1.0}
```

- **分区表**

  支持对指定的某个分区收集Column Stats。

  i. 通过MaxCompute客户端创建一张分区表srcpart，命令示例如下：

  ```
  create table if not exists srcpart_test (key string, value string) partitioned by (ds string, hr string) lifecycle 30;
  ```

  ii. 向表中插入数据，命令示例如下：

  ```
  insert into table srcpart_test partition(ds='20201220', hr='11') values ('123', 'val_123'), ('76', 'val_76'), ('447', 'val
  _447'), ('1234', 'val_1234');
  insert into table srcpart_test partition(ds='20201220', hr='12') values ('3', 'val_3'), ('12331', 'val_12331'), ('42', 'va
  l_42'), ('12', 'val_12');
  insert into table srcpart_test partition(ds='20201221', hr='11') values ('543', 'val_543'), ('2', 'val_2'), ('4', 'val_4')
  , ('9', 'val_9');
  insert into table srcpart_test partition(ds='20201221', hr='12') values ('23', 'val_23'), ('56', 'val_56'), ('4111', 'val_
  4111'), ('12333', 'val_12333');
  ```

  iii. 执行 `analyze` 命令收集指定分区的Column Stats，命令示例如下：

  ```
  analyze table srcpart_test partition(ds='20201221') compute statistics for columns (key , value);
  ```

  iv. 执行 `show statistic` 命令测试Column Stats收集结果，命令示例如下：

  ```
  show statistic srcpart_test partition (ds='20201221') columns (key , value);
  ```

  返回结果如下：

  ```
  ID = 20210105121800689g28p****
  (ds=20201221,hr=11) key:MaxLength       3.0
  (ds=20201221,hr=11) key:AvgLength:      1.0
  (ds=20201221,hr=11) key:DistinctNum:    4.0
  (ds=20201221,hr=11) key:NullNum:        0.0
  (ds=20201221,hr=11) key:TopK:   {2=1.0, 4=1.0, 543=1.0, 9=1.0}
  (ds=20201221,hr=11) value:MaxLength     7.0
  (ds=20201221,hr=11) value:AvgLength:    5.0
  (ds=20201221,hr=11) value:DistinctNum:  4.0
  (ds=20201221,hr=11) value:NullNum:      0.0
  (ds=20201221,hr=11) value:TopK:         {val_2=1.0, val_4=1.0, val_543=1.0, val_9=1.0}
  (ds=20201221,hr=12) key:MaxLength       5.0
  (ds=20201221,hr=12) key:AvgLength:      3.0
  (ds=20201221,hr=12) key:DistinctNum:    4.0
  (ds=20201221,hr=12) key:NullNum:        0.0
  (ds=20201221,hr=12) key:TopK:   {12333=1.0, 23=1.0, 4111=1.0, 56=1.0}
  (ds=20201221,hr=12) value:MaxLength     9.0
  (ds=20201221,hr=12) value:AvgLength:    7.0
  (ds=20201221,hr=12) value:DistinctNum:  4.0
  (ds=20201221,hr=12) value:NullNum:      0.0
  (ds=20201221,hr=12) value:TopK:         {val_12333=1.0, val_23=1.0, val_4111=1.0, val_56=1.0}
  ```

Freeride使用说明

您需要在Session级别同时执行如下两个命令设置属性：

- `set odps.optimizer.stat.collect.auto=true;`：启用Freeride功能，自动收集表的Column Stats。
- `set odps.optimizer.stat.collect.plan=xx;`：配置收集计划，收集指定列的指定Column Stats指标。

```
--收集target_table表中列名为key的avgColLen指标。
set odps.optimizer.stat.collect.plan={"target_table":"{\"key\":\"AVG_COL_LEN\"}"}
--收集target_table表中列名为s_binary的min和max，以及列名为s_int的topK和nNulls指标。
set odps.optimizer.stat.collect.plan={"target_table":"{\"s_binary\":\"MIN,MAX\",\"s_int\":\"TOPK,NULLS\"}"};
```

> **说明** 如果出现配置上述属性后，无法收集到信息的问题，可能是Freeride功能未生效。您需要查看Logview的json summary页签中是否可以找到 `odps.optimizer.stat.collect.auto` 属性。如果没有找到该属性，说明当前服务器版本较低，无法使用该功能。MaxCompute会陆续将服务器版本升级至支持Freeride功能的版本。

Column Stats指标在 `set odps.optimizer.stat.collect.plan=xx;` 中的标识对照关系为：

- min：MIN
- max：MAX
- nNulls：NULLS
- avgColLen：AVG_COL_LEN
- maxColLen：MAX_COL_LEN
- ndv：NDV
- topK：TOPK

MaxCompute支持通过 `create table`、`insert into`、`insert overwrite` 三种方式触发Freeride收集Column Stats。

为呈现上述三种方式的实现，假设先创建一个源表src_test并插入数据，命令示例如下：

```
create table if not exists src_test (key string, value string);
insert overwrite table src_test values ('100', 'val_100'), ('100', 'val_50'), ('200', 'val_200'), ('200', 'val_300');
```

- `create table`：在创建目标表target的同时，收集对应的Column Stats。命令示例如下：

```
--创建目标表。
set odps.optimizer.stat.collect.auto=true;
set odps.optimizer.stat.collect.plan={"target_test":"{\"key\":\"AVG_COL_LEN,NULLS\"}"};
create table target_test as select key, value from src_test;
--测试Column Stats收集结果。
show statistic target_test columns;
```

返回结果如下：

```
key:AvgLength:  3.0
key:NullNum:    0.0
```

- `insert into`：在使用 `insert into` 追加数据时，收集对应的Column Stats。命令示例如下：

```
--创建一个目标表。
create table freeride_insert_into_table like src_test;
--追加数据。
set odps.optimizer.stat.collect.auto=true;
set odps.optimizer.stat.collect.plan={"freeride_insert_into_table":"{\"key\":\"AVG_COL_LEN,NULLS\"}"};
insert into table freeride_insert_into_table select key, value from src order by key, value limit 10;
--测试Column Stats收集结果。
show statistic freeride_insert_into_table columns;
```

- `insert overwrite`：在使用 `insert overwrite` 覆盖数据时，收集对应的Column Stats。命令示例如下：

```
--创建一个目标表。
create table freeride_insert_overwrite_table like src_test;
--覆盖数据。
set odps.optimizer.stat.collect.auto=true;
set odps.optimizer.stat.collect.plan={"freeride_insert_overwrite_table":"{\"key\":\"AVG_COL_LEN,NULLS\"}"};
insert overwrite table freeride_insert_overwrite_table select key, value from src_test order by key, value limit 10;
--测试Column Stats收集结果。
show statistic freeride_insert_overwrite_table columns;
```

16.MaxCompute开源支持

MaxCompute是阿里巴巴自研的一站式的快速、完全托管的TB/PB/EB级数据仓库解决方案。本文将为您介绍MaxCompute相关的开源功能。

SDK

MaxCompute提供Java SDK和Python SDK接口实现创建、查看、删除MaxCompute表等操作。通过SDK，您可以通过编辑代码灵活地操作MaxCompute。MaxCompute提供的SDK如下：

- Java SDK

  Java SDK使用说明请参见Java SDK介绍。

  服务支持方式：您可通过访问官方文档和在线提交工单的方式获取支持。

- Python SDK

  PyODPS是MaxCompute的Python版本的SDK，提供对MaxCompute对象的基本操作和DataFrame框架，让您可以轻松地在MaxCompute上进行数据分析。更多详情请参见GitHub项目aliyun-odps-python-sdk和包括所有接口、类的细节等详细内容的PyODPS文档。

  - 欢迎各位开发者参与到PyODPS的生态开发中，在您开始使用PyODPS之前请先安装PyODPS，详细内容请参见PyODPS安装指南。
  - 如何在DataWorks上使用PyODPS，详情请参见PyODPS开发指南。PyODPS提供了DataFrame API，详情请参见PyODPS DataFrame概述。
  - 欢迎您在GitHub aliyun-odps-python-sdk反馈问题和需求，加快PyODPS生态成长。

  服务支持方式：您可通过访问官方文档和在线提交工单的方式获取支持。

MaxCompute RODPS

MaxCompute R语言插件：RODPS。使用说明请参见GitHub ODPS Plugin for R。

服务支持方式：在GitHub ODPS Plugin for R中留言或新建Issue。

ODPS JDBC是MaxCompute官方提供的JDBC驱动，它向Java程序提供了一套执行SQL任务的接口。项目托管在GitHub ODPS JDBC。

服务支持方式：在GitHub ODPS JDBC中留言或新建Issue。

Mars

Mars是一个基于张量的统一分布式计算框架。使用Mars进行科学计算，不仅使大规模科学计算任务的实现从MapReduce上的数千行代码降低到Mars上的数行代码，更在性能上有大幅提升。

Mars已经在Github上开源代码，您可以一起参与共建Mars。详情请参见GitHub开源代码Mars 。

关于Mars的更多说明信息请参见Mars开发指南。

服务支持方式：在Github Mars中留言或新建Issue。

Data collector

MaxCompute数据集成工具插件，包括Flume、OGG、Kettle、Sqoop、Hive 。

Data Collector是MaxCompute主要开源数据采集工具的集合，包括：

- Flume插件
- OGG插件
- Sqoop
- Kettle插件
- Hive Data Transfer UDTF

  Flume和OGG插件是基于DataHub的SDK实现，而Sqoop、Kettle以及Hive Data Transfer UDTF是基于Tunnel的SDK实现。DataHub和Tunnel分别是MaxCompute系统的实时和批量数据通道。Flume插件和OGG插件面向于实时数据传输，而Sqoop、Kettle和Hive Data Transfer UDTF则适合用于离线批量的数据传输。

源代码请参见GitHub Aliyun MaxCompute Data Collectors。插件说明请参见wiki。

服务支持方式：在GitHub Aliyun MaxCompute Data Collectors中留言或新建Issue。