# 阿里云
# MaxCompute

## 用户指南

文档版本：20181008

# 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 |
| --- | --- | --- |
|  | 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。 |  **禁止：**<br>重置操作将丢失用户配置数据。 |
|  | 该类警示信息可能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：**<br>重启操作将导致业务中断，恢复业务所需时间约10分钟。 |
|  | 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：**<br>您也可以通过按**Ctrl** + **A**选中全部文件。 |
| > | 多级菜单递进。 | 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。 | 单击 **确定**。 |
| `courier` 字体 | 命令。 | 执行 `cd /d C:/windows` 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。 | `swich` *`{stand \| slave}`* |

# 目录

# 1 SQL

## 1.1 SQL概述

MaxCompute SQL适用于海量数据（GB、TB、EB级别），离线批量计算的场合。MaxCompute作业提交后会有几十秒到数分钟不等的排队调度，所以适合处理跑批作业，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。

MaxCompute SQL采用的是类似于SQL的语法，可以看作是标准SQL的子集，但不能因此简单地把MaxCompute等价成一个数据库，它在很多方面并不具备数据库的特征，如事务、主键约束、索引等。目前在MaxCompute中允许的最大SQL长度是3MB。

关键字

MaxCompute将SQL语句的关键字作为保留字。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。保留字不区分大小写。下面只给出常用的保留字列表，完整的保留字列表请参见*MaxCompute SQL保留字*。

```
%    &    &&    (    )    *    +
 -    .    /    ;    <    <=    <>
 =    >    >=    ?    ADD    ALL    ALTER
 AND  AS    ASC    BETWEEN    BIGINT    BOOLEAN    BY
 CASE CAST  COLUMN    COMMENT    CREATE    DESC    DISTINCT
 DISTRIBUTE    DOUBLE    DROP    ELSE    FALSE    FROM    FULL
 GROUP    IF    IN    INSERT    INTO    IS    JOIN
 LEFT    LIFECYCLE    LIKE    LIMIT    MAPJOIN    NOT    NULL
 ON    OR    ORDER    OUTER    OVERWRITE    PARTITION    RENAME
 REPLACE    RIGHT    RLIKE    SELECT    SORT    STRING    TABLE
 THEN    TOUCH    TRUE    UNION    VIEW    WHEN    WHERE
```

类型转换说明

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**。更多详情请参见类型转换。

- 显式类型转换：是指用cast将一种数据类型的值转换为另一种类型的值的行为。
- 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。隐式转换作用域包括各种运算符、内建函数等作用域。

分区表

MaxCompute SQL支持分区表。指定分区表会对您带来诸多便利，例如提高SQL运行效率、减少计费等。关于分区的详情请参见基本概念>分区。

**UNION ALL**

参与UNION ALL运算的所有列的数据类型、列个数、列名称必须完全一致，否则会报异常。

## 1.2 运算符

关系操作符

| 操作符 | 说明 |
| --- | --- |
| A=B | 如果A或B为NULL，返回NULL；如果A等于B，返回TRUE，否则返回FALSE。 |
| A<>B | 如果A或B为NULL，返回NULL；如果A不等于B，返回TRUE，否则返回FALSE。 |
| A<B | 如果A或B为NULL，返回NULL；如果A小于B，返回TRUE，否则返回FALSE。 |
| A<=B | 如果A或B为NULL，返回NULL；如果A小于等于B，返回TRUE，否则返回FALSE。 |
| A>B | 如果A或B为NULL，返回NULL；如果A大于B，返回TRUE，否则返回FALSE。 |
| A>=B | 如果A或B为NULL，返回NULL；如果A大于等于B，返回TRUE，否则返FALSE。 |
| A IS NULL | 如果A为NULL，返回TRUE，否则返回FALSE。 |
| A IS NOT NULL | 如果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。（%）匹配任意多个字符，（_）匹配单个字符。要匹配（%）或_’）要用转义符表示‘（%’）‘（_’）。<br>‘aaa’ like‘a__’= TRUE<br>‘aaa’ like‘a%’ = TRUE<br>‘aaa’ like‘aab’= FALSE<br>‘a%b’ like‘a\%b’= TRUE<br>‘axb’ like ‘a\%b’= FALSE |
| A RLIKE B | A是字符串，B是字符串常量正则表达式。如果匹配成功，返回TRUE，否则返回FALSE。 如果B为空串会报错退出。如果A或B为NULL，返回NULL。 |
| A IN B | B是一个集合，如果A为NULL，返回NULL，如A在B中则返回TRUE，否则返回FALSE 若B仅有一个元素NULL，即A IN（NULL），则返回NULL。 若B含 |

| 操作符 | 说明 |
| --- | --- |
| | 有NULL元素，将NULL视为B集合中其他元素的类型。 B必须是常数并且至少有一项，所有类型要一致。 |
| BETWEEN AND | 表达式为A [NOT] BETWEEN B AND C。如果A、B或C为空，则为空。如果A大于或等于B且小于或等于C，则为true，否则为false。 |

常见用法如下所示：

```
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%';
```

由于Double值存在一定的精度差，因此，不建议您直接使用等号对两个Double类型的数据进行比较。您可以使用两个Double类型相减，然后取绝对值的方式进行判断。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，意为取绝对值，详情请参见*ABS*。
- 通常情况下，MaxCompute的Double类型能够保障14位有效数字。

算术操作符

| 操作符 | 说明 |
| --- | --- |
| A + B | 如果A或B为NULL，返回NULL；否则返回A + B的结果。 |
| A – B | 如果A或B为NULL，返回NULL；否则返回A – B的结果。 |
| A * B | 如果A或B为NULL，返回NULL；否则返回A * B的结果。 |
| A / B | 如果A或B为NULL，返回NULL；否则返回A / B的结果。注：如果A和B为Bigint类型，返回结果为Double类型。 |
| A % B | 如果A或B为NULL，返回NULL；否则返回A模B的结果。 |
| +A | 仍然返回A。 |

| 操作符 | 说明 |
| --- | --- |
| -A | 如果A为NULL，返回NULL，否则返回-A。 |

常见用法如下：

```
select age+10, age-10, age%10, -age, age*age, age/10 from user;
```

说明：

- 只有参数是String、Bigint或Double类型才能参与算术运算，日期型和布尔型不允许参与运算。
- String类型在参与运算前会进行隐式类型转换，转换为Double类型。
- Bigint和Double类型共同参与计算时，会将Bigint隐式转换为Double再进行计算，返回结果为Double类型。
- A和B都是Bigint类型，进行A/B运算，返回结果为Double类型。进行上述其他运算，仍然返回Bigint类型。

位操作符

| 操作符 | 说明 |
| --- | --- |
| A & B | 返回A与B进行按位与的结果。例如1&2返回0，1&3返回1，NULL与任何值按位与都为NULL。 A和B必须为Bigint类型。 |
| A \| B | 返回A与B进行按位或的结果。例如1 \|2返回3，1 \|3返回3，NULL与任何值按位或都为NULL。 A和B 必须为Bigint类型。 |

说明：

位运算符不支持隐式转换，只允许Bigint类型。

逻辑操作符

```
操作符              说明
A and B          TRUE and TRUE=TRUE
                 TRUE and FALSE=FALSE
                 FALSE and TRUE=FALSE
                 FALSE and NULL=FALSE
                 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 NULL=NULL
                 NULL or FALSE=NULL
```

```
                    TRUE or NULL=TRUE
                    NULL or TRUE=TRUE
                    NULL or NULL=NULL
    NOT A           如果A是NULL，返回NULL
                    如果A是TRUE，返回FALSE
                    如果A是FALSE，返回TRUE
```

说明：

逻辑操作符只允许Boolean类型参与运算，不支持隐式类型转换。

## 1.3 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

显式类型转换是用CAST将一种数据类型的值转换为另一种类型的值的行为，在MaxCompute SQL中支持的显式类型转换，如下表所示：

| From/To | Bigint | Double | String | Datetime | Boolean | Decimal |
|---|---|---|---|---|---|---|
| Bigint | – | Y | Y | N | N | Y |
| Double | Y | – | Y | N | N | Y |
| String | Y | Y | – | Y | N | Y |
| Datetime | N | N | Y | – | N | N |
| Boolean | N | N | N | N | – | N |
| Decimal | Y | Y | Y | N | N | - |

其中，**Y** 表示可以转换，**N** 表示不可以转换，– 表示不需要转换。

示例如下：

```
select cast(user_id as double) as new_id from user;
select cast('2015-10-01 00:00:00' as datetime) as new_date from user;
```

说明：

- 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`。
- 不支持的显式类型转换会导致异常。
- 如果在执行时转换失败，报错退出。
- 日期类型转换时采用默认格式yyyy-mm-dd hh:mi:ss，详情请参见*String类型与Datetime类型之间的转换*。
- 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*。
- 关于cast的介绍请参见*CAST*。
- 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。

### 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：

| | boolea | tinyint | smallin | int | bigint | float | string | varcha | timesta | binary |
|---|---|---|---|---|---|---|---|---|---|---|
| boolean to | Y | N | N | N | N | N | N | N | N | N |
| tinyint to | N | Y | Y | Y | Y | Y | Y | Y | N | N |
| smallint to | N | N | Y | Y | Y | Y | Y | Y | N | N |
| int to | N | N | Y | Y | Y | Y | Y | Y | N | – |
| bigint to | N | N | N | N | Y | Y | Y | Y | N | N |
| float to | N | N | N | N | Y | Y | Y | Y | N | – |
| double to | N | N | N | N | N | N | Y | Y | N | N |
| decimal to | N | N | N | N | N | N | Y | Y | N | N |
| string tof | N | N | N | N | N | N | Y | Y | N | N |
| varchar to | N | N | N | N | Y | Y | N | N | – | – |
| timestamp to | N | N | N | N | N | N | Y | Y | Y | N |
| binary to | N | N | N | N | N | N | N | N | N | Y |

其中，**T**表示可以转换，**F**表示不可以转换。

**说明：**

- MaxCompute2.0新增了DECIMAL类型与Datetime的常量定义方式，100BD就是数值为100的DECIMAL，Datetime2017-11-11 00:00:00就是Datetime类型的常量。常量定义的方便之处在于可以直接用到values子句和values表中。
- 旧版MaxCompute中，因为历史原因，Double可以隐式的转换为Bigint，这个转换潜在可能有数据丢失，一般数据库系统都不允许。

常见用法如下所示：

```
select user_id+age+'12345',
          concat(user_name,user_id,age)
  from user;
```

**说明：**

- 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。
- 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，显式的用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。详情请参见隐式类型转换的作用域。

- **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和IN。由于LIKE、RLIKE和IN的隐式类型转换规则不同于其他关系运算符，将单独拿出章节对这三种关系运算符做出说明。本小节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 | - | Double | Double | N | N | Decimal |
| Double | Double | - | Double | N | N | Decimal |
| String | Double | Double | - | Datetime | N | Decimal |
| Datetime | N | N | Datetime | - | N | N |
| Boolean | N | N | N | N | - | N |

| From/To | Bigint | Double | String | Datetime | Boolean | Decimal |
|---|---|---|---|---|---|---|
| Decimal | Decimal | Decimal | Decimal | N | N | - |

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，请参见关系操作符。

- 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和IN。

  - LIKE和RLIKE的使用方式，如下所示：

    ```
    source like pattern;
    source rlike pattern;
    ```

    两者在隐式类型转换中的注意事项，如下所示：

    - 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，如下所示：

    ```
    key in (value1, value2, ···)
    ```

    In的隐式转换规则：

    - 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，若Bigint、Double、String之间比较，统一转为Double，若Datetime和String之间比较，统一转为Datetime。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，如下所示：

  - 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和not，其隐式转换规则，如下所示：

- 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。

**内建函数涉及到隐式转换**

MaxCompute SQL提供了大量的系统函数，以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，如下所示：

- 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。

**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，如下所示：

- 如果返回类型只有Bigint、Double，统一转为Double。
- 如果返回类型中有String类型，统一转为String，如果不能转则报错（如Boolean类型）。
- 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就是非法的Datetime格式，无法从这个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的格式。

类似于下面的转换尝试，将会失败导致异常，如下所示：

```
cast("2013/12/31 02/34/34" as datetime)
cast("20131231023434" as datetime)
cast("2013-12-31 2:34:34" as datetime)
```

dd部分的阈值上限取决于月份实际拥有的天数，如果超出对应月份实际拥有的天数，将会导致异常退出，如下所示：

```
cast("2013-02-29 12:12:12" as datetime)      -- 异常返回，2013年2月没有29日
cast("2013-11-31 12:12:12" as datetime)      -- 异常返回，2013年11月没有31日
```

MaxCompute提供了TO_DATE函数，用以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*。

# 1.4 DDL语句

## 1.4.1 表操作

**创建表**

**创建表的语法格式，如下所示：**

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[STORED BY StorageHandler] -- 仅限外部表
[WITH SERDEPROPERTIES (Options)] -- 仅限外部表
[LOCATION OSSLocation];-- 仅限外部表
[LIFECYCLE days]
[AS select_statement]
 CREATE TABLE [IF NOT EXISTS] table_name
```

```
LIKE existing_table_name
```

- 创建表时，如果不指定if not exists选项而存在同名表，则返回出错。若指定此选项，则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信息不会被改动。
- 表名与列名均对大小写不敏感，不能有特殊字符，只能用英文的a-z，A-Z及数字和下划线_，且以字母开头，名称的长度不超过128字节。
- 单表的列定义个数最多1200个。
- 数据类型：Bigint、Double、Boolean、Datetime、Decimal和String等，MaxCompute2.0版本扩展了很多数据类型。

  说明：

  目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

  - session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
  - project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

    ```
    - setproject odps.sql.type.system.odps2=true;
    ```

    对setproject的详细说明请参见：其他操作。
- Partitioned by指定表的分区字段，目前支持Tinyint、Smallint、 Int、 Bigint、Varchar和String类型。

  分区值不允许有双字节字符（如中文），必须是以英文字母a-z，A-Z开始后可跟字母数字，名称的长度不超过128字节。允许的字符包括：空格、冒号（:）、下划线（_）、美元符（$）、井号（#）、点（.），感叹号（!）和（@），出现其他字符行为未定义，例如（\t）、（\n）、（/）等。当利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，可以提高处理效率。
- 一张表最多允许60000个分区，单表的分区层次不能超过6级。
- 注释内容是长度不超过1024字节的有效字符串。
- lifecycle表的生命周期，单位：天。create table like语句不会复制源表的生命周期属性。
- 有关外部表的更多详情请参见访问OSS非结构化数据。

示例如下：

假设创建表sale_detail来保存销售记录，该表使用销售时间（sale_date）和销售区域（region）作为分区列，建表语句如下所示：

```
create table if not exists sale_detail
(
shop_name     string,
customer_id   string,
total_price   double
)
partitioned by (sale_date string,region string);
-- 创建一张分区表sale_detail
```

通过create table…as select…语句创建表，并在建表的同时将数据复制到新表中，如下所示：

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

此时，如果sale_detail中存在数据，上面的示例会将sale_detail的数据全部复制到sale_detail_ctas1表中。

 说明：

此处sale_detail是一张分区表，而通过create table…as select…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，即sale_detail_ctas1是一个含有5列的非分区表。

在create table…as select…语句中，如果在select子句中使用常量作为列的值，建议指定列的名字，如下所示：

```
create table sale_detail_ctas2 as
        select shop_name,
            customer_id,
            total_price,
            '2013' as sale_date,
            'China' as region
        from sale_detail;
```

如果不加列的别名，如下所示：

```
create table sale_detail_ctas3 as
        select shop_name,
            customer_id,
            total_price,
            '2013',
            'China'
```

```
        from sale_detail;
```

则创建的表sale_detail_ctas3的第四、五列类似于_c5、_c6。

如果希望源表和目标表具有相同的表结构，可以尝试使用create table...like操作，如下所示：

```
create table sale_detail_like like sale_detail;
```

此时，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。

### 查看表信息

查看表信息的语法格式，如下所示：

```
desc <table_name>;
desc extended <table_name>;--查看外部表信息
```

示例如下：

- 假设查看上述示例中表sale_detail的信息，可输入如下命令：

  ```
  desc sale_detail;
  ```

  结果如下所示：

  ```
  odps@ $odps_project>desc sale_detail;
  +------------------------------------------------------------------------
  +
  | Owner: ALIYUN$maojing.mj@alibaba-inc.com | Project: $odps_project
                    |
  | 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     |
   |
+------------------------------------------------------------------------
+
OK
```

- 假设查看上述示例表sale_detail_like中的信息，可输入如下命令：

```
desc sale_detail_like
```

结果如下所示：

```
odps@ $odps_project>desc sale_detail_like;
+------------------------------------------------------------------------
+
| Owner: ALIYUN$maojing.mj@alibaba-inc.com | Project: $odps_project
                  |
| 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     |       |
   |
+------------------------------------------------------------------------
+
| Partition Columns:
   |
+------------------------------------------------------------------------
+
| sale_date       | string     |
   |
| region          | string     |
   |
+------------------------------------------------------------------------
+
OK
```

由上可见，除生命周期属性外，sale_detail_like的其他属性（字段类型，分区类型等）均与sale_detail完全一致。查看表信息的更多详情请参见*Describe Table*。

如果您查看表sale_detail_ctas1的信息，会发现sale_date、region两个字段仅会作为普通列存在，而不是表的分区。

### 删除表

删除表的语法格式，如下所示：

```
DROP TABLE [IF EXISTS] table_name;
```

说明：

- 如果不指定if exists选项而表不存在，则返回异常。若指定此选项，无论表是否存在，皆返回成功。
- 删除外部表时，OSS上的数据不会被删除。

示例如下：

```
create table sale_detail_drop like sale_detail;
    drop table sale_detail_drop;
    --若表存在，成功返回；若不存在，异常返回；
    drop table if exists sale_detail_drop2;
```

```
    --无论是否存在 sale_detail_drop2 表，均成功返回。
```

### 重命名表

重命名表的语法格式，如下所示：

```
ALTER TABLE table_name RENAME TO new_table_name;
```

说明：

- rename操作仅修改表的名字，不改动表中的数据。
- 如果已存在与new_table_name同名表，则报错。
- 如果table_name不存在，则报错。

示例如下：

```
create table sale_detail_rename1 like sale_detail;
alter table sale_detail_rename1 rename to sale_detail_rename2;
```

### 修改表的注释

修改表的注释的语法格式，如下所示：

```
ALTER TABLE table_name SET COMMENT 'tbl comment';
```

说明：

- table_name必须是已存在的表。
- comment最长1024字节。

示例如下：

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

通过MaxCompute的**desc**命令可以查看表中comment的修改，详情请参见常用命令>表操作中的Describe Table。

### 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。

修改表的修改时间的语法格式，如下所示：

```
ALTER TABLE table_name TOUCH;
```

 说明：

- table_name不存在，则报错返回。
- 此操作会改变表的LastDataModifiedTime的值，此时，MaxCompute会认为表的数据有变动，生命周期的计算会重新开始。

清空非分区表里的数据

将指定的非分区表中的数据清空，该命令不支持分区表。对于分区表，可以用`ALTER TABLE table_name DROP PARTITION`的方式将分区里的数据清除。

清空非分区表里的数据的语法格式，如下所示：

```
TRUNCATE TABLE table_name;
```

## 1.4.2 生命周期操作

修改表的生命周期

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

修改表的生命周期属性的语法格式，如下所示：

```
ALTER TABLE table_name SET lifecycle days;
```

 说明：

- days参数为生命周期时间，只接受正整数，单位为天。
- 如果表table_name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，将会被MaxCompute自动回收（类似drop table操作）。
- 在MaxCompute中，每当表的数据被修改后，表的LastDataModifiedTime将会被更新，因此，MaxCompute会根据每张表的LastDataModifiedTime以及lifecycle的设置来判断是否要回收此表。
- 如果table_name是分区表，则根据各分区的LastDataModifiedTime判断该分区是否该被回收。
- 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，不能再分区级设置生命周期。

- 创建表时即可指定生命周期。

示例如下：

```
create table test_lifecycle(key string) lifecycle 100;
 -- 新建test_lifecycle表，生命周期为100天。
 alter table test_lifecycle set lifecycle 50;
 -- 修改test_lifecycle表，将生命周期设为50天。
```

禁止生命周期

某些情况下，部分特定的分区不希望被生命周期功能自动回收掉，比如一个月的月初或双十一期间的数据，此时您可以禁止该分区被生命周期功能回收。

禁止生命周期的语法格式，如下所示：

```
ALTER TABLE table_name [partition_spec] ENABLE|DISABLE LIFECYCLE;
```

示例如下：

```
ALTER TABLE trans PARTITION(dt='20141111') DISABLE LIFECYCLE;
```

## 1.4.3 视图操作

创建视图

创建视图的语法格式，如下所示：

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS] view_name
    [(col_name [COMMENT col_comment], ...)]
    [COMMENT view_comment]
    [AS select_statement]
```

 说明：

- 创建视图时，必须有对视图所引用表的读权限。
- 视图只能包含一个有效的select语句。
- 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。
- 当建好视图后，如果视图的引用表发生了变更，有可能导致视图无法访问，例如删除被引用表。您需要自己维护引用表及视图之间的对应关系。
- 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来重建视图，重建后视图本身的权限保持不变。

示例如下：

```
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;
```

删除视图

删除视图的语法格式，如下所示：

```
DROP VIEW [IF EXISTS] view_name;
```

说明：

如果视图不存在且没有指定if exists，则报错。

示例如下：

```
DROP VIEW IF EXISTS sale_detail_view;
```

重命名视图

重命名视图的语法格式，如下所示：

```
ALTER VIEW view_name RENAME TO new_view_name;
```

说明：

如果已存在同名视图，则报错。

示例如下：

```
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;
    alter view sale_detail_view rename to market;
```

## 1.4.4 分区/列操作

添加分区

添加分区的语法格式，如下所示：

```
ALTER TABLE TABLE_NAME ADD [IF NOT EXISTS] PARTITION partition_spec
```

```
partition_spec:(partition_col1 = partition_col_value1, partition_col2
 = partiton_col_value2, ...)
```

说明：

- 仅支持新增分区，不支持新增分区字段。
- 如果未指定if not exists而同名的分区已存在，则出错返回。
- 目前MaxCompute单表支持的分区数量上限为6万。
- 对于多级分区的表，如果想添加新的分区，必须指明全部的分区值。

示例如下：

假设为表sale_detail添加一个分区，如下所示：

```
alter table sale_detail add if not exists partition (sale_date='201312
', region='hangzhou');
-- 成功添加分区，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='shanghai');
-- 成功添加分区，用来存储2013年12月上海地区的销售记录。
alter table sale_detail add if not exists partition(sale_date='
20111011');
-- 仅指定一个分区sale_date，出错返回
alter table sale_detail add if not exists partition(region='shanghai
');
-- 仅指定一个分区region，出错返回
```

删除分区

删除分区的语法格式，如下所示：

```
ALTER TABLE TABLE_NAME DROP [IF EXISTS] PARTITION partition_spec;
partition_spec:(partition_col1 = partition_col_value1, partition_col2
 = partiton_col_value2, ...)
```

说明：

如果分区不存在且未指定if exists，则报错返回。

示例如下：

假设从表sale_detail中删除一个分区，如下所示：

```
alter table sale_detail drop if exists partition(sale_date='201312',
region='hangzhou');
```

```
-- 成功删除2013年12月杭州分区的销售。
```

### 添加列

添加列的语法格式，如下所示：

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1, col_name2 type2
...)
```

说明：

添加的新列不支持指定顺序，默认在最后一列。

### 修改列名

修改列名的语法格式，如下所示：

```
ALTER TABLE table_name CHANGE COLUMN old_col_name RENAME TO new_col_na
me;
```

说明：

- old_col_name必须是已存在的列。
- 表中不能有名为new_col_name的列。

### 修改列、分区注释

修改列、分区注释的语法格式，如下所示：

```
ALTER TABLE table_name CHANGE COLUMN col_name COMMENT comment_string;
```

说明：

COMMENT内容最长为1024字节。

### 同时修改列名及列注释

同时修改列名及列注释的语法格式，如下所示：

```
ALTER TABLE table_name CHANGE COLUMN old_col_name new_col_name
column_type COMMENT column_comment;
```

说明：

- old_col_name必须是已存在的列。
- 表中不能有名为new_col_name的列。

- COMMENT内容最长为1024字节。

修改表、分区的修改时间

MaxCompute SQL提供touch操作用来修改分区的LastDataModifiedTime。效果会将分区的LastDataModifiedTime修改为当前时间。

修改表、分区的修改时间的语法格式，如下所示：

```
ALTER TABLE table_name TOUCH PARTITION(partition_col='partition_
col_value', ...);
```

说明：

- table_name或partition_col不存在，则报错返回。
- 指定的partition_col_value不存在，则报错返回。
- 此操作会改变表的LastDataModifiedTime的值，此时，MaxCompute会认为表或分区的数据有变动，生命周期的计算会重新开始。

修改分区值

MaxCompute SQL支持通过rename操作更改对应表的分区值。

修改分区值的语法格式，如下所示：

```
ALTER TABLE table_name PARTITION (partition_col1 = partition_
col_value1, partition_col2 = partiton_col_value2, ...)
RENAME TO PARTITION (partition_col1 = partition_col_newvalue1,
partition_col2 = partiton_col_newvalue2, ...);
```

说明：

- 不支持修改分区列列名，只能修改分区列对应的值。
- 修改多级分区的一个或者多个分区值，多级分区的每一级的分区值都必须写上。

# 1.5 INSERT操作

## 1.5.1 更新表中的数据（INSERT OVERWRITE/INTO）

命令格式如下：

```
INSERT OVERWRITE|INTO TABLE tablename [PARTITION (partcol1=val1,
partcol2=val2 ...)] [(col1,col2 ...)]
select_statement
```

```
FROM from_statement;
```

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|into**后需要加入table关键字，不是直接使用tablename。
- 当Insert的目标表是分区表时，指定分区值[PARTITION (partcol1=val1, partcol2=val2 …)]语法中不允许使用函数等表达式。
- 目前INSERT OVERWRITE还不支持指定插入列的功能，暂时只能用INSERT INTO。

在MaxCompute SQL处理数据的过程中，Insert overwrite/into用于将计算的结果保存目标表中。

**Insert into**与**Insert overwrite**的区别是：**Insert into**会向表或表的分区中追加数据，而**Insert overwrite**会在向表或分区中插入数据前清空表中的原有数据。

在使用MaxCompute处理数据的过程中，**Insert overwrite/into**是最常用到的语句，它们会将计算的结果保存到一个表中，以供下一步计算使用。比如计算sale_detail表中不同地区的销售额，操作如下：

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

说明：

在进行Insert更新数据操作时，源表与目标表的对应关系依赖于在select子句中列的顺序，而不是表与表之间列名的对应关系，下面的SQL语句仍然是合法的：

```
insert overwrite table sale_detail_insert partition (sale_date='2013
', region='china')
select customer_id, shop_name, total_price from sale_detail;
    -- 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插入数据是，sale_detail的插入顺
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
```

向某个分区插入数据时，分区列不允许出现在select列表中。

```
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;
    -- 报错返回，sale_date，region 为分区列，不允许出现在静态分区的 insert 语
句中。
```

同时，partition的值只能是常量，不可以出现表达式。以下用法是非法的：

```
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;
```

## 1.5.2 多路输出（MULTI INSERT）

MaxCompute SQL支持在一个语句中插入不同的结果表或者分区。

命令格式如下：

```
FROM from_statement
     INSERT OVERWRITE | INTO TABLE tablename1 [PARTITION (partcol1=
val1, partcol2=val2 ...)]
     select_statement1 [FROM from_statement]
      [INSERT OVERWRITE | INTO TABLE tablename2 [PARTITION (partcol1=
val3, partcol2=val4 ...)]
        select_statement2 [FROM from_statement]]
```

 说明：

- 一般情况下，单个SQL中最多可以写256路输出，超过256路，则报语法错误。
- 在一个multi insert中：
  - 对于分区表，同一个目标分区不允许出现多次。
  - 对于未分区表，该表不能出现多次。
- 对于同一张分区表的不同分区，不能同时有Insert overwrite和Insert into操作，否则报错返回。

示例如下：

```
create table sale_detail_multi like sale_detail;
    from sale_detail
        insert overwrite table sale_detail_multi partition (sale_date
='2010', region='china' )
            select shop_name, customer_id, total_price where .....
        insert overwrite table sale_detail_multi partition (sale_date
='2011', region='china' )
            select shop_name, customer_id, total_price where .....;
```

```
    -- 成功返回，将 sale_detail 的数据插入到 sales 里的 2010 年及 2011 年中
国大区的销售记录中。
    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;
    -- 出错返回，同一分区出现多次。
    from sale_detail
        insert overwrite table sale_detail_multi partition (sale_date
='2010', region='china' )
            select shop_name, customer_id, total_price
        insert into table sale_detail_multi partition (sale_date='
2011', region='china' )
            select shop_name, customer_id, total_price;
    -- 出错返回，同一张表的不同分区，不能同时有 insert overwrite 和 insert
into 操作。
```

## 1.5.3 输出到动态分区（DYNAMIC PARTITION）

在Insert overwrite到一张分区表时，可以在语句中指定分区的值。也可以用另外一种更加灵活的方式，在分区中指定一个分区列名，但不给出值。相应地，在select子句中的对应列来提供分区的值。

命令格式如下：

```
insert overwrite table tablename partition (partcol1, partcol2 ...)
select_statement from from_statement;
```

 说明：

- select_statement字段中，后面的字段将提供目标表动态分区值。如目标表就一级动态分区，则select_statement最后一个字段值即为目标表的动态分区值。
- 目前，在使用动态分区功能的SQL中，在分布式环境下，单个进程最多只能输出512个动态分区，否则引发运行时异常。
- 现阶段，任意动态分区SQL不允许生成超过2000个动态分区，否则引发运行时异常。
- 动态生成的分区值不允许为NULL，也不支持含特殊字符和中文，否则会引发异常。

```
FAILED: ODPS-0123031:Partition exception - invalid dynamic
partition value:
province=xxx
```

- 如果目标表有多级分区，在运行Insert语句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。

动态分区的示例如下：

```
create table total_revenues (revenue bigint) partitioned by (region
string);
insert overwrite table total_revenues partition(region)
select total_price as revenue, region
from sale_detail;
```

按照上述写法，在SQL运行之前，是不知道会产生哪些分区的，只有在select运行结束后，才能由region字段产生的值确定会产生哪些分区，这也是叫做动态分区的原因。

其他示例如下：

```
create table sale_detail_dypart like sale_detail;--创建示例目标表
```

示例一：

```
insert overwrite table sale_detail_dypart partition (sale_date, region
)
select shop_name,customer_id,total_price,sale_date,region from
sale_detail;
-- 成功返回;
```

- 此时sale_detail表中，sale_date的值决定目标表的sale_date分区值，region的值决定目标表的region分区值。
- 动态分区中，**select_statement字段和目标表动态分区的对应是按字段顺序决定的**。如该示例中，select语句若写成`select shop_name,customer_id,total_price,region ,sale_date from sale_detail;`，则sale_detail表中，region值决定决定目标表的sale_date分区值，sale_date的值决定目标表的region分区值。

示例二：

```
insert overwrite table sale_detail_dypart partition (sale_date='2013
', region)
select shop_name,customer_id,total_price,region from sale_detail;
-- 成功返回，多级分区，指定一级分区
```

示例三：

```
insert overwrite table sale_detail_dypart partition (sale_date='2013
', region)
select shop_name,customer_id,total_price from sale_detail;
-- 失败返回，动态分区插入时，动态分区列必须在select列表中
```

示例四：

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

```
-- 失败返回，不能仅指定低级子分区，而动态插入高级分区
```

另外，旧版MaxCompute在进行动态分区时，如果分区列的类型与对应select列表中列的类型不严格一致，会报错。MaxCompute2.0则支持隐式类型转换，示例如下：

```
create table parttable(a int, b double) partitioned by (p string);
insert into parttable partition(p) select key, value, current_ti
mestmap() from src;
select * from parttable;
```

执行上述语句后返回结果如下：

| a | b | c |
|---|---|---|
| 0 | NULL | 2017-01-23 22:30:47.130406621 |
| 0 | NULL | 2017-01-23 22:30:47.130406621 |

## 1.5.4 VALUES

通常在业务测试阶段，需要给一个小数据表准备些基本数据，您可以通过**INSERT … VALUES**的方法快速对测试表写入一些测试数据。

 说明：

目前INSERT OVERWRITE还不支持这种指定插入列的功能，暂时只能用INSERT INTO。

命令格式如下：

```
INSERT  INTO  TABLE  tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)][co1name1,colname2...]
[VALUES (col1_value,col2_value,...),(col1_value,col2_value,...),...]
```

示例一：

```
drop table if exists srcp;
create table if not exists srcp (key string ,value bigint) partitioned
 by (p string);
insert into table srcp partition (p='abc') values ('a',1),('b',2),('c
',3);
```

Insert…values语句执行成功后，查询表srcp分区p=abc，结果如下：

```
+-----+------------+---+
| key | value      | p |
+-----+------------+---+
| a   | 1          | abc |
| b   | 2          | abc |
| c   | 3          | abc |
```

```
+-----+------------+---+
```

当表有很多列，而准备数据的时候希望只插入部分列的数据，此时可以使用插入列表功能。

示例二：

```
drop table if exists srcp;
create table if not exists srcp (key string ,value bigint) partitioned
 by (p string);
insert into table srcp partition (p)(key,p) values ('d','20170101'),('
e','20170101'),('f','20170101');
```

Insert…values语句执行成功后，查询表srcp分区p=20170101，结果如下：

```
+-----+------------+---+
| key | value      | p |
+-----+------------+---+
| d   | NULL       | 20170101 |
| e   | NULL       | 20170101 |
| f   | NULL       | 20170101 |
+-----+------------+---+
```

对于在values中没有制定的列，可以看到取缺省值为NULL。插入列表功能不一定和values一起用，对于Insert into…select…，同样可以使用。

Insert…values有一个限制：values必须是常量，但是有时候希望在插入的数据中进行一些简单的运算，此时可以使用MaxCompute的values table功能，详情见示例三。

示例三：

```
drop table if exists srcp;
create table if not exists srcp (key string ,value bigint) partitioned
 by (p string);
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);
```

其中的values (…), (…) t (a, b)相当于定义了一个名为t，列为a，b的表，类型为（a string，b bigint），其中的类型从values列表中推导。这样在不准备任何物理表的时候，可以模拟一个有任意数据的，多行的表，并进行任意运算。

Insert…values语句执行成功后，查询表srcp分区p=20170102，结果如下：

```
+-----+------------+---+
| key | value      | p |
+-----+------------+---+
| d4  | 2          | 20170102 |
| e5  | 2          | 20170102 |
| f6  | 2          | 20170102 |
```

```
+-----+------------+---+
```

**说明：**

- values只支持常量不支持函数，类似ARRAY复杂类型目前无法构造对应的常量，可以写为如下语句：

```
insert into table srcp (p ='abc') select 'a',array('1', '2',
       '3');
```

  可达到一样的效果。

- 通过values写入DATETIME、TIMESTAMP类型，需要在values中指定类型名称，如下所示：

```
insert into table srcp (p ='abc') values (datetime'2017-11-11
           00:00:00',timestamp'2017-11-11 00:00:00.123456789');
```

实际上，values表并不限于在Insert语句中使用，任何DML语句都可以使用。

还有一种values表的特殊形式，如下所示：

```
select abs(-1), length('abc'), getdate();
```

如上述语句所示，可以不写from语句，直接执行select，只要select的表达式列表不用任何上游表数据就可以。其底层实现为从一个1行，0列的匿名values表选取。这样，在希望测试一些函数，比如自己的UDF等时，便不用再手工创建DUAL表。

# 1.6 SELECT操作

## 1.6.1 Select语法介绍

**Select语法介绍**

命令格式如下：

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[ORDER BY order_condition]
[DISTRIBUTE BY distribute_condition [SORT BY sort_condition] ]
[LIMIT number]
```

在使用**Select**语句时，请注意以下几点：

- Select操作从表中读取数据，要读的列可以用列名指定，或者用*代表所有的列，一个简单的Select语句，如下所示：

  ```
  select * from sale_detail;
  ```

  若您只读取sale_detail的一列shop_name，如下所示：

  ```
  select shop_name from sale_detail;
  ```

  在where中可以指定过滤的条件，如下所示：

  ```
  select * from sale_detail where shop_name like 'hang%';
  ```

  当使用Select语句屏显时，目前最多只能显示10000行结果。当Select作为子句时，无此限制，Select子句会将全部结果返回给上层查询。

- select分区表时禁止全表扫描。

  2018-01-10 20点后创建的新项目，默认情况下执行SQL时，针对该project里的分区表不允许全表扫描，必须有分区条件指定需要扫描的分区，由此减少SQL的不必要I/O，从而减少计算资源的浪费，同时也减少了不必要的后付费模式的计算费用（后付费模式中，数据输入量是计量计费参数之一）。

  例如表定义是`t1(c1,c2) partitioned by(ds)`，在新项目里执行如下语句会被禁止，返回error：

  ```
  Select * from t1 where c1=1;
  Select * from t1 where (ds='20180202' or c2=3);
  Select * from t1 left outer join t2 on a.id =b.id and a.ds=b.ds and
  b.ds='20180101);
  --Join进行关联时，若分区剪裁条件放在where中，则分区剪裁生效，若放在on条件中，从表的分区剪裁会生效，主表则进行全表扫描。
  ```

  若实在需要对分区表进行全表扫描，可以在对分区表全表扫描的SQL语句前加一个set语句`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，则要全表扫描需同时提交如下简单查询命令：

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

```
select * from sale_detail;
```

如果需要整个项目都允许全表扫描，可以通过开关自行打开或关闭（true/false），命令如下：

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

- 在table_reference中支持使用嵌套子查询，如下所示：

```
select * from (select region from sale_detail) t where region = '
shanghai';
```

- where子句支持的过滤条件，如下表所示：

| 过滤条件 | 描述 |
| --- | --- |
| > 、 < 、 =、 >=、 <=、 <> | 关系操作符 |
| like、rlike | like和rlike的source和pattern参数均仅接受String类型。 |
| in、not in | 如果在in/not in条件后加子查询，子查询只能返回一列值，且返回值的数量不能超过1000。 |

在Select语句的where子句中，您可以指定分区范围，这样可以仅仅扫描表的指定部分，避免全表扫描。如下所示：

```
SELECT sale_detail.* FROM sale_detail WHERE sale_detail.sale_date >=
 '2008' AND sale_detail.sale_date <= '2014';
```

MaxCompute SQL的where子句支持between…and条件查询，上述SQL可以重写如下：

```
SELECT sale_detail.* FROM sale_detail WHERE sale_detail.sale_date
BETWEEN '2008' AND  '2014';
```

- **distinct**：如果有重复数据行时，在字段前使用distinct，会将重复字段去重，只返回一个值，而使用all将返回字段中所有重复的值，不指定此选项时默认效果和all相同。

  使用distinct只返回一行记录，如下所示：

```
select distinct region from sale_detail;
select distinct region, sale_date from sale_detail;
-- distinct多列，distinct的作用域是 Select 的列集合，不是单个列。
```

- **group by**：分组查询，一般group by和聚合函数配合使用。在Select中包含聚合函数时有以下规则：

  — 用group by的key可以是输入表的列名。

  — 也可以是由输入表的列构成的表达式，不允许是Select语句的输出列的别名。

— 规则i的优先级高于规则ii。当规则i和规则ii发生冲突时，即group by的key既是输入表的列或表达式，又是Select的输出列，以规则i为准。

示例如下：

```
select region from sale_detail group by region;
-- 直接使用输入表列名作为group by的列，可以运行
select sum(total_price) from sale_detail group by region;
-- 以region值分组，返回每一组的销售额总量，可以运行
select region, sum(total_price) from sale_detail group by region;
-- 以region值分组，返回每一组的region值(组内唯一)及销售额总量，可以运行
select region as r from sale_detail group by r;
-- 使用select列的别名运行，报错返回
select 2 + total_price as r from sale_detail group by 2 + total_pric
e;
-- 必须使用列的完整表达式
select region, total_price from sale_detail group by region;
-- 报错返回，select的所有列中，没有使用聚合函数的列，必须出现在group by中
select region, total_price from sale_detail group by region,
total_price;
-- 可以运行
```

之所以有这样的限制，是因为在SQL解析中，group by操作通常是先于Select操作的，因此group by只能接受输入表的列或表达式为key。

**说明：**

关于聚合函数的详情请参见聚合函数。

- order by：对所有数据按照某几列进行全局排序。如果您希望按照降序对记录进行排序，可以使用DESC关键字。由于是全局排序，**order by必须与limit共同使用**。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。

  与group by不同，order by后面必须加Select列的别名，当Select某列时，如果没有指定列的别名，将列名作为列的别名。

```
select * from sale_detail order by region;
-- 报错返回，order by没有与limit共同使用
select * from sale_detail order by region limit 100;
select region as r from sale_detail order by region limit 100;
-- 报错返回，order by后面必须加列的别名。
select region as r from sale_detail order by r limit 100;
```

`[limit number]`的number是常数，限制输出行数。当使用无limit的Select语句直接从屏幕输出查看结果时，最多只输出10000行。每个项目空间的这个屏显最大限制限制可能不同，可以通过**`setproject`**命令控制。

- distribute by：对数据按照某几列的值做Hash分片，必须使用Select的输出列别名。

```
select region from sale_detail distribute by region;
-- 列名即是别名，可以运行
select region as r from sale_detail distribute by region;
-- 报错返回，后面必须加列的别名。
select region as r from sale_detail distribute by r;
```

- sort by：局部排序，语句前必须加distribute by。实际上sort by是对distribute by的结果进行局部排序。必须使用Select的输出列别名。

```
select region from sale_detail distribute by region sort by region;
select region as r from sale_detail sort by region;
-- 没有distribute by，报错退出。
```

- order by不和distribute by/sort by共用，同时group by也不和distribute by/sort by共用，必须使用Select的输出列别名。

 说明：

- order by/sort by/distribute by的key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/sort by/distribute by是后于Select操作的，因此它们只能接受Select语句的输出列为key。

## 1.6.2 Select语序

按照Select语法格式书写的Select语句，实际上的逻辑执行顺序与标准的书写语序实际并不相同，如下语句：

```
SELECT key, max(value) FROM src t WHERE value > 0 GROUP BY key HAVING
sum(value) > 100 ORDER BY key LIMIT 100;
```

实际上的逻辑执行顺序是`FROM->WHERE->GROUY BY->HAVING->SELECT->ORDER BY->LIMIT`。order by中只能引用Select列表中生成的列，而不是访问From的源表中的列。Having可以访问的是group by key和聚合函数。Select的时候，如果有group by，便只能访问group key和聚合函数，而不是From中源表中的列。

为了避免混淆，MaxCompute支持以执行顺序书写查询语句，例如上面的语句可以写为：

```
FROM src t WHERE value > 0 GROUP BY key HAVING sum(value) > 100 SELECT
 key, max(value) ORDER BY key LIMIT 100;
```

## 1.6.3 子查询

**子查询基本定义**

普通的Select是从几张表中读数据，如`select column_1, column_2 … from table_name`，但查询的对象也可以是另外一个Select操作，如下所示：

```
select * from (select shop_name from sale_detail) a;
```

下面这种用法也可以。但是有限制：只能返回一条记录。

```
select (select a from table1) from table2;
```

**说明：**

子查询必须要有别名。

在from子句中，子查询可以当作一张表来使用，与其它的表或子查询进行Join操作，如下所示：

```
create table shop as select * 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;
```

**IN SUBQUERY / NOT IN SUBQUERY**

IN SUBQUERY与LEFT SEMI JOIN类似。

示例如下：

```
SELECT * from mytable1 where id in (select id from mytable2);
--等效于
SELECT * from mytable1 a LEFT SEMI JOIN mytable2 b on a.id=b.id;
```

目前MaxCompute不仅支持IN SUBQUERY，还支持correlated条件。

示例如下：

```
SELECT * from mytable1 where id in (select id from mytable2 where
value = mytable1.value);
```

其中子查询中的`where value = mytable1.value`即是一个correlated条件，旧版MaxCompute对于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时，会报错。现

在**MaxCompute**已经支持这种用法，这样的过滤条件事实上构成了SEMI JOIN中ON条件的一部分。

NOT IN SUBQUERY类似于LEFT ANTI JOIN，但是也有显著不同。

示例如下：

```
SELECT * from mytable1 where id not in (select id from mytable2);
--如果mytable2中的所有id都不为NULL，则等效于
SELECT * from mytable1 a LEFT ANTI JOIN mytable2 b on a.id=b.id;
```

如果mytable2中有任何为Null的列，则not in表达式会为Null，导致where条件不成立，无数据返回，此时与LEFT ANTI JOIN不同。

MaxCompute 1.0版本也支持[NOT] IN SUBQUERY不作为JOIN条件，例如出现在非WHERE语句中，或者虽然在Where语句中，但无法转换为Join条件。当前MaxCompute 2.0版本仍然支持这种用法，但是此时因为无法转换为SEMI JOIN而必须实现启动一个单独的作业来运行SUBQUERY，所以不支持correlated条件。

示例如下：

```
SELECT * from mytable1 where id in (select id from mytable2) OR value
 > 0;
```

因为Where中包含了or，导致无法转换为SEMI JOIN，会单独启动作业执行子查询。

另外在处理分区表的时候，也会有特殊处理：

```
SELECT * from sales_detail where ds in (select dt from sales_date);
```

其中的ds如果是分区列，则`select dt from sales_date`会单独启动作业执行子查询，而不会转化为SEMIJOIN，执行后的结果会逐个与ds比较，sales_detail中ds值不在返回结果中的分区不会读取，保证分区裁剪仍然有效。

**EXISTS SUBQUERY/NOT EXISTS SUBQUERY**

EXISTS SUBQUERY时，当SUBQUERY中有至少一行数据时，返回true，否则false。NOT EXISTS时则相反。

目前只支持含有correlated WHERE条件的子查询。EXISTS SUBQUERY/NOT EXISTS SUBQUERY实现的方式是转换为LEFT SEMI JOIN或者LEFT ANTI JOIN。

示例如下：

```
SELECT * from mytable1 where exists (select * from mytable2 where id
 = mytable1.id);
--等效于
SELECT * from mytable1 a LEFT SEMI JOIN mytable2 b on a.id=b.id;
```

而

```
SELECT * from mytable1 where not exists (select * from mytable2 where
id = mytable1.id);
--等效于
SELECT * from mytable1 a LEFT ANTI JOIN mytable2 b on a.id=b.id;
```

**SCALAR SUBQUERY**

当SUBQUERY的输出结果为单行单列的时候，可以当做标量来使用。如：

```
select * from t1 where (select count(*)  from t2 where t1.a = t2.a) >
1;
-- 等效于
select t1.* from t1 left semi join (select a, count(*) from t2 group
by a having count(*) > 1) t2 on t1 .a = t2.a;
```

语句`select count(*) from t2 where t1.a = t2.a;`的输出结果是一个row set，但可以判断这条语句的输出有且仅有一行一列，因此可以将其当做标量使用，即可以参与标量运算（‘>’）。但在实现过程中，实际会尽可能地转成join来处理，如等效的第二条语句。

注意：能当成标量来使用的SUBQUERY必须是在编译阶段就能够确认返回结果只有一行一列的查询，如果一条语句，即使能够确定在实际运行过程中只会产生一行数据，但是编译过程中确定不了，编译器也是会报错。

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，且不是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。

同时还有两个限制：

- 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。如：

  ```
  select * from t1 where (select count(*) from t2 where t1.a = t2.a)
   = 3;--允许
  ```

```
 select * from t1 where (select count(*) from t2 where (select count
(*) from t3 where t3.a = t1.a) = 2) = 3;-- 不允许，不能在子查询的子查询中
引用外部查询的列
```

- scalar subquery只能where中使用。比如下面是不允许的

```
select * from t1 where (select t1.b + count(*) from t2) = 3;  -- 不能
在子查询的select 中引用
  select (select count(*) from t2 where t2.a = t1.a)  from t1;
 -- 不能在外层查询的select 中引用
```

实际上所有的满足一行一列输出值的子查询都可以类似上述示例进行重写（如果查询的结果只有一行，在外面包一层MAX或MIN操作，其结果不变）。

## 1.6.4 UNION ALL/UNION [DISTINCT]

命令格式如下：

```
select_statement UNION ALL select_statement;
select_statement UNION [DISTINCT] select_statement;
```

- **UNION ALL**：将两个或多个Select操作返回的数据集联合成一个数据集，如果结果有重复行时，会返回所有符合条件的行，不进行重复行的去重处理。
- **UNION [DISTINCT]**：其中DISTINCT可忽略。将两个或多个Select操作返回的数据集联合成一个数据集，如果结果有重复行时，将进行重复行的去重处理。

UNION ALL示例如下：

```
select * from sale_detail where region = 'hangzhou'
        union all
select * from sale_detail where region = 'shanghai';
```

UNION示例如下：

```
SELECT * FROM src1 UNION SELECT * FROM src2;
--执行的效果相当于
SELECT DISTINCT * FROM (SELECT * FROM src1 UNION ALL SELECT * FROM
src2) t;
```

说明：

- union all/union操作对应的各个查询的列个数、名称和类型必须一致。如果列名不一致时，可以使用列的别名加以解决。
- 一般情况下，MaxCompute最多允许256个表的union all/union，超过此限制报语法错误。

关于UNION后LIMIT的语义，如下所示：

UNION后如果有CLUSTER BY、DISTRIBUTE BY、SORT BY、ORDER BY或者LIMIT子句，其作用于前面所有UNION的结果，而不是UNION的最后一个select_statement。MaxCompute目前在`set odps.sql.type.system.odps2=true;`时，也采用此行为。

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;
```

返回结果如下所示：

```
+------+
| a    |
+------+
| 0    |
| 1    |
| 2    |
+------+
```

## 1.6.5 JOIN操作

MaxCompute的JOIN支持多路链接，但不支持笛卡尔积，即无on条件的链接。

命令格式如下：

```
join_table:
        table_reference join table_factor [join_condition]
        | table_reference {left outer|right outer|full outer|inner}
join table_reference join_condition
    table_reference:
        table_factor
        | join_table
    table_factor:
        tbl_name [alias]
        | table_subquery alias
        | ( table_references )
    join_condition:
        on equality_expression ( and equality_expression )*
```

说明：

equality_expression是一个等式表达式。

**`left join`**：左连接，会从左表（shop）中返回所有的记录，即使在右表（sale_detail）中没有匹配的行。

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
        left outer join sale_detail b on a.shop_name=b.shop_name;
```

```
    -- 由于表shop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。
```

**right outer join**：右连接，返回右表中的所有记录，即使在左表中没有记录与它匹配。

示例如下：

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
        right outer join sale_detail b on a.shop_name=b.shop_name;
```

**full outer join**：全连接，返回左右表中的所有记录。

示例如下：

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
        full outer join sale_detail b on a.shop_name=b.shop_name;
```

在表中存在至少一个匹配时，inner join返回行。关键字inner可省略。

```
select a.shop_name from shop a inner join sale_detail b on a.shop_name
=b.shop_name;
select a.shop_name from shop a join sale_detail b on a.shop_name=b.
shop_name;
```

连接条件，只允许and连接的等值条件。只有在MAPJOIN中，可以使用不等值连接或者使用or连接多个条件。

```
select a.* from shop 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;
    -- 支持多路join链接示例
select a.* from shop a join sale_detail b on a.shop_name != b.
shop_name;
    -- 不支持不等值Join链接条件，报错返回。
```

IMPLICIT JOIN，MaxCompute支持如下Join方式：

```
SELECT * FROM table1, table2 WHERE table1.id = table2.id;
--执行的效果相当于
SELECT * FROM table1 JOIN table2 ON table1.id = table2.id;
```

## 1.6.6 SEMI JOIN

MaxCompute支持SEMI JOIN（半连接）。SEMI JOIN中，右表只用来过滤左表的数据而不出现在结果集中。支持LEFT SEMI JOIN和LEFT ANTI JOIN两种语法。

#### LEFT SEMI JOIN

当Join条件成立时，返回左表中的数据。也就是mytable1中某行的Id在mytable2的所有Id中出现过，此行就保留在结果集中。

示例如下：

```
SELECT * from mytable1 a LEFT SEMI JOIN mytable2 b on a.id=b.id;
```

只会返回mytable1中的数据，只要mytable1的Id在mytable2的Id中出现。

#### LEFT ANTI JOIN

当Join条件不成立时，返回左表中的数据。也就是mytable1中某行的Id在mytable2的所有Id中没有出现过，此行便保留在结果集中。

示例如下：

```
SELECT * from mytable1 a LEFT ANTI JOIN mytable2 b on a.id=b.id;
```

只会返回mytable1中的数据，只要mytable1的Id在mytable2的Id没有出现。

## 1.6.7 MAPJOIN HINT

当一个大表和一个或多个小表做Join时，可以使用MapJoin，性能比普通的Join要快很多。MapJoin的基本原理为：在小数据量情况下，SQL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，从而加快Join的执行速度。

 说明：

当您使用 MapJoin 时，要注意以下问题：

- left outer join的左表必须是大表。
- right outer join的右表必须是大表。
- inner join左表或右表均可以作为大表。
- full outer join不能使用MapJoin。
- MapJoin支持小表为子查询。
- 使用MapJoin时，需要引用小表或是子查询时，需要引用别名。
- 在MapJoin中，可以使用不等值连接或者使用or连接多个条件。
- 目前，MaxCompute在MapJoin中最多支持指定8张小表，否则报语法错误。

- 如果使用MapJoin，则所有小表占用的内存总和不得超过512MB。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512MB限制是加载到内存后的空间大小。
- 多个表Join时，最左边的两个表不能同时是MapJoin的表。

示例如下：

```
select /* + mapjoin(a) */
        a.shop_name,
        b.customer_id,
        b.total_price
    from shop a join sale_detail b
    on a.shop_name = b.shop_name;
```

MaxCompute SQL不支持在普通Join的on条件中使用不等值表达式，or逻辑等复杂的Join条件，但是在MapJoin中可以进行如上操作。

示例如下：

```
select /*+ mapjoin(a) */
        a.total_price,
        b.total_price
    from shop a join sale_detail b
    on a.total_price < b.total_price or a.total_price + b.total_price
 < 500;
```

## 1.6.8 HAVING子句

由于MaxCompute SQL的Where关键字无法与合计函数一起使用，可以采用HAVING子句。

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
```

示例如下：

比如有一张订单表Orders，包括客户名称（Customer），订单金额（OrderPrice），订单日期（Order_date），订单号（Order_id）四个字段。现在希望查找订单总额少于2000的客户。SQL语句如下所示：

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
```

```
HAVING SUM(OrderPrice)<2000
```

## 1.6.9 Explain

MaxCompute SQL提供Explain操作，用来显示对应于DML语句的最终执行计划结构的描述。所谓执行计划就是最终用来执行SQL语义的程序。

命令格式如下：

```
EXPLAIN <DML query>;
```

Explain的执行结果包含如下内容：

- 对应于该DML语句的所有Task的依赖结构。
- Task中所有Task的依赖结构。
- Task中所有Operator的依赖结构。

示例如下：

```
EXPLAIN
SELECT abs(a.key), b.value FROM src a JOIN src1 b ON a.value = b.value
;
```

Explain的输出结果会有以下三个部分：

- 首先是Job间的依赖关系：`job0 is root job`，因为该query只需要一个Job（job0），所以只需要一行信息。
- 其次是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和M2_Stg1这两个Task会先执行，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，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在所有对应当前query的Task中是唯一的。

- 之后用下划线分隔的数字代表当前Task的直接依赖，如J3_1_2_Stg1意味着当前Task（ID为3）依赖ID为1和ID为2的两个Task。

- 第三部分即Task中的Operator结构，Operator串描述了一个Task的执行语义。

```
In Task M1_Stg1:
  Data source: yudi_2.src                          # "Data source"描述了
当前Task的输入内容
  TS: alias: a                                     # TableScanOperator
      RS: order: +                                 # ReduceSinkOperator
          keys:
               a.value
          values:
               a.key
          partitions:
               a.value
In Task J3_1_2_Stg1:
  JOIN: a INNER JOIN b                             # JoinOperator
      SEL: Abs(UDFToDouble(a._col0)), b._col5      # SelectOperator
          FS: output: None                         # FileSinkOperator
In Task M2_Stg1:
  Data source: yudi_2.src1
  TS: alias: b
      RS: order: +
          keys:
               b.value
          values:
               b.value
          partitions:
               b.value
```

  - 各Operator的含义，如下所示：

    - **TableScanOperator**：描述了query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**：描述了query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，会显示成`< alias >.< column_name >`。
      - 如果是表达式的结果，会显示函数的形式，如`func1(arg1_1, arg1_2, func2(arg2_1, arg2_2))`。
      - 如果是常量，则直接显示值内容。
    - **FilterOperator**：描述了query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**：描述了query语句中的Join语句块的逻辑，Explain结果中会显示哪些表用哪种方式Join在一起。

- **GroupByOperator**：描述了聚合操作的逻辑，如果query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**：描述了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来执行数据分发操作。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**：描述了最终数据的存储操作，如果query中有Insert语句块，Explain结果中会显示目标表名称。
- **LimitOperator**：描述了query语句中的limit 语句块的逻辑，Explain结果中会显示limit数。
- **MapjoinOperator**：类似**JoinOperator**，描述了大表的Join操作。

说明：

如果query足够复杂，Explain的结果太多，会导致触发API的限制，使得您看到的Explain结果不完整。这时候可以通过拆分query，各部分分别Explain，来了解Job的结构。

## 1.6.10 Common Table Expression（CTE）

MaxCompute支持SQL标准的CTE，提高SQL语句的可读性与执行效率。

命令格式如下所示：

```
WITH
     cte_name AS
    (
        cte_query
    )
    [,cte_name2  AS
     (
     cte_query2
     )
    ,……]
```

- **cte_name**：指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，都是指CTE。
- **cte_query**：是一个Select语句，它产生的结果集用于填充CTE。

示例如下：

```
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。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：

```
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。CTE的with子句中可以指定多个子查询，像使用变量一样在整个语句中反复重用。除重用外，也不必反复嵌套。

## 1.7 Select Transform语法

Select Transform功能允许您指定启动一个子进程，将输入数据按照一定的格式通过stdin输入子进程，并且通过parse子进程的stdout输出，来获取输出数据。适用于实现MaxCompute SQL没有的功能又不想写UDF的场景。

命令格式如下所示：

```
SELECT TRANSFORM(arg1, arg2 ...)
(ROW FORMAT DELIMITED (FIELDS TERMINATED BY field_delimiter (ESCAPED
BY character_escape)?)?
(LINES SEPARATED BY line_separator)?
(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)?)?
```

说明如下：

- SELECT TRANSFORM关键字可以用MAP关键字或者REDUCE关键字来替换，无论使用哪个关键字语义是完全一样的。为了使语法更清晰，推荐您使用SELECT TRANSFORM。
- arg1,arg2...是transform的参数，其格式和select子句的item类似。默认的格式下，参数的各个表达式的结果会在隐式转换成string后，用\t拼起来，输入到子进程中（此格式可以进行配置，详情请参见下文对ROW FORMAT的说明）。
- Using指定要启动的子进程的命令。

  说明：

  - 大多数的MaxCompute SQL命令Using子句指定的是资源（Resources），但此处是为了和Hive的语法兼容。
  - Using中的格式和Shell的语法非常类似，但并非真的启动Shell来执行，而是直接根据命令的内容来创建了子进程，所以很多Shell的功能不能用，比如输入输出重定向，管道，循环等。若有需要，Shell本身也可以作为子进程命令来使用。
- RESOURCES子句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。
  - 支持使用resources子句：如using 'sh foo.sh bar.txt' Resources 'foo.sh','bar.txt'。
  - 支持在SQL语句前使用set odps.sql.session.resources=foo.sh,bar.txt;来指定。注意这种配置是全局的，意味着整个SQL中所有的select transform都可以访问这个setting配置的资源。
- ROW FORMAT子句允许自定义输入输出的格式。

  语法中有两个row format子句，第一个子句指定输入的格式，第二个指定输出的格式。 默认情况下使用\t来作为列的分隔符，\n作为行的分隔符，Null使用\N（注意是两个字符，反斜杠字符和字符N）来表示。

  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。

- Hive指定格式的各种语法，如inputRecordReader、outputRecordReader、Serde等，MaxCompute也都支持，不过需要打开Hive兼容模式才能用，即在SQL语句前加set语句`set odps.sql.hive.compatible=true;`，详情请参见*Hive*的文档。
- 若使用Hive的inputRecordReader、outputRecordReader等自定义类，可能会降低执行性能。

- AS子句指定输出列。
  - 输出列可以不指定类型，默认为String类型，如as(col1, col2)。也可以指定类型，如as(col1: bigint, col2:boolean)。
  - 由于输出实际是parse子进程stdout获取的，如果指定的类型不是String，系统会隐式调用Cast函数，而Cast有可能出现runtime exception。
  - 输出列类型不支持部分指定部分不指定，如as(col1, col2:bigint)。
  - as可以省略，此时默认stdou的输出中第一个\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脚本示例**

假设通过Shell脚本生成50行数据，值是从1到50，对应data字段输出：

```
SELECT  TRANSFORM(script) USING 'sh' AS (data) FROM (SELECT  'for i in
 `seq 1 50`; do echo $i; done' AS script) t;
```

直接将Shell命令作为transform数据输入。

select transform不仅仅是语言支持的扩展，一些简单的功能，如awk、python、perl、shell都支持直接在命令里面写脚本，不需要写脚本文件，上传资源等，开发过程更简单，如上述示例所示。当然，功能复杂的可以上传脚本文件来执行，如下文将介绍的python示例。

**调用Python脚本示例**

准备好Python文件，假设脚本文件名为myplus.py，如下所示：

```
#!/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 int(token[0]) + int(token[1])
```

```
line = sys.stdin.readline()
```

将该Python脚本文件添加为MaxCompute资源（Resource）：

```
add py ./myplus.py -f;
```

您也可通过DataWorks控制台进行新增资源操作。

接下来使用select transform语法调用资源。

```
Create table testdata(c1 bigint,c2 bigint);--创建测试表
insert into Table testdata values (1,4),(2,5),(3,6);--测试表中插入测试数
据
--接下来执行select transform如下：
SELECT TRANSFORM (testdata.c1, testdata.c2) USING 'python myplus.py'
resources 'myplus.py' AS (result bigint) FROM testdata;
-- 或者
set odps.sql.session.resources=myplus.py;
SELECT TRANSFORM (testdata.c1, testdata.c2) USING 'python myplus.py'
AS (result bigint) FROM testdata;
```

执行结果如下：

```
+-----+
| cnt |
+-----+
| 5   |
| 7   |
| 9   |
+-----+
```

Python脚本无需依赖MaxCompute的Python框架，也没有格式要求。

也支持直接将py命令作为transform数据输入，如Shell的例子也可以用py命令实现：

```
SELECT  TRANSFORM('for i in xrange(1, 50):  print i;') USING 'python'
AS (data);
```

**调用Java脚本示例**

与前面调用Python脚本类似，编辑好Java文件导出Jar包，再通过add file方式将Jar包加为MaxCompute资源，然后select transform调用。

准备好Jar文件，假设脚本文件名为Sum.jar，Java代码如下所示：

```
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]));
    }
  }
}
```

将Jar文件添加为MaxCompute的Resource。

```
add jar ./Sum.jar -f;
```

接下来使用select transform语法调用资源。

```
Create table testdata(c1 bigint,c2 bigint);--创建测试表
insert into Table testdata values (1,4),(2,5),(3,6);--测试表中插入测试数
据
--接下来执行select transform如下：
SELECT TRANSFORM(testdata.c1, testdata.c2) USING 'java -cp Sum.jar com
.aliyun.odps.test.Sum' resources 'Sum.jar' from testdata;
--或者
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;
```

执行结果如下所示：

```
+-----+
| cnt |
+-----+
| 5   |
| 7   |
| 9   |
+-----+
```

很多Java的utility可以直接拿来运行。

**说明：**

Java和Python虽然有现成的udtf框架，但是用select transform编写更简单，并且不需要额外依赖，也没有格式要求，甚至可以实现离线脚本拿来直接就用（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）。

**调用其他脚本语言**

select transform不仅仅支持上述语言扩展，还支持其它的常用unix命令或脚本解释器，如awk、perl等。

以调用awk，把第二列原样输出为例，如下所示：

```
SELECT TRANSFORM(*) USING "awk '//{print $2}'" as (data) from testdata
;
```

perl示例如下：

```
SELECT TRANSFORM (testdata.c1, testdata.c2) USING "perl -e 'while($
input = <STDIN>){print $input;}'" FROM testdata;
```

说明：

目前由于MaxCompute计算集群上没有PHP和Ruby，所以不支持调用这两种脚本，后期系统会努力改进争取能支持PHP和Ruby。

串联使用示例

select transform还可以串联使用，如使用distribute by和sort by对输入数据做预处理。

```
SELECT TRANSFORM(key, value) USING 'cmd2' from
(
    SELECT TRANSFORM(*) USINg 'cmd1' from
    (
        SELECt * FROM data distribute by col2 sort by col1
    ) t distribute by key sort by value
) t2;
```

或者用map、reduce的关键字，可能更符合某些用户的认知习惯（如前文说明，无论使用哪个关键字，语义完全一样）。

```
@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;
```

**Select Transform性能介绍**

性能上，Select Transform与UDTF在不同场景效果也不同。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。

由于transform的开发更加简便，所以Select Transform更适合做adhoc的数据分析。

**UDTF的优势**

- UDTF的输出结果和输入参数是有类型的，而Transform的子进程基于stdin/stdout传输数据，所有数据都当做string处理，因此transform多了一步类型转换。
- Transform数据传输依赖于操作系统的管道，而目前管道的buffer仅有4KB，且不能设置，transform读空/写满pipe会导致进程被挂起。

- UDTF的常量参数可以不用传输，而Transform没办法利用这个优化。

**Select Transform的优势**

- 子进程和父进程是两个进程，而UDTF是单线程的，如果计算占比比较高，数据吞吐量比较小，可以利用服务器的多核特性。
- 数据的传输通过更底层的系统调用来读写，效率比Java高。
- Select Transform支持的某些工具，如awk，是natvie代码实现的，和Java相比，理论上会有性能优势。

# 1.8 SQL限制项汇总

一些用户因没注意限制条件，业务启动后才发现限制条件，导致业务停止。为避免此类现象发生，方便用户查看，本文将对MaxCompute SQL限制项做以下汇总：

| 边界名 | 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，只能用英文的a-z,A-Z及数字和下划线_，且以字母开头 |
| 注释长度 | 1024字节 | 长度限制 | 注释内容是长度不超过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1200个 |
| 单表分区数 | 60000 | 数量限制 | 一张表最多允许60000个分区 |
| 表的分区层级 | 6级 | 数量限制 | 在表中建的分区层次不能超过6级 |
| 表统计定义个数 | 100个 | 数量限制 | 表统计定义个数 |
| 表统计定义长度 | 64000 | 长度限制 | 表统计项定义长度 |
| 屏显 | 10000行 | 数量限制 | SELECT语句屏显默认最多输出10000行 |
| insert目标个数 | 256个 | 数量限制 | multiins同时insert的数据表数量 |
| UNION ALL | 256个表 | 数量限制 | 最多允许256个表的UNION ALL |
| MAPJOIN | 8个小表 | 数量限制 | MAPJOIN最多允许8张小表 |
| MAPJOIN内存限制 | 512M | 数量限制 | MAPJOIN所有小表的内存限制不能超过512M |
| 窗口函数 | 5个 | 数量限制 | 一个SELECT中最多允许5个窗口函数 |

| 边界名 | 最大值/限制条件 | 分类 | 说明 |
| --- | --- | --- | --- |
| ptinsubq | 1000行 | 数量限制 | 一个 partition 列 in subquery 时，subquery 的返回结果不可超过 1000 行。 |
| sql语句长度 | 2M | 长度限制 | 允许的sql语句的最大长度 |
| wherer子句条件个数 | 256个 | 数量限制 | where子句中可使用条件个数 |
| 列记录长度 | 8M | 数量限制 | 表中单个cell的最大长度 |
| in的参数个数 | 1024 | 数量限制 | in的最大参数限制，如in (1,2,3…,1024)。in(…)如果参数过多，会造成编译时的压力；1024是建议值、不是限制值 |
| jobconf.json | 1M | 长度限制 | jobconf.json的大小为1M。当表包含的Partition数量太多时，可能超过jobconf.json超过1M。 |
| 视图 | 不可写 | 操作限制 | 视图不可以写，不可用insert操作 |
| 列的数据类 | 不允许 | 操作限制 | 不允许修改列的数据类型，列位置 |
| java udf函数 | 不能是abstract或者static | 操作限制 | java udf函数不能是 abstract 或者static |
| 最多查询分区个数 | 10000 | 数量限制 | 最多查询分区个数不能超过10000 |

**说明：**

以上MaxCompute SQL限制项均不可以被人为修改配置。

## 1.9 Lateral View

### 单个Lateral View语句

语法定义如下：

```
lateralView: LATERAL VIEW [OUTER] udtf(expression) tableAlias AS
columnAlias (',' columnAlias) * fromClause: FROM baseTable (lateralVie
w)*
```

说明如下：

- Lateral view通常和split、explode等UDTF一起封装使用，它能够将一行数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。
- Lateral view首先为原始表的每行调用UDTF，UDTF会把一行拆分成一行或者多行，Lateral view再把结果聚合，产生一个支持别名表的虚拟表。
- Lateral view outer：当table function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table function输出列为null。

示例如下：

假设我们有一张表pageAds，它有两列数据，第一列是pageid string，第二列是adid_list，即用逗号分隔的广告ID集合。

| string pageid | Array<int> adid_list |
| --- | --- |
| “front_page” | [1, 2, 3] |
| “contact_page” | [3, 4, 5] |

需求是要统计所有广告ID在所有页面中出现的次数，实现过程如下所示。

1. 拆分广告ID，如下所示：

```
SELECT pageid, adid
 FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid;
```

执行结果如下：

| string pageid | int adid |
| --- | --- |
| “front_page” | 1 |
| “front_page” | 2 |
| “front_page” | 3 |
| “contact_page” | 3 |
| “contact_page” | 4 |
| “contact_page” | 5 |

2. 进行聚合的统计，语句如下：

```
SELECT adid, count(1)
    FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid
GROUP BY adid;
```

执行结果如下：

| int adid | count(1) |
| --- | --- |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |

**多个Lateral View语句**

一个from语句后可以跟多个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。

以下面的表为例：

| Array<int> col1 | Array<string> col2 |
| --- | --- |
| [1, 2] | [“a”, “b”, “c”] |
| [3, 4] | [“d”, “e”, “f”] |

- 执行单个语句：

```
SELECT myCol1, col2 FROM baseTable
    LATERAL VIEW explode(col1) myTable1 AS myCol1;
```

执行结果如下所示：

| int mycol1 | Array<string> col2 |
| --- | --- |
| 1 | [“a”, “b”, “c”] |
| 2 | [“a”, “b”, “c”] |
| 3 | [“d”, “e”, “f”] |
| 4 | [d”, “e”, “f”] |

- 加上一个Lateral View语句，如下所示：

```
SELECT myCol1, myCol2 FROM baseTable
    LATERAL VIEW explode(col1) myTable1 AS myCol1
    LATERAL VIEW explode(col2) myTable2 AS myCol2;
```

执行结果如下所示：

| int myCol1 | string myCol2 |
| --- | --- |
| 1 | “a” |
| 1 | “b” |
| 1 | “c” |
| 2 | “a” |
| 2 | “b” |
| 2 | “c” |
| 3 | “d” |
| 3 | “e” |
| 3 | “f” |
| 4 | “d” |
| 4 | “e” |
| 4 | “f” |

# 1.10 内建函数

## 1.10.1 日期函数

MaxCompute SQL提供了针对Datetime类型的操作函数。

**DATEADD**

命令格式如下：

```
datetime dateadd(datetime date, bigint delta, string datepart)
```

命令说明如下：

按照指定的单位datepart和幅度delta修改date的值。

参数说明：

- **date**：Datetime类型，日期值。若输入为String类型会隐式转换为Datetime类型后参与运算，其它类型抛异常。
- **delta**：Bigint类型，修改幅度。若输入为String类型或Double型会隐式转换到Bigint类型后参与运算，其他类型会引发异常。若delta大于0，则增，否则减。

- **datepart**：String类型常量。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。

  关于类型转换的规则，请参见*String类型与Datetime类型之间的转换*。此外也支持扩展的日期格式：年-year，月-month或mon，日-day，小时-hour。非常量、不支持的格式或其它类型会抛异常。

返回值：

返回Datetime类型。若任一输入参数为NULL，返回NULL。

 说明：

- 按照指定的单位增减delta时，导致的对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、60进制进行计算。
- 当delta的单位是月时，计算规则如下：

  若Datetime的月部分在增加delta值之后不造成day溢出，则保持day值不变，否则把day值设置为结果月份的最后一天。
- datepart的取值遵循String与Datetime类型转换的约定，即yyyy表示年，mm表示月，Datetime相关的内建函数如无特殊说明均遵守此约定。同时如果没有特殊说明，所有Datetime相关的内建函数的part部分也同样支持扩展的日期格式：年-year，月-month或mon，日-day，小时-hour。

示例如下：

```
若trans_date = 2005-02-28 00:00:00：
dateadd(trans_date, 1, 'dd') = 2005-03-01 00:00:00
-- 加一天，结果超出当年2月份的最后一天，实际值为下个月的第一天
dateadd(trans_date, -1, 'dd') = 2005-02-27 00:00:00
-- 减一天
dateadd(trans_date, 20, 'mm') = 2006-10-28 00:00:00
-- 加20个月，月份溢出，年份加1
若trans_date = 2005-02-28 00:00:00, dateadd(transdate, 1, 'mm') = 2005
-03-28 00:00:00
若trans_date = 2005-01-29 00:00:00, dateadd(transdate, 1, 'mm') = 2005
-02-28 00:00:00
-- 2005年2月没有29日，日期截取至当月最后一天
若trans_date = 2005-03-30 00:00:00, dateadd(transdate, -1, 'mm') =
2005-02-28 00:00:00
```

 说明：

此处对trans_date的数值表示仅作示例使用，在文档中有关Datetime的介绍会经常使用到这种简易的表达方式。

在MaxCompute SQL中，Datetime类型没有直接的常数表示方式，如下使用方式是错误的：

```
select dateadd(2005-03-30 00:00:00, -1, 'mm') from tbl1;
```

如果一定要描述Datetime类型常量，请尝试如下方法：

```
select dateadd(cast("2005-03-30 00:00:00" as datetime), -1, 'mm') from
 tbl1;
-- 将String类型常量显式转换为Datetime类型
```

**DATEDIFF**

命令格式如下：

```
bigint datediff(datetime date1, datetime date2, string datepart)
```

命令说明如下：

计算两个时间date1、date2在指定时间单位datepart的差值。

参数说明：

- **datet1**、**date2**：Datetime类型，被减数和减数，若输入为String类型会隐式转换为Datetime类型后参与运算，其它类型抛异常。
- **datepart**：String类型常量。支持扩展的日期格式。若datepart不符合指定格式或者其它类型则会发生异常。

返回值：

返回Bigint类型。任一输入参数是NULL，返回NULL。如果date1小于date2，返回值可以为负数。

 说明：

计算时会按照datepart切掉低单位部分，然后再计算结果。

示例如下：

```
若start = 2005-12-31 23:59:59，end = 2006-01-01 00:00:00:
    datediff(end, start, 'dd') = 1
    datediff(end, start, 'mm') = 1
    datediff(end, start, 'yyyy') = 1
    datediff(end, start, 'hh') = 1
    datediff(end, start, 'mi') = 1
    datediff(end, start, 'ss') = 1
```

```
    datediff('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'ss') =
1800
    datediff('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'mi') = 30
若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：
 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') = 16
```

**DATEPART**

命令格式如下所示：

```
bigint datepart(datetime date, string datepart)
```

命令说明如下：

提取日期date中指定的时间单位datepart的值。

参数说明：

返回值：

- **date**：Datetime类型，若输入为String类型会隐式转换为Datetime类型后参与运算，其它类型抛异常。
- **datepart**：String类型常量，支持扩展的日期格式。若datepart不符合指定格式或者其它类型则会发生异常。

返回Bigint类型。若任一输入参数为NULL，返回NULL。

示例如下：

```
datepart('2013-06-08 01:10:00', 'yyyy')  =  2013
datepart('2013-06-08 01:10:00', 'mm')  =  6
```

**DATETRUNC**

命令格式如下：

```
datetime datetrunc (datetime date, string datepart)
```

命令说明如下：

返回日期date被截取指定时间单位datepart后的日期值。

参数说明：

- **date**：Datetime类型，若输入为String类型会隐式转换为Datetime类型后参与运算，其它类型抛异常。

- **datepart**：String类型常量，支持扩展的日期格式。若datepart不符合指定格式或者其它类型则会发生异常。

返回值：

Datetime类型。任意一个参数为NULL时，返回NULL。

示例如下：

```
datetrunc('2011-12-07 16:28:46', 'yyyy') = 2011-01-01 00:00:00
datetrunc('2011-12-07 16:28:46', 'month') = 2011-12-01 00:00:00
datetrunc('2011-12-07 16:28:46', 'DD') = 2011-12-07 00:00:00
```

### FROM_UNIXTIME

命令格式如下：

```
datetime from_unixtime(bigint unixtime)
```

命令说明如下：

将数字型的unix时间日期值unixtime转为日期值。

参数说明：

**unixtime**：Bigint类型，秒数，unix格式的日期时间值，若输入为String、Double、Decimal类型会隐式转换为Bigint后参与运算。

返回值：

返回Datetime类型的日期值，unixtime为NULL时，返回NULL。

示例如下：

```
from_unixtime(123456789) = 1973-11-30 05:33:09
```

### GETDATE

命令格式如下：

```
datetime getdate()
```

命令说明如下：

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：

返回当前日期和时间，Datetime类型。

说明：

在一个MaxCompute SQL任务中（以分布式方式执行），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（2.0版本会精确到毫秒）。

### ISDATE

命令格式如下：

```
boolean isdate(string date, string format)
```

命令说明如下：

判断一个日期字符串能否根据对应的格式串转换为一个日期值，如果转换成功，返回TRUE，否则返回FALSE。

参数说明：

- **date**：String格式的日期值，若输入为Bigint、Double、Decimal或Datetime类型，会隐式转换为String类型后参与运算，其它类型报异常。
- **format**：String类型常量，不支持日期扩展格式。其它类型或不支持的格式会抛异常。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。如isdate("1234-yyyy", "yyyy-yyyy")，会返回TRUE。

返回值：

返回Boolean类型，如任意参数为NULL，返回NULL。

### LASTDAY

命令格式如下：

```
datetime lastday(datetime date)
```

命令说明如下：

取date当月的最后一天，截取到天，时分秒部分为00:00:00。

参数说明：

**date**：Datetime类型，若输入为String 类型，会隐式转换为Datetime类型后参与运算，其它类型报异常。

返回值：

返回Datetime类型，如输入为NULL，返回NULL。

### TO_DATE

命令格式如下：

```
datetime to_date(string date, string format)
```

命令说明如下：

将一个format格式的字符串date转成日期值。

参数说明：

- **date**：String类型，要转换的字符串格式的日期值，若输入为Bigint、Double、Decimal或者Datetime类型，会隐式转换为String类型后参与运算，为其它类型则抛异常，为空串时抛异常。
- **format**：String类型常量，日期格式。非常量或其他类型会引发异常。format不支持日期扩展格式，其他字符作为无用字符在解析时忽略。

  **format**参数至少包含yyyy，否则引发异常，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。如`to_date("1234-2234", "yyyy-yyyy")` 会返回1234-01-01 00:00:00。

返回值：

返回Datetime类型，格式为yyyy-mm-dd hh:mi:ss 。若任意一个输入的参数为NULL，则返回NULL值。

示例如下：

```
to_date('阿里巴巴2010-12*03', '阿里巴巴yyyy-mm*dd') = 2010-12-03 00:00:00
to_date('20080718', 'yyyymmdd') = 2008-07-18 00:00:00
to_date('200807182030','yyyymmddhhmi') = 2008-07-18 20:30:00
to_date('2008718', 'yyyymmdd')
-- 格式不符合，引发异常
to_date('阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd')
-- 格式不符合，引发异常
to_date('2010-24-01', 'yyyy')
```

```
-- 格式不符合，引发异常
```

### TO_CHAR

命令格式如下：

```
string to_char(datetime date, string format)
```

命令说明如下：

将日期类型date按照format指定的格式转成字符串。

参数类型：

- **date**：Datetime类型，要转换的日期值，若输入为String类型，会隐式转换为Datetime类型后参与运算，其它类型抛异常。
- **format**：String类型常量。非常量或其他类型会引发异常。format中的日期格式部分会被替换成相应的数据，其它字符直接输出。

返回值：

返回String类型。若任意一个输入的参数为NULL，则返回NULL值。

示例如下：

```
to_char('2010-12-03 00:00:00', '阿里金融yyyy-mm*dd') = '阿里金融2010-12*
03'
to_char('2008-07-18 00:00:00', 'yyyymmdd') = '20080718'
to_char('阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd') -- 引发异常
to_char('2010-24-01', 'yyyy') -- 会引发异常
to_char('2008718', 'yyyymmdd') -- 会引发异常
```

关于其他类型向String类型转换的详情请参见字符串函数>*TO_CHAR*。

### UNIX_TIMESTAMP

命令格式如下：

```
bigint unix_timestamp(datetime date)
```

命令说明如下：

将日期date转化为整型的unix格式的日期时间值。

参数说明：

**date**：Datetime类型日期值，若输入为String类型，会隐式转换为Datetime类型后参与运算，其它类型抛异常。

返回值：

返回Bigint类型，表示unix格式日期值，date为NULL时，返回NULL值。

### WEEKDAY

命令格式如下：

```
bigint weekday (datetime date)
```

命令说明如下：

返回date日期当前周的第几天。

参数说明：

**date**：Datetime类型，若输入为String类型，会隐式转换为Datetime类型后参与运算，其它类型抛异常。

返回值：

返回Bigint类型，若输入参数为NULL，返回NULL值。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。

### WEEKOFYEAR

命令格式如下：

```
bigint weekofyear(datetime date)
```

命令说明如下：

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：

关于这一周算上一年， 还是下一年，主要是看这一周大多数日期（4天以上）在哪一年多。 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**：Datetime类型日期值，若输入为String类型，会隐式转换为Datetime类型后参与运算，其它类型抛异常。

返回值：

返回Bigint类型。若输入为NULL，返回NULL值。

示例如下：

```
select weekofyear(to_date("20141229", "yyyymmdd")) from dual;
返回结果：
+------------+
| _c0        |
+------------+
| 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("20141231", "yyyymmdd")) from dual；--返回结果为1。
 select weekofyear(to_date("20151229", "yyyymmdd")) from dual；--返回结果为53。
```

**MaxCompute2.0扩展函数**

升级到MaxCompute2.0后，产品扩展了部分日期函数。如果您用到的函数涉及新数据类型，在使用新函数的SQL前，需要加一个set语句：

```
set odps.sql.type.system.odps2 = true;--开启新类型
```

若想同时提交，执行以下语句：

```
set odps.sql.type.system.odps2 = true;
select year('1970-01-01 12:30:00') = 1970 from dual;
```

下文将为您介绍新扩展的函数的详情。

**YEAR**

命令格式如下：

```
INT year(string date)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

命令说明如下：

返回一个日期的年。

参数说明：

**date**：String类型日期值，格式至少包含yyyy-mm-dd且不含多余的字符串，否则返回null值。

返回值：

返回INT类型。

示例如下：

```
year('1970-01-01 12:30:00') = 1970
year('1970-01-01') = 1970
year('70-01-01') = 70
year(1970-01-01) = null
year('1970/03/09') = null
year(null) --返回异常
```

### QUARTER

命令格式如下：

```
INT quarter(datetime/timestamp/string date )
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

命令说明如下：

返回一个日期的季度，范围是1-4。

参数说明：

**date**：Datetime/Timestamp/String类型日期值，日期格式至少包含yyyy-mm-dd，其他会返回null值。

返回值：

返回INT类型，输入null，则返回null值。

示例如下：

```
quarter('1970-11-12 10:00:00') = 4
quarter('1970-11-12') = 4
```

**MONTH**

命令格式如下：

```
INT month(string date)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回一个日期的月份。

参数说明：

**date**：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
month('2014-09-01') = 9
month('20140901') = null
```

### DAY

命令格式如下：

```
INT day(string date)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回一个日期的天。

参数说明：

**date**：String类型日期值（格式为yyyy-mm-dd、yyyy-mm-dd hh:mi:ss）其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
day('2014-09-01') = 1
```

```
day('20140901') = null
```

### DAYOFMONTH

命令格式如下：

```
INT dayofmonth(date)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回年/月/日中的具体日期。

例如2017年10月13日，执行命令`int dayofmonth(2017-10-13)`返回结果为13。

参数说明：

**date**：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
dayofmonth('2014-09-01') = 1
```

```
dayofmonth('20140901') = null
```

### HOUR

命令格式如下：

```
INT hour(string date)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回一个日期的小时。

参数说明：

**`date`**：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
hour('2014-09-01 12:00:00') = 12
hour('12:00:00') = 12
```

```
hour('20140901120000') = null
```

**MINUTE**

命令格式如下：

```
INT minute(string date)
```

 说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回一个日期的分钟。

参数说明：

**`date`**：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
minute('2014-09-01 12:30:00') = 30
minute('12:30:00') = 30
```

```
minute('20140901120000') = null
```

### SECOND

命令格式如下：

```
INT second(string date)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回一个日期的秒钟。

参数说明：

**`date`**：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
second('2014-09-01 12:30:45') = 45
second('12:30:45') = 45
```

```
second('20140901123045') = null
```

### CURRENT_TIMESTAMP

命令格式如下：

```
timestamp current_timestamp()
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回当前timestamp类型的时间戳，值不固定。

返回值：

返回timestamp类型。

示例如下：

```
select current_timestamp() from dual;--返回'2017-08-03 11:50:30.661'
```

### ADD_MONTHS

命令格式如下：

```
string add_months(string startdate, int nummonths)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回开始日期startdate增加nummonths个月后的日期。

参数说明：

- **`startdate`**：String类型，格式至少包含年-月-日的日期，否则返回null值。
- **`num_months`**：Int型数值。

返回值：

返回String类型的日期，格式为yyyy-mm-dd。

示例如下：

```
add_months('2017-02-14',3) = '2017-05-14'
add_months('17-2-14',3) = '0017-05-14'
add_months('2017-02-14 21:30:00',3) = '2017-05-14'
add_months('20170214',3) = null
```

### LAST_DAY

命令格式如下：

```
string last_day(string date)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

命令说明如下：

返回该日期所在月份的最后一天日期。

参数说明：

**date**：String类型，格式为yyyy-MM-dd HH:mi:ss或yyyy-MM-dd。

返回值：

返回String类型的日期，格式为yyyy-mm-dd。

示例如下：

```
last_day('2017-03-04') = '2017-03-31'
last_day('2017-07-04 11:40:00') = '2017-07-31'
last_day('20170304') = null
```

### NEXT_DAY

命令格式如下：

```
string next_day(string startdate, string week)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

命令说明如下：

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。

参数说明：

- **startdate**：String类型，格式为yyyy-MM-dd HH:mi:ss或yyyy-MM-dd。
- **week**：String类型，一个星期前2个或3个字母，或者一个星期的全名，如Mo、TUE、FRIDAY。

返回值：

返回String类型的日期，格式为yyyy-mm-dd。

示例如下：

```
next_day('2017-08-01','TU') = '2017-08-08'
next_day('2017-08-01 23:34:00','TU') = '2017-08-08'
next_day('20170801','TU') = null
```

### MONTHS_BETWEEN

命令格式如下：

```
double months_between(datetime/timestamp/string date1, datetime/
timestamp/string date2)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回日期date1和date2之间的月数。

参数说明：

- **date1**：Datetime/Timestamp/String类型，格式为yyyy-MM-dd HH:mi:ss或yyyy-MM-dd。

- **date2**：Datetime/Timestamp/String类型，格式为yyyy-MM-dd HH:mi:ss或yyyy-MM-dd。

返回值：

返回Double类型。

- 当date1晚于date2，返回值为正。当date2晚于date1，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-date2的天数除以31天。

示例如下：

```
months_between('1997-02-28 10:30:00', '1996-10-30') = 3.9495967741
935485
months_between('1996-10-30','1997-02-28 10:30:00' ) = -3.9495967741
935485
months_between('1996-09-30','1996-12-31') = -3.0
```

## 1.10.2 数学函数

### ABS

命令格式如下：

```
Double abs(Double number)
Bigint abs(Bigint number)
Decimal abs(Decimal number)
```

命令说明如下：

该函数用于返回number的绝对值。

参数说明：

**number**：当number为Double、Bigint或Decimal类型时。

- 输入为Bigint，返回Bigint。
- 输入为Double，返回Double类型。
- 输入为Decimal，返回Decimal类型。

若输入为String类型，会隐式转换为Double类型后参与运算，其它类型抛异常。

返回值：

返回Double、Bigint或Decimal类型，取决于输入参数的类型。若输入为null，则返回null。

 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，这种情况下可能会损失精度。

示例如下：

```
abs(null)=null
abs(-1)=1
abs(-1.2)=1.2
abs("-2")=2.0
abs(12232083745629837659238745692374 8)=1.2232083745629837e32
```

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、聚合函数外）的使用方式与其类似，不再举例。

```
select abs(id) from tbl1;
-- 取tbl1表内id字段的绝对值
```

### ACOS

命令格式如下：

```
Double acos(Double number)
Decimal acos(Decimal number)
```

命令说明如下：

该函数用于计算number的反余弦函数。

参数说明：

**number**：Double类型或Decimal类型，-1 ⩽ number ⩽1。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型，值域在0 ~ π之间。若number为null，则返回null。

示例如下：

```
acos("0.87")=0.5155940062460905
acos(0)=1.5707963267948966
```

### ASIN

命令格式如下：

```
Double asin(Double number)
```

```
Decimal asin(Decimal number)
```

命令说明如下：

该函数用于计算number的反正弦函数。

参数说明：

**number**：Double类型或Decimal类型，-1 ⩽ number ⩽1。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型，值域在-π/2 ~π/2之间。若number为null，返回null。

示例如下：

```
asin(1)=1.5707963267948966
asin(-1)=-1.5707963267948966
```

**ATAN**

命令格式如下：

```
Double atan(Double number)
```

命令说明如下：

该函数用于计算number的反正切函数。

参数说明：

**number**：Double类型，若输入为String类型或Bigint类型，会隐式转换到Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型，值域在-π/2~π/2之间。若number为null，则返回null。

示例如下：

```
atan(1)=0.7853981633974483
atan(-1)=-0.7853981633974483
```

**CEIL**

命令格式如下：

```
Bigint ceil(Double value)
```

```
Bigint ceil(Decimal value)
```

命令说明如下：

向上取整，函数返回不小于输入值value的最小整数。

参数说明：

**value**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换到Double类型后参与运算，其他类型抛异常。

返回值：

返回Bigint类型。任意一个参数输入为null，则返回null。

示例如下：

```
ceil(1.1)=2
ceil(-1.1)=-1
```

**CONV**

命令格式如下：

```
String conv(String input, Bigint from_base, Bigint to_base)
```

命令说明如下：

该函数为进制转换函数。

参数说明：

- **input**：以String表示的要转换的整数值，接受Bigint、Double的隐式转换。
- **from_base**、**to_base**：以十进制表示的进制的值，可接受的的值为2、8、10和16。接受String及Double的隐式转换。

返回值：

返回String类型。任意一个参数输入为null，返回null。转换过程以64位精度工作，溢出时报异常。输入如果是负值，即以（-）开头，报异常。如果输入的是小数，则会转为整数值后进行进制转换，小数部分会被舍弃。

示例如下：

```
conv('1100', 2, 10)='12'
conv('1100', 2, 16)='c'
conv('ab', 16, 10)='171'
```

```
conv('ab', 16, 16)='ab'
```

### COS

命令格式如下：

```
Double cos(Double number)
Decimal cos(Decimal number)
```

命令说明如下：

该函数用于计算number的余弦函数，输入为弧度值。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换到Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

示例如下：

```
cos(3.1415926/2)=2.6794896585028633e-8
cos(3.1415926)=-0.9999999999999986
```

### COSH

命令格式如下：

```
Double cosh(Double number)
Decimal cosh(Decimal number)
```

命令说明如下：

该函数用于计算number的双曲余弦函数。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后，参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，返回null。

### COT

命令格式如下：

```
Double cot(Double number)
Decimal cot(Decimal number)
```

命令说明如下：

该函数用于计算number的余切函数，输入为弧度值。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

### EXP

命令格式如下：

```
Double exp(Double number)
Decimal exp(Decimal number)
```

命令说明如下：

该函数用于计算number的指数函数。

返回值：

返回number的指数值。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

### FLOOR

命令格式如下：

```
Bigint floor(Double number)
```

```
Bigint floor(Decimal number)
```

命令说明如下：

向下取整，函数返回不大于number的最大整数值。

参数说明：

**number**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Bigint类型。若number为null，则返回null。

示例如下：

```
floor(1.2)=1
floor(1.9)=1
floor(0.1)=0
floor(-1.2)=-2
floor(-0.1)=-1
floor(0.0)=0
floor(-0.0)=0
```

**LN**

命令格式如下：

```
Double ln(Double number)
Decimal ln(Decimal number)
```

命令说明如下：

该函数用于返回number的自然对数。

参数说明：

**number**：Double类型或Decimal类型。

- 若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。
- 若number为null，则返回null。若number为负数或零，则执行报错。

返回值：

返回Double类型或Decimal类型。

### LOG

命令格式如下：

```
Double log(Double base, Double x)
Decimal log(Decimal base, Decimal x)
```

命令说明如下：

该函数用于返回以base为底的x的对数。

参数说明：

- **base**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **x**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型的对数值。

- 若base和x中存在null，则返回null。
- 若base和x中某一个值为负数或 0，会引发异常。
- 若base为1（会引发一个除零行为），也会引发异常。

### POW

命令格式如下：

```
Double pow(Double x, Double y)
Decimal pow(Decimal x, Decimal y)
```

命令说明如下：

该函数用于返回x的y次方，即x^y。

参数说明：

- **X**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **Y**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若x或y为null，则返回null。

**RAND**

命令格式如下：

```
Double rand(Bigint seed)
```

命令说明如下：

该函数以seed为种子，返回Double类型的随机数，返回值区间是的0～1。

参数说明：

**seed**：可选参数，Bigint类型，随机数种子，决定随机数序列的起始值。

返回值：

返回Double类型。

示例如下：

```
select rand() from dual;
select rand(1) from dual;
```

**ROUND**

命令格式如下：

```
Double round(Double number, [Bigint Decimal_places])
Decimal round(Decimal number, [Bigint Decimal_places])
```

命令说明如下：

该函数四舍五入到指定小数点位置。

参数说明：

- **number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **Decimal_place**：Bigint类型常量，四舍五入计算到小数点后的位置，其他类型参数会引发异常。如果省略表示四舍五入到个位数，默认值为0。

返回值：

返回Double类型或Decimal类型。若**number**或**Decimal_places**为null，则返回null。

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。

示例如下：

```
round(125.315)=125.0
round(125.315, 0)=125.0
round(125.315, 1)=125.3
round(125.315, 2)=125.32
round(125.315, 3)=125.315
round(-125.315, 2)=-125.32
round(123.345, -2)=100.0
round(null)=null
round(123.345, 4)=123.345
round(123.345, -4)=0.0
```

**SIN**

命令格式如下所示：

```
Double sin(Double number)
Decimal sin(Decimal number)
```

命令说明如下：

该函数用于计算number的正弦函数，输入为弧度值。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

**SINH**

命令格式如下：

```
Double sinh(Double number)
Decimal sinh(Decimal number)
```

命令说明如下：

该函数用于计算number的双曲正弦函数。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

**SQRT**

命令格式如下：

```
Double sqrt(Double number)
Decimal sqrt(Decimal number)
```

命令说明如下：

该函数用于计算number的平方根。

参数说明：

**number**：Double类型或Decimal类型，必须大于0，小于0时引发异常。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

**TAN**

命令说明如下：

```
Double tan(Double number)
Decimal tan(Decimal number)
```

命令说明如下：

该函数用于计算number的正切函数，输入为弧度值。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

### TANH

命令格式如下：

```
Double tanh(Double number)
Decimal tanh(Decimal number)
```

命令说明如下：

该函数用于计算number的双曲正切函数。

参数说明：

**number**：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

### TRUNC

命令格式如下：

```
Double trunc(Double number[, Bigint Decimal_places])
Decimal trunc(Decimal number[, Bigint Decimal_places])
```

命令说明如下：

该函数用于将输入值number截取到指定小数点位置。

参数说明：

- **number**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **Decimal_places**：Bigint类型常量，要截取到的小数点位置，其他类型参数会隐式转为Bigint，省略此参数时默认到截取到个位数。

返回值：

返回值类型为Double或Decimal类型。若**number**或**Decimal_places**为null，则返回null。

说明：

- 返回Double类型时，返回的结果的显示可能不符合预期，如示例`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。
- **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。

示例如下：

```
trunc(125.815)=125.0
trunc(125.815,0)=125.0
trunc(125.815,1)=125.80000000000001
trunc(125.815,2)=125.81
trunc(125.815,3)=125.815
trunc(-125.815,2)=-125.81
trunc(125.815,-1)=120.0
trunc(125.815,-2)=100.0
trunc(125.815,-3)=0.0
trunc(123.345,4)=123.345
trunc(123.345,-4)=0.0
```

### MaxCompute2.0扩展函数

升级到MaxCompute2.0后，产品扩展部分数学函数，新函数若用到新数据类型时，在使用新函数的SQL前，需要加一个set语句：

```
set odps.sql.type.system.odps2=true;
```

下文将为您详细介绍新扩展的函数。

### LOG2

命令格式如下：

```
Double log2(Double number)
Double log2(Decimal number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于以2为底，返回number的对数。

参数说明：

**number**：Double或Decimal类型。

返回值：

返回Double类型。若输入为0或null，则返回null。

示例如下：

```
log2(null)=null
log2(0)=null
log2(8)=3.0
```

### LOG10

命令格式如下：

```
Double log10(Double number)
Double log10(Decimal number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于以10为底，返回number的对数。

参数说明：

**number**：Double或Decimal类型。

返回值：

返回Double类型。若输入为0或null，则返回null。

示例如下：

```
log10(null)=null
log10(0)=null
log10(8)=0.9030899869919435
```

**BIN**

命令格式如下：

```
String bin(Bigint number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于返回number的二进制代码表示。

参数说明：

**number**：Bigint类型。

返回值：

返回String类型。若输入为0，返回0，输入为null，则返回null。

示例如下：

```
bin(0)='0'
bin(null)='null'
```

```
bin(12)='1100'
```

### HEX

命令格式如下：

```
String hex(Bigint number)
String hex(String number)
String hex(BINARY number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于将整数或字符转换为十六进制格式。

参数说明：

**number**：如果number是Bigint类型，那么返回number的十六进制表示。如果变量是String类型，则返回该字符串的十六进制表示。

返回值：

返回String类型。若输入为0，返回0，输入为null，则返回异常。

示例如下：

```
hex(0)=0
hex('abc')='616263'
hex(17)='11'
hex('17')='3137'
hex(null)异常返回失败
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

**UNHEX**

命令格式如下：

```
BINARY unhex(String number)
```

说明：
目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于返回十六进制字符串所代表的字符串。

参数说明：

**number**：为十六进制字符串。

返回值：

返回Binary类型，若输入0，返回失败，若输入为null，则返回null。

示例如下：

```
unhex('616263')='abc'
unhex(616263)='abc'
```

**RADIANS**

命令格式如下：

```
Double radians(Double number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于将角度转换为弧度。

参数说明：

**number**：Double类型数据。

返回值：

返回Double类型，若输入为null，返回null。

示例如下：

```
radians(90)=1.5707963267948966
radians(0)=0.0
```

```
radians(null)=null
```

### DEGREES

命令格式如下：

```
Double degrees(Double number)
Double degrees(Decimal number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于将弧度转换为角度。

参数说明：

**number**：Double或Decimal类型数据。

返回值：

返回Double类型，若输入null，则返回null。

示例如下：

```
degrees(1.5707963267948966)=90.0
degrees(0)=0.0
degrees(null)=null
```

### SIGN

命令格式如下：

```
Double sign(Double number)
```

```
Double sign(Decimal number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于取输入数据的符号，1.0表示正，-1.0表示负，否则0.0。

参数说明：

**number**：Double或Decimal类型数据。

返回值：

返回Double类型，若输入0，则返回0.0，输入为null，则返回null。

示例如下：

```
sign(-2.5)=-1.0
sign(2.5)=1.0
sign(0)=0.0
sign(null)=null
```

**E**

命令格式如下：

```
Double e()
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于返回e的值。

返回值：

返回Double类型。

示例如下：

```
e()=2.718281828459045
```

**PI**

命令格式如下：

```
Double pi()
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于返回π的值。

返回值：

返回Double类型。

示例如下：

```
pi()=3.141592653589793
```

**FACTORIAL**

命令格式如下：

```
Bigint factorial(Int number)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于返回number的阶乘。

参数说明：

**number**：Int类型数据，且在[0..20]之间。

返回值：

返回Bigint类型，输入为0，则返回1，输入为null或[0..20]之外的值，返回null。

示例如下：

```
factorial(5)=120 --5!=5*4*3*2*1=120
```

**CBRT**

命令格式如下：

```
Double cbrt(Double number)
```

 说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于计算number的立方根。

参数说明：

**number**：Double类型数据。

返回值：

返回Double类型，输入为null，返回null。

示例如下：

```
cbrt(8)=2
cbrt(null)=null
```

**SHIFTLEFT**

命令格式如下：

```
Int shiftleft(Tinyint|Smallint|Int number1, Int number2)
```

```
Bigint shiftleft(Bigint number1, Int number2)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于按位左移（**<<**）。

参数说明：

- **number1**：Tinyint|Smallint|Int|Bigint整型数据。
- **number2**：Int整型数据。

返回值：

返回Int或Bingint类型。

示例如下：

```
shiftleft(1,2)=4   --1的二进制左移2位（1<<2,0001左移两位是0100）
shiftleft(4,3)=32   --4的二进制左移3位（4<<3,0100左移3位是100000）
```

**SHIFTRIGHT**

命令格式如下：

```
Int shiftright(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftright(Bigint number1, Int number2)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于按位右移（**>>**）。

参数说明：

- **number1**：Tinyint|Smallint|Int|Bigint整型数据。
- **number2**：Int整型数据。

返回值：

返回Int或Bigint类型。

示例如下：

```
shiftright(4,2)=1 -- 4的二进制右移2位（4>>2,0100右移两位是0001）
shiftright(32,3)=4 -- 32的二进制右移3位（32>>3,100000右移3位是0100）
```

**SHIFTRIGHTUNSIGNED**

命令格式如下：

```
Int shiftrightunsigned(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftrightunsigned(Bigint number1, Int number2)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

命令说明如下：

该函数用于无符号按位右移（**>>>**）。

参数说明：

- **number1**：Tinyint|Smallint|Int|Bigint整型数据。
- **number2**：Int整型数据。

返回值：

返回Int或Bigint类型。

示例如下：

```
shiftrightunsigned(8,2)=2  --8的二进制无符号右移2位(8>>>2,1000右移两位是
0010)
shiftrightunsigned(-14,2)=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)
```

## 1.10.3 窗口函数

MaxCompute SQL中可以使用窗口函数进行灵活的分析处理工作，窗口函数只能出现在select子句中，窗口函数中不要嵌套使用窗口函数和聚合函数，窗口函数不可以和同级别的聚合函数一起使用。

目前在一个MaxCompute SQL语句中，最多可以使用5个窗口函数。

窗口函数的语法声明如下：

```
window_func() over (partition by [col1,col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] windowing_clause)
```

- **partition by**部分用来指定开窗的列。分区列的值相同的行被视为在同一个窗口内。现阶段，同一窗口内最多包含1亿行数据（建议不超过500万行），否则运行时报错。
- **order by**用来指定数据在一个窗口内如何排序。
- **windowing_clause**部分可以用rows指定开窗方式，有以下两种方式：

- rows between x preceding|following and y preceding|following表示窗口范围是从前或后x行到前或后y行。
- rows x preceding|following窗口范围是从前或后第x行到当前行。

说明：

- x，y必须为大于等于0的整数常量，限定范围0 ~ 10000，值为0时表示当前行。必须指定order by才可以用rows方式指定窗口范围。
- 并非所有的窗口函数都可以用rows指定开窗方式，支持这种用法的窗口函数有AVG、COUNT、MAX、MIN、STDDEV和SUM。

当窗口函数多次调用使用同样的窗口，反复写OVER语句会显得冗余，由此MaxCompute支持WINDOW语句预定义窗口，如下所示：

```
SELECT ROW_NUMBER() OVER w, RANK() OVER w FROM t  WINDOW w as (
partition by c1 order by c2);
```

说明：

WINDOW语句定义出现在where/groupby/having语句之后，在order by/cluster by/distribute by/sort by/limit语句之前。

**COUNT**

命令格式如下：

```
Bigint count([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算计数值。

参数说明：

- **expr**：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 by [col1, col2…]**：指定开窗口的列。
- **order by col1 [asc|desc], col2[asc|desc]**：不指定order by时，返回当前窗口内expr的计数值，指定order by时返回结果以指定的顺序排序，并且值为当前窗口内从开始行到当前行的累计计数值。

返回值：

返回Bigint类型。

**说明：**

当指定distinct关键字时，不能写order by。

示例如下：

假设存在表test_src，表中存在Bigint类型的列user_id。

```
select user_id,count(user_id) over (partition by user_id) as count
from test_src;
    +---------+------------+
    | user_id |  count     |
    +---------+------------+
    | 1       | 3          |
    | 1       | 3          |
    | 1       | 3          |
    | 2       | 1          |
    | 3       | 1          |
    +---------+------------+
  -- 不指定order by时，返回当前窗口内user_id的计数值
select user_id,count(user_id) over (partition by user_id order by
user_id) as count
from test_src;
    +---------+------------+
    | user_id | count      |
    +---------+------------+
    | 1       | 1          |      -- 窗口起始
    | 1       | 2          |      -- 到当前行共计两条记录，返回2
    | 1       | 3          |
    | 2       | 1          |
    | 3       | 1          |
    +---------+------------+
  -- 指定order by时，返回当前窗口内从开始行到当前行的累计计数值。
```

## AVG

命令格式如下：

```
avg([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算平均值。

参数说明：

- **distinct**：当指定distinct关键字时，表示取唯一值的平均值。

- **expr**：Double类型，Decimal类型。
  - 当输入值为String、Bigint类型时，会隐式转换到Double类型后参与运算，其它类型抛异常。
  - 当输入值为null时，该行不参与计算。
  - Boolean类型不允许参与计算。
- **partition by [col1, col2...]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：不指定order by时，返回当前窗口内所有值的平均值，指定order by时，返回结果以指定的方式排序，并且返回窗口内从开始行到当前行的累计平均值。

返回值：

返回Double类型。

说明：

指明distinct关键字时，不能写order by。

### MAX

命令格式如下：

```
max([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算最大值。

参数说明：

- **expr**：除Boolean外的任意类型，当值为null时，该行不参与计算。当指定distinct关键字时，表示取唯一值的最大值（指定该参数与否对结果没有影响）。
- **partition by [col1, col2…]**：指定开窗口的列。
- **order by [col1[asc|desc], col2[asc|desc**：不指定order by时，返回当前窗口内的最大值。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的最大值。

返回值：

返回值的类型同expr类型。

**说明：**

指明distinct关键字时，不能写order by。

**MIN**

命令格式如下：

```
min([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算最小值。

参数说明：

- **expr**：除Boolean外的任意类型，当值为null时，该行不参与计算。当指定distinct关键字时，表示取唯一值的最小值（指定该参数与否对结果没有影响）。
- **partition by [col1, col2…]**：指定开窗口的列。
- **order by [col1[asc|desc], col2[asc|desc**：不指定order by时，返回当前窗口内的最小值。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的最小值。

返回值：

返回值类型同expr类型。

**说明：**

指明distinct关键字时，不能写order by。

**MEDIAN**

命令格式如下：

```
Double median(Double number1,number2...) over(partition by [col1, col2
…])
Decimal median(Decimal number1,number2...) over(partition by [col1,
col2…])
```

命令说明如下：

该函数用于计算中位数最小值。

参数说明：

- **number1,number1…**：Double类型或Decimal类型的1到255个数字。
  - 当输入值为String类型或Bigint类型，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 当输入值为null时，返回null。
  - 如果传入的参数是Double类型，会默认转成Double的Array。
- **partition by [col1, col2…]**：指定开窗口的列。

返回值：

返回值类型同Double类型。

**STDDEV**

命令格式如下：

```
Double stddev([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
Decimal stddev([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算总体标准差。

参数说明：

- **expr**：Double类型或Decimal类型。
  - 当输入值为String类型或Bigint类型时，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 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 by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：不指定order by时，返回当前窗口内的总体标准差。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的总体标准差。

返回值：

输入值为Decimal类型时，返回Decimal类型，否则返回Double类型。

示例如下：

```
select window, seq, stddev_pop('1\01') over (partition by window order
 by seq) from dual;
```

 说明：

- 当指定distinct关键字时，不能写order by。
- **stddev**还有一个别名函数**stddev_pop**，用法和**stddev**一样。

### STDDEV_SAMP

命令格式如下：

```
Double stddev_samp([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
Decimal stddev_samp([distinct] expr) over((partition by [col1,col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算样本标准差。

参数说明：

- **expr**：Double类型或Decimal类型。
  - 当输入值为String类型或Bigint类型时，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 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 by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：不指定order by时，返回当前窗口内的样本标准差。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的样本标准差。

返回值：

输入值为Decimal类型时，返回Decimal类型，否则返回Double类型。

 说明：

指明distinct关键字时，不能写order by。

### SUM

命令格式如下：

```
sum([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算汇总值。

参数说明：

- **expr**：Double类型、Decimal类型或Bigint类型。
  - 当输入值为String类型时，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 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 by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：不指定order by时，返回当前窗口内expr的汇总值。指定order by时，返回结果以指定的方式排序，并且返回当前窗口从首行至当前行的累计汇总值。

返回值：

- 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。

**说明：**

指明distinct关键字时，不能写order by。

### DENSE_RANK

命令说明如下：

```
Bigint dense_rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于计算连续排名。col2相同的行数据获得的排名相同。

参数说明：

- **partition by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：指定排名依据的值。

返回值：

返回Bigint类型。

示例如下：

假设表emp中的数据如下所示：

```
| 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
```

现在需要将所有职工根据部门分组，每个组内根据SAL做降序排序，获得职工自己组内的序号。

```
SELECT deptno, ename, sal, DENSE_RANK() OVER (PARTITION BY deptno
ORDER BY sal DESC) AS nums--deptno(部门)作为开窗列，sal（薪水）作为结果返回
时需要排序的值。
    FROM emp;
--执行结果如下：
+------------+-------+------------+------------+
| deptno     | ename | sal        | nums       |
+------------+-------+------------+------------+
| 10         | JACCKA | 5000.0     | 1          |
| 10         | KING  | 5000.0     | 1          |
| 10         | CLARK | 2450.0     | 2          |
| 10         | WELAN | 2450.0     | 2          |
| 10         | TEBAGE | 1300.0     | 3          |
| 10         | MILLER | 1300.0     | 3          |
| 20         | SCOTT | 3000.0     | 1          |
| 20         | FORD  | 3000.0     | 1          |
| 20         | JONES | 2975.0     | 2          |
| 20         | ADAMS | 1100.0     | 3          |
| 20         | SMITH | 800.0      | 4          |
| 30         | BLAKE | 2850.0     | 1          |
| 30         | ALLEN | 1600.0     | 2          |
| 30         | TURNER | 1500.0     | 3          |
```

```
| 30          | MARTIN | 1250.0     | 4          |
| 30          | WARD   | 1250.0     | 4          |
| 30          | JAMES | 950.0      | 5          |
+------------+-------+------------+------------+
```

**RANK**

命令格式如下：

```
Bigint rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于计算排名。col2相同的行数据获得排名顺序下降。

参数说明：

- **partition by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：指定排名依据的值。

返回值：

返回Bigint类型。

示例如下：

假设表emp中的数据如下所示：

```
| 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
```

现在需要将所有职工根据部门分组，每个组内根据SAL做降序排序，获得职工自己组内的序号。

```
SELECT deptno,ename,sal, RANK() OVER (PARTITION BY deptno ORDER BY sal
 DESC) AS nums--deptno(部门)作为开窗列，sal（薪水）作为结果返回时需要排序的值。
FROM emp;
```

```
--执行结果如下：
+------------+-------+------------+------------+
| deptno     | ename | sal        | nums       |
+------------+-------+------------+------------+
| 10         | JACCKA | 5000.0     | 1          |
| 10         | KING  | 5000.0     | 1          |
| 10         | CLARK | 2450.0     | 3          |
| 10         | WELAN | 2450.0     | 3          |
| 10         | TEBAGE | 1300.0     | 5          |
| 10         | MILLER | 1300.0     | 5          |
| 20         | SCOTT | 3000.0     | 1          |
| 20         | FORD  | 3000.0     | 1          |
| 20         | JONES | 2975.0     | 3          |
| 20         | ADAMS | 1100.0     | 4          |
| 20         | SMITH | 800.0      | 5          |
| 30         | BLAKE | 2850.0     | 1          |
| 30         | ALLEN | 1600.0     | 2          |
| 30         | TURNER | 1500.0     | 3          |
| 30         | MARTIN | 1250.0     | 4          |
| 30         | WARD  | 1250.0     | 4          |
| 30         | JAMES | 950.0      | 6          |
+------------+-------+------------+------------+
```

### LAG

命令格式如下：

```
lag(expr , Bigint offset, default) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]])
```

命令说明如下：

按偏移量取当前行之前第几行的值，如当前行号为rn，则取行号为rn-offset的值。

参数说明：

- **expr**：任意类型。
- **offset**：Bigint类型常量。输入值为String、Double到Bigint的隐式转换，offset>0。
- **default**：当offset指定的范围越界时的缺省值，常量，默认值为null。
- **partition by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：指定返回结果的排序方式。

返回值：

返回值类型同expr类型。

示例如下：

```
select seq, lag(seq+100, 1) over (partition by window order by seq) as
 r from sliding_window;
  +------------+------------+
  | seq        | r          |
```

```
+------------+------------+
| 0          | NULL       |
| 1          | 100        |
| 2          | 101        |
| 3          | 102        |
| 4          | 103        |
| 5          | 104        |
| 6          | 105        |
| 7          | 106        |
| 8          | 107        |
| 9          | 108        |
+------------+------------+
```

**LEAD**

命令格式如下：

```
lead(expr , Bigint offset, default) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]])
```

命令说明如下：

按偏移量取当前行之后第几行的值，如当前行号为rn，则取行号为rn+offset的值。

参数说明：

- **expr**：任意类型。
- **offset**：可选，Bigint类型常量。输入值为String、Decimal、Double到Bigint的隐式转换，offset>0。
- **default**：可选，当offset指定的范围越界时的缺省值，常量。
- **partition by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：指定返回结果的排序方式。

返回值：

返回值类型同expr类型。

示例如下：

```
select c_Double_a,c_String_b,c_int_a,lead(c_int_a,1) over(partition by
 c_Double_a order by c_String_b) from dual;
select c_String_a,c_time_b,c_Double_a,lead(c_Double_a,1) over(
partition by c_String_a order by c_time_b) from dual;
```

```
select c_String_in_fact_num,c_String_a,c_int_a,lead(c_int_a) over(
partition by c_String_in_fact_num order by c_String_a) from dual;
```

### PERCENT_RANK

命令格式如下：

```
percent_rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于计算一组数据中某行的相对排名。

参数说明：

- **partition by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：指定排名依据的值。

返回值：

返回Double类型，值域为[0, 1]，相对排名的计算方式为`(rank-1)/(number of rows -1)`。

**说明：**

目前限制单个窗口内的行数不超过10,000,000条。

### ROW_NUMBER

命令格式如下：

```
row_number() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于计算行号，从1开始。

参数说明：

- **partition by [col1, col2..]**：指定开窗口的列。
- **order by col1[asc|desc], col2[asc|desc]**：指定结果返回时的排序的值。

返回值：

返回Bigint类型。

示例如下：

假设表emp中的数据如下所示：

```
| 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
```

现在需要将所有职工根据部门分组，每个组内根据SAL做降序排序，获得职工自己组内的序号。

```
SELECT deptno,ename,sal,ROW_NUMBER() OVER (PARTITION BY deptno ORDER
BY sal DESC) AS nums--deptno(部门)作为开窗列，sal(薪水)作为结果返回时需要排
序的值。
    FROM emp;
--执行结果如下：
+------------+-------+------------+------------+
| deptno     | ename | sal        | nums       |
+------------+-------+------------+------------+
| 10         | JACCKA | 5000.0     | 1          |
| 10         | KING  | 5000.0     | 2          |
| 10         | CLARK | 2450.0     | 3          |
| 10         | WELAN | 2450.0     | 4          |
| 10         | TEBAGE | 1300.0     | 5          |
| 10         | MILLER | 1300.0     | 6          |
| 20         | SCOTT | 3000.0     | 1          |
| 20         | FORD  | 3000.0     | 2          |
| 20         | JONES | 2975.0     | 3          |
| 20         | ADAMS | 1100.0     | 4          |
| 20         | SMITH | 800.0      | 5          |
| 30         | BLAKE | 2850.0     | 1          |
| 30         | ALLEN | 1600.0     | 2          |
| 30         | TURNER | 1500.0     | 3          |
| 30         | MARTIN | 1250.0     | 4          |
| 30         | WARD  | 1250.0     | 5          |
| 30         | JAMES | 950.0      | 6          |
+------------+-------+------------+------------+
```

### CLUSTER_SAMPLE

命令格式如下：

```
boolean cluster_sample([Bigint x, Bigint y])
```

```
over(partition by [col1, col2..])
```

命令说明如下：

该函数用于分组抽样。

参数说明：

- **x**：Bigint类型常量，x>=1。若指定参数y，x表示将一个窗口分为x份。否则，x表示在一个窗口中抽取x行记录（即有x行返回值为true）。x为null时，返回值为null。
- **y**：Bigint类型常量，y>=1，y<=x。表示从一个窗口分的x份中抽取y份记录（即y份记录返回值为true）。y为null时，返回值为null。
- **partition by [col1, col2]**：指定开窗口的列。

返回值：

返回Boolean类型。

示例如下：

假设表test_tbl中有key，value两列，key为分组字段，值有groupa，groupb两组，value为值，如下所示：

```
+------------+--------------------+
| key        | value              |
+------------+--------------------+
| groupa     | -1.34764165478145  |
| groupa     | 0.740212609046718  |
| groupa     | 0.167537127858695  |
| groupa     | 0.630314566185241  |
| groupa     | 0.0112401388646925 |
| groupa     | 0.199165745875297  |
| groupa     | -0.320543343353587 |
| groupa     | -0.273930924365012 |
| groupa     | 0.386177958942063  |
| groupa     | -1.09209976687047  |
| groupb     | -1.10847690938643  |
| groupb     | -0.725703978381499 |
| groupb     | 1.05064697475759   |
| groupb     | 0.135751224393789  |
| groupb     | 2.13313102040396   |
| groupb     | -1.11828960785008  |
| groupb     | -0.849235511508911 |
| groupb     | 1.27913806620453   |
| groupb     | -0.330817716670401 |
| groupb     | -0.300156896191195 |
| groupb     | 2.4704244205196    |
| groupb     | -1.28051882084434  |
```

```
    +------------+--------------------+
```

想要从每组中抽取约10%的值，可以用以下MaxCompute SQL完成。

```
select key, value
    from (
        select key, value, cluster_sample(10, 1) over(partition by key
) as flag
        from tbl
        ) sub
    where flag = true;
+-----+------------+
| key | value      |
+-----+------------+
| groupa | 0.167537127858695 |
| groupb | 0.135751224393789 |
+-----+------------+
```

## 1.10.4 聚合函数

聚合函数，其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SQL中的group by语句联用。

### COUNT

命令格式如下：

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

命令说明如下：

该函数用于计算记录数。

参数说明：

- distinct|all：指明在计数时是否去除重复记录，默认是all，即计算全部记录，如果指定distinct，则可以只计算唯一值数量。
- value：可以为任意类型，当value值为NULL时，该行不参与计算，value可以为*，当count(*)时，返回所有行数。

返回值：

返回Bigint类型。

示例如下：

```
-- 如表tbla有列col1类型为 Bigint
+------+
| COL1 |
+------+
| 1 |
```

```
+------+
| 2 |
+------+
| NULL |
+------+
select count(*) from tbla; -- 值为3,
select count(col1) from tbla; -- 值为2
```

聚合函数可以和group by一同使用，例如：假设存在表test_src，存在如下两列：key string类型，value double类型。

```
-- test_src的数据为
+-----+-------+
| key | value |
+-----+-------+
| a | 2.0 |
+-----+-------+
| a | 4.0 |
+-----+-------+
| b | 1.0 |
+-----+-------+
| b | 3.0 |
+-----+-------+
-- 此时执行如下语句，结果为：
select key, count(value) as count from test_src group by key;
+-----+-------+
| key | count |
+-----+-------+
| a | 2 |
+-----+-------+
| b | 2 |
+-----+-------+
-- 聚合函数将对相同key值得value值做聚合计算。下面介绍的其他聚合函数使用方法均与此例相同，不一一举例。
```

### AVG

命令格式如下：

```
double avg(double value)
decimal avg(decimal value)
```

命令说明如下：

该函数用于计算平均值。

参数说明：

value：Double类型或Decimal 类型，若输入为String或Bigint，会隐式转换到Double类型后参与运算，其它类型抛异常。当value值为NULL 时，该行不参与计算。Boolean类型不允许参与计算。

返回值：

若输入Decimal类型，返回Decimal类型，其他合法输入类型返回Double类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+-------+
| value |
+-------+
| 1 |
| 2 |
| NULL |
+-------+
-- 则对该列计算avg结果为(1+2)/2=1.5
select avg(value) as avg from tbla;
+------+
| avg |
+------+
| 1.5 |
+------+
```

**MAX**

命令格式如下：

```
max(value)
```

命令说明如下：

该函数用于计算最大值。

参数说明：

value：可以为任意类型，当列中的值为NULL时，该行不参与计算。Boolean类型不允许参与运算。

返回值：

返回值的类型与value类型相同。

示例如下：

```
-- 如表tbla有一列col1，类型为 Bigint
+------+
| col1 |
+------+
| 1 |
+------+
| 2 |
+------+
| NULL |
+------+
```

```
select max(value) from tbla; -- 返回值为2
```

**MIN**

命令格式如下：

```
MIN(value)
```

命令说明如下：

该函数用于计算最小值。

参数说明：

value：可以为任意类型，当列中的值为NULL时，该行不参与计算。Boolean类型不允许参与计算。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| NULL |
+------+
select min(value) from tbla; -- 返回值为1
```

**MEDIAN**

命令格式如下：

```
double median(double number)
decimal median(decimal number)
```

命令说明如下：

该函数用于计算中位数。

参数说明：

number：Double类型或者Decimal类型。若输入为String类型或Bigint类型，会隐式转换到Double类型后参与运算，其他类型报错。

返回值：

返回Double类型或者Decimal类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
select MEDIAN(value) from tbla; -- 返回值为 3.0
```

**STDDEV**

命令格式如下：

```
double stddev(double number)
decimal stddev(decimal number)
```

命令说明如下：

该函数用于计算总体标准差。

参数说明：

number：Double类型或者Decimal类型。若输入为String类型或Bigint类型，会隐式转换到Double类型后参与运算，其他类型时报错。

返回值：

返回Double类型或者Decimal类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
```

```
select STDDEV(value) from tbla; -- 返回值为 1.4142135623730951
```

### STDDEV_SAMP

命令格式如下：

```
double stddev_samp(double number)
decimal stddev_samp(decimal number)
```

命令说明如下：

该函数用于计算样本标准差。

参数说明：

number：Double类型或者Decimal类型。若输入为string类型或Bigint类型会隐式转换到double类型后参与运算，其他类型时报错。

返回值：

返回Double类型或者Decimal类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
select STDDEV_SAMP(value) from tbla; -- 返回值为 1.5811388300841898
```

### SUM

命令格式如下：

```
sum(value)
```

命令说明如下：

该函数用于计算汇总值。

参数说明：

value：Double、Decimal或Bigint类型，若输入为String会隐式转换到Double类型后参与运算，当列中的值为NULL时，该行不参与计算。Boolean类型不允许参与计算。

返回值：

输入为Bigint时，返回Bigint，输入为Double或String时，返回Double类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| NULL |
+------+
select sum(value) from tbla; -- 返回值为3
```

**WM_CONCAT**

命令格式如下：

```
string wm_concat(string separator, string str)
```

命令说明如下：

该函数用指定的spearator做分隔符，链接str中的值。

参数说明：

- separator：String类型常量，分隔符。其他类型或非常量将引发异常。
- str：String类型，若输入为Bigint，Double或者Datetime类型，会隐式转换为String后参与运算，其它类型报异常。

返回值：

返回String类型。

 说明：

语句`select wm_concat(',', name) from test_src;`中，若test_src为空集合，此条语句返回NULL值。

### COLLECT_LIST

命令格式如下：

```
ARRAY collect_list(col)
```

命令说明如下：

该函数在给定group内，将col指定的表达式聚合为一个数组。

参数说明：

col：表的某列，可为任意数据类型。

返回值：

返回ARRAY类型。

 说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

### COLLECT_SET

命令格式如下：

```
ARRAY collect_set(col)
```

命令说明如下：

该函数在给定group内，将col指定的表达式聚合为一个无重复元素的集合数组。

参数说明：

col：表的某列，可为任意数据类型。

返回值：

返回ARRAY类型。

 说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

## 1.10.5 字符串函数

### CHAR_MATCHCOUNT

命令格式如下：

```
bigint char_matchcount(string str1, string str2)
```

命令说明如下：

该函数用于计算str1中有多少个字符出现在str2中。

参数说明：

- str1，str2：String类型，必须为有效的UTF-8字符串，如果对比中发现有无效字符则函数返回负值。
- bigint：返回值为bigint类型。任一输入为NULL返回NULL。

示例如下：

```
char_matchcount('abd','aabc') = 2
```

```
-- str1中得两个字符串'a','b'在str2中出现过
```

**CHR**

命令格式如下：

```
string chr(bigint ascii)
```

命令说明如下：

该函数用于将给定ASCII码ascii转换成字符。

参数说明：

- ascii：Bigint类型ASCII值，若输入为String类型或Double类型或Decimal类型会隐式转换到Bigint类型后参与运算，其它类型抛异常。
- String：返回值为String类型。参数范围是0~255，超过此范围会引发异常。输入值为NULL返回NULL。

**CONCAT**

命令格式如下：

```
string concat(string a, string b...)
```

命令说明如下：

该函数的返回值是将参数中的所有字符串连接在一起的结果。

参数说明：

- a，b等为String类型，若输入为Bigint，Double，Decimal或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- String：返回值为String类型。如果没有参数或者某个参数为NULL，结果均返回NULL。

示例如下：

```
concat('ab','c') = 'abc'
concat() = NULL
```

```
concat('a', null, 'b') = NULL
```

### GET_JSON_OBJECT

命令格式如下：

```
STRING GET_JSON_OBJECT(STRING json,STRING path)
```

命令说明如下：

该函数用于在一个标准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，因此反复调用可能会造成性能和费用的浪费。利用GET_JSON_OBJECT结合UDTF，您可以轻松转换JSON格式日志数据，避免多次调用函数，详情参见利用*MaxCompute*内建函数及*UDTF*转换*JSON*格式日志数据。

参数说明：

- json： String类型，标准的json格式字符串。
- path：**String**类型，用于描述在**json**中的**path**，以**$**开头。关于新实现中json path的说明，请参见：*JsonPath*，$表示根节点，（.）表示child，[number]表示数组下标，对于数组，格式为key[sub1][sub2][sub3]……，[*]返回整个数组，*不支持转义。
- String：返回值为String类型。

 **说明：**

- 如果json为空或者非法的json格式，返回NULL。
- 如果path为空或者不合法（json中不存在）返回NULL。
- 如果json合法，path也存在则返回对应字符串。

示例一如下：

```
+----+
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"
}
```

通过以下查询，可以提取json对象中的信息：

```
odps> SELECT get_json_object(src_json.json, '$.owner') FROM src_json;
amy
```

```
odps> SELECT get_json_object(src_json.json, '$.store.fruit\[0]') FROM
src_json;
{"weight":8,"type":"apple"}
odps> SELECT get_json_object(src_json.json, '$.non_exist_key') FROM
src_json;
NULL
```

示例二如下：

```
get_json_object('{"array":[["aaaa",1111],["bbbb",2222],["cccc",3333
]]}','$.array[1][1]')= "2222"
get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["
h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[*]') = "["h0","h1","h2"]"
get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["
h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[1]') = "h1"
```

**INSTR**

命令格式如下：

```
bigint instr(string str1, string str2[, bigint start_position[, bigint
 nth_appearance]])
```

命令说明如下：

该函数用于计算子串str2在字符串str1中的位置。

参数说明：

- str1：String类型，搜索的字符串，若输入为Bigint，Double，Decimal或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- str2：String类型，要搜索的子串，若输入为Bigint，Double，Decimal或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- start_position：bigint类型，其它类型会抛异常，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。
- nth_appearance：bigint类型，大于0，表示子串在字符串中的第nth_appearance次匹配的位置，如果 nth_appearance为其它类型或小于等于0会抛异常。
- bigint：返回值为bigint类型。

 说明：

- 如果在str1中未找到str2，返回0。
- 任一输入参数为NULL返回NULL。
- 如果str2为空串时总是能匹配成功，instr(‘abc’, ‘’) 会返回 1。

示例如下：

```
instr('Tech on the net', 'e') = 2
instr('Tech on the net', 'e', 1, 1) = 2
instr('Tech on the net', 'e', 1, 2) = 11
instr('Tech on the net', 'e', 1, 3) = 14
```

### IS_ENCODING

命令格式如下：

```
boolean is_encoding(string str, string from_encoding, string
to_encoding)
```

命令说明如下：

用于判断输入字符串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。可用于判断输入是否为乱码，通常的用法是将from_encoding设为utf-8，to_encoding设为gbk。

参数说明：

- str：String类型，输入为NULL返回NULL。空字符串则可以被认为属于任何字符集。
- from_encoding，to_encoding：String类型，源及目标字符集。输入为NULL返回NULL。
- boolean：返回值为Boolean类型，如果str能够成功转换，则返回true，否则返回false。

示例如下：

```
is_encoding('测试', 'utf-8', 'gbk') = true
is_encoding('測試', 'utf-8', 'gbk') = true
-- gbk字库中有这两个繁体字
is_encoding('測試', 'utf-8', 'gb2312') = false
-- gb2312库中不包括这两个字
```

### KEYVALUE

命令格式如下：

```
KEYVALUE(STRING srcStr,STRING split1,STRING split2, STRING key)
KEYVALUE(STRING srcStr,STRING key) //split1 = ";"，split2 = ":"
```

命令说明如下：

将srcStr（源字符串）按split1分成key-value对，按split2将key-value对分开，返回key所对应的value。

参数说明：

- srcStr输入待拆分的字符串。

- key：String类型。源字符串按照split1和split2拆分后，根据该key值的指定，返回其对应的value。
- split1，split2：用来作为分隔符的字符串，按照指定的这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。

返回值：

- String 类型；
- Split1或split2为NULL时，返回NULL。
- srcStr，key为NULL或者没有匹配的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。

示例一：

```
keyvalue('0:1\;1:2', 1) = '2'
```

说明：

源字符串为“0:1\;1:2”，因为没有指定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拆分后变成：

```
0 1/
1 2
```

返回key为1所对应的value值2。

示例二：

```
keyvalue("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;
shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;","\;",":","tf") = "
21910" value:21910。
```

说明：

源字符串如下所示：

```
“\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf
:0\;market:shoes\;instPayAmount:0\;”
```

按照split1“\;”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split2":"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
```

key值为tf，返回其对应的value:21910。

**LENGTH**

命令格式如下：

```
bigint length(string str)
```

命令说明如下：

返回字符串str的长度。

参数说明：

- str：String类型，若输入为Bigint，Double，Decimal或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- bigint：返回值为Bigint类型。若str是NULL返回NULL。如果str非UTF-8编码格式，返回-1。

示例如下：

```
length('hi! 中国') = 6
```

**LENGTHB**

命令格式如下：

```
bigint lengthb(string str)
```

命令说明如下：

返回字符串str的以字节为单位的长度。

参数说明：

- str：String类型，若输入为Bigint，Double，Decimal或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- bigint：返回值为Bigint类型。若str是NULL返回NULL。

示例如下：

```
lengthb('hi! 中国') = 10
```

**MD5**

命令格式如下：

```
string md5(string value)
```

命令说明如下：

计算输入字符串value的md5值。

参数说明：

- value：String类型，如果输入类型是Bigint，Double，Decimal或者Datetime会隐式转换成String类型参与运算，其它类型报异常。输入为NULL，返回NULL。
- String：返回值为String类型。

### REGEXP_EXTRACT

命令格式如下：

```
string regexp_extract(string source, string pattern[, bigint
occurrence])
```

命令说明如下：

将字符串source按照pattern正则表达式的规则拆分，返回第occurrence个group的字符。

参数说明：

- source：String类型，待搜索的字符串。
- pattern：String类型常量，pattern为空串时抛异常，pattern中如果没有指定group，抛异常。
- occurrence：Bigint类型常量，必须 >=0，其它类型或小于0时抛异常，不指定时默认为1，表示返回第一个group。若occurrence = 0，返回满足整个pattern的子串。
- String：返回值为String类型，任一输入为NULL返回NULL。

示例如下：

```
regexp_extract('foothebar', 'foo(.*?)(bar)', 1) = the
regexp_extract('foothebar', 'foo(.*?)(bar)', 2) = bar
regexp_extract('foothebar', 'foo(.*?)(bar)', 0) = foothebar
regexp_extract('8d99d8', '8d(\\d+)d8') = 99
-- 如果是在MaxCompute客户端上提交正则计算的SQL，需要使用两个"\"作为转移字符
regexp_extract('foothebar', 'foothebar')
-- 异常返回，pattern中没有指定group
```

### REGEXP_INSTR

命令格式如下：

```
bigint regexp_instr(string source, string pattern[,
bigint start_position[, bigint nth_occurrence[, bigint return_option
]])
```

命令说明如下：

返回字符串source从start_position开始，和pattern第n次（nth_occurrence）匹配的子串的起始/结束位置。任一输入参数为NULL时返回NULL。

参数说明：

- source：String类型，待搜索的字符串。
- pattern：String类型常量，pattern为空串时抛异常。

- start_position：Bigint类型常量，搜索的开始位置。不指定时默认值为1，其它类型或小于等于0的值会抛异常。
- nth_occurrence：Bigint类型常量，不指定时默认值为1，表示搜索第一次出现的位置。小于等于0或者其它类型抛异常。
- return_option：Bigint类型常量，值为0或1，其它类型或不允许的值会抛异常。0表示返回匹配的开始位置，1表示返回匹配的结束位置。
- bigint：返回值为Bigint类型。return_option指定的类型返回匹配的子串在source中的开始或结束位置。

示例如下：

```
regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14
```

### REGEXP_REPLACE

命令格式如下：

```
string regexp_replace(string source, string pattern, string replace_st
ring[, bigint occurrence])
```

命令说明如下：

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。

参数说明：

- source：String类型，要替换的字符串。
- pattern：String类型常量，要匹配的模式，pattern为空串时抛异常。
- replace_string：String类型，将匹配的pattern替换成的字符串。
- occurrence：Bigint类型常量，必须大于等于0，表示将第几次匹配替换成replace_string，为0时表示替换掉所有的匹配子串。其它类型或小于0抛异常。可缺省，默认值为0。
- String：返回值为String类型，当引用不存在的组时，不进行替换。当输入source，pattern，occurrence 参数为NULL时返回NULL，若replace_string为NULL且pattern有匹配，返回NULL，replace_string为NULL 但pattern不匹配，则返回原串。

 说明：

当引用不存在的组时，行为未定义。

示例如下：

```
regexp_replace("123.456.7890", "([[:digit:]]{3})\\.([[:digit:]]{3})\\.
([[:digit:]]{4})",
"(\\1)\\2-\\3", 0) = "(123)456-7890"
regexp_replace("abcd", "(.)", "\\1 ", 0) = "a b c d "
regexp_replace("abcd", "(.)", "\\1 ", 1) = "a bcd"
regexp_replace("abcd", "(.)", "\\2", 1) = "abcd"
-- 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"
regexp_replace("abcd", "a", "\\1", 0) = "bcd"
-- 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_SUBSTR**

命令格式如下：

```
string regexp_substr(string source, string pattern[, bigint start_posi
tion[, bigint nth_occurrence]])
```

命令说明如下：

从start_position位置开始，source中第nth_occurrence次匹配指定模式pattern的子串。

参数说明：

- source：String类型，搜索的字符串。
- pattern：String类型常量，要匹配的模型，pattern为空串时抛异常。
- start_position：Bigint常量，必须大于0。其它类型或小于等于0时抛异常，不指定时默认为1，表示从 source的第一个字符开始匹配。不指定时默认为1，表示从source的第一个字符开始匹配。
- nth_occurrence：Bigint常量，必须大于0，其它类型或小于等于0时抛异常。不指定时默认为1，表示返回第一次匹配的子串。不指定时默认为1，表示返回第一次匹配的子串。
- String：返回值为String类型。任一输入参数为NULL返回NULL。没有匹配时返回NULL。

示例如下：

```
regexp_substr ("I love aliyun very much", "a[[:alpha:]]{5}") = "aliyun
"
regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum
:]]*', 1, 1) = " have"
```

```
regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum
:]]*', 1, 2) = " 2"
```

**REGEXP_COUNT**

命令格式如下：

```
bigint regexp_count(string source, string pattern[, bigint start_posi
tion])
```

命令说明如下：

计算source中从start_position开始，匹配指定模式pattern的子串的次数。

参数说明：

- source：String类型，搜索的字符串，其它类型报异常。
- pattern：String类型常量，要匹配的模型，pattern为空串时抛异常，其它类型报异常。
- start_position：Bigint类型常量，必须大于0。其它类型或小于等于0时抛异常，不指定时默认为1，表示从source的第一个字符开始匹配。
- bigint：返回值为Bigint类型。没有匹配时返回0。任一输入参数为NULL返回NULL。

示例如下：

```
regexp_count('abababc', 'a.c') = 1
regexp_count('abcde', '[[:alpha:]]{2}', 3) = 1
```

**SPLIT_PART**

命令格式如下：

```
string split_part(string str, string separator, bigint start[, bigint
end])
```

命令说明如下：

依照分隔符separator拆分字符串str，返回从第start部分到第end部分的子串（闭区间）。

参数说明：

- Str：String类型，要拆分的字符串。如果是Bigint，Double，Decimal或者Datetime类型会隐式转换到string类型后参加运算，其它类型报异常。
- Separator：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，其它类型会引发异常。

- start：Bigint类型常量，必须大于0。非常量或其它类型抛异常。返回段的开始编号（从1开始），如果没有指定end，则返回start指定的段。
- end：Bigint类型常量，大于等于start，否则抛异常。返回段的截止编号，非常量或其他类型会引发异常。可省略，缺省时表示最后一部分。
- String：返回值为String类型。若任意参数为NULL，返回NULL；若separator为空串，返回原字符串str。

 说明：

- 如果separator不存在于str中，且start指定为1，返回整个str。若输入为空串，输出为空串。
- 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，但start大于6，返回空串。
- 若end大于片段个数，按片段个数处理。

示例如下：

```
split_part('a,b,c,d', ',', 1) = 'a'
split_part('a,b,c,d', ',', 1, 2) = 'a,b'
split_part('a,b,c,d', ',', 10) = ''
```

**SUBSTR**

命令格式如下：

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

命令说明如下：

返回字符串str从start_position开始往后数，长度为length的子串。

参数说明：

- str：String类型，若输入为Bigint，Decimal，Double或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- start_position：Bigint类型，起始位置为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，往前数依次就是-2，-3…，其它类型抛异常。
- length：Bigint类型，大于0，其它类型或小于等于 0 抛异常。子串的长度。
- String：返回值为String类型。若任一输入为NULL，返回NULL。

 说明：

当length被省略时，返回到str结尾的子串。

示例如下：

```
substr("abc", 2) = "bc"
substr("abc", 2, 1) = "b"
substr("abc",-2,2)="bc"
substr("abc",-3)="abc"
```

**SUBSTRING**

命令格式如下：

```
string substring(string|binary str, int start_position[, int length])
```

命令说明如下：

返回字符串str从start_position开始往后数，长度为length的子串。

参数说明：

- str：String | Binary类型，若输入为其它类型返回null或报错。
- start_position：Int类型，起始位置为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，往前数依次就是-2，-3…，其它类型抛异常。
- length：Bigint类型，大于0，其它类型或小于等于0抛异常。子串的长度。
- String：返回值为String类型。若任一输入为NULL，返回NULL。

 说明：

当length被省略时，返回到str结尾的子串。

示例如下：

```
substring('abc', 2) = 'bc'
substring('abc', 2, 1) ='"b'
substring('abc',-2,2)='bc'
substring('abc',-3,2)='ab'
substring(BIN(2345),2,3)='001'
```

**TOLOWER**

命令格式如下：

```
string tolower(string source)
```

命令说明如下：

输出英文字符串source对应的小写字符串。

参数说明：

- source：String类型，若输入为Bigint，Double，Decimal或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
tolower("aBcd") = "abcd"
tolower("哈哈Cd") = "哈哈cd"
```

**TOUPPER**

命令格式如下：

```
string toupper(string source)
```

命令说明如下：

输出英文字符source串对应的大写字符串。

参数说明：

- source：String类型，若输入为Bigint，Double，Decimal或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
toupper("aBcd") = "ABCD"
toupper("哈哈Cd") = "哈哈CD"
```

**TO_CHAR**

命令格式如下：

```
string to_char(boolean value)
string to_char(bigint value)
string to_char(double value)
string to_char(decimal value)
```

命令说明如下：

将Boolean类型、Bigint类型、Decimal类型或者Double类型转为对应的String类型表示。

参数说明：

- value：可以接受Boolean类型、Bigint类型、Decimal类型或者Double类型输入，其它类型抛异常。对 Datetime类型的格式化输出请参考另一同名函数TO_CHAR 。
- String：返回值为String类型。如果输入为NULL，返回NULL。

示例如下：

```
to_char(123) = '123'
to_char(true) = 'TRUE'
to_char(1.23) = '1.23'
to_char(null) = NULL
```

**TRIM**

命令格式如下：

```
string trim(string str)
```

命令说明如下：

将输入字符串str去除左右空格。

参数说明：

- str：String类型，若输入为Bigint，Decimal，Double或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

**LTRIM**

命令格式如下：

```
string ltrim(string str)
```

命令说明如下：

将输入的字符串str去除左边空格。

参数说明：

- str：String类型，若输入为Bigint，Decimal，Double或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
select ltrim(' abc ') from dual;
返回：
```

```
+-----+
| _c0 |
+-----+
| abc |
+-----+
```

**RTRIM**

命令格式如下：

```
string rtrim(string str)
```

命令说明如下：

将输入的字符串str去除右边空格。

参数说明：

- str：String类型，若输入为Bigint，Decimal，Double或者Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
select rtrim('a abc ') from dual;
返回：
+-----+
| _c0 |
+-----+
| a abc |
+-----+
```

**REVERSE**

命令格式如下：

```
STRING REVERSE(string str)
```

命令说明如下：

返回倒序字符串。

参数说明：

- str：String类型，若输入为Bigint，Double，Decimal或Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
select reverse('abcedfg') from dual;
返回：
+-----+
| _c0 |
+-----+
| gfdecba |
+-----+
```

**SPACE**

命令格式如下：

```
STRING SPACE(bigint n)
```

命令说明如下：

空格字符串函数，返回长度为n的字符串。

参数说明：

- n: Bigint类型。长度不超过2M。如果为空，则抛异常。
- String：返回值为String类型。

示例如下：

```
select length(space(10)) from dual; ----返回10。
select space(400000000000) from dual； ----报错，长度超过2M。
```

**REPEAT**

命令格式如下：

```
STRING REPEAT(string str, bigint n)
```

命令说明如下：

返回重复n次后的str字符串。

参数说明：

- str：String类型，若输入为Bigint，Double，Decimal或Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- n: Bigint类型。长度不超过2M。如果为空，则抛异常。
- String：返回值为String类型。

示例如下：

```
select repeat('abc',5) from lxw_dual;
返回：abcabcabcabcabc
```

### ASCII

命令格式如下：

```
Bigint ASCII(string str)
```

命令说明如下：

返回字符串str第一个字符的ascii码。

参数说明：

- str：String类型，若输入为Bigint，Double，Decimal或Datetime类型会隐式转换为String后参与运算，其它类型报异常。
- Bigint：返回值为Bigint类型。

示例如下：

```
select ascii('abcde') from dual;
返回值：97
```

### MaxCompute2.0扩展函数

升级到MaxCompute2.0后，产品扩展部分数学函数，新函数若用到新数据类型时，在使用新函数的SQL前，需要加一个set语句：

```
set odps.sql.type.system.odps2=true;
```

说明：

请在用到该函数的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。

下文将为您详细介绍新扩展的字符串函数。

### CONCAT_WS

命令格式如下：

```
string concat_ws(string SEP, string a, string b...)
```

```
string concat_ws(string SEP, array)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

返回值是将参数中的所有字符串安装指定的分隔符连接在一起的结果。

参数说明：

- SEP：String类型的分隔符，若不指定，返回的值异常。
- a/b等：String类型，若输入为Bigint，Decimal，Double或Datetime类型会隐式转换为String后参与运算，其它类型报异常。

返回值：

返回String类型。如果没有参数或者某个参数为NULL，结果均返回NULL。

示例如下：

```
concat_ws(':','name','hanmeimei')='name:hanmeimei'
concat_ws(':','avg',null,'34')=null
```

**LPAD**

命令格式如下：

```
string lpad(string a, int len, string b)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

用b字符串将a字符串向左补足到len位。

参数说明：

- len：Int整型。
- a/b等：String类型。

返回值：

为String类型。若len小于a的位数，则返回a从左开始截取len位字符；若len为0则返回空。

示例如下：

```
lpad('abcdefgh',10,'12')='12abcdefgh'
lpad('abcdefgh',5,'12')='abcde'
lpad('abcdefgh',0,'12')返回空
```

**RPAD**

命令格式如下：

```
string rpad(string a, int len, string b)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

命令说明如下：

用b字符串将a字符串向右补足到len位。

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

参数说明：

- len：Int整型。
- a/b等：String类型。

返回值：

为String类型。若len小于a的位数，则返回a从左开始截取len位字符；若len为0则返回空。

示例如下：

```
rpad('abcdefgh',10,'12')='abcdefgh12'
rpad('abcdefgh',5,'12')='abcde'
```

```
rpad('abcdefgh',0,'12')返回空
```

**REPLACE**

命令格式如下：

```
string replace(string a, string OLD, string NEW)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

用NEW字符串替换a字符串中与OLD字符串完全重合的部分并返回a。

参数说明：

参数说明：参数都为String类型。

返回值：

返回String类型，若某个参数为null，则返回null。

示例如下：

```
replace('ababab','abab','12')='12ab'
replace('ababab','cdf','123')='ababab'
```

```
replace('123abab456ab',null,'abab')=null
```

### SOUNDEX

命令格式如下：

```
string soundex(string a)
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

将普通字符串转换成soundex字符串。

参数说明：a为String类型。

返回值：返回String类型，若输入为null，则返回null。

示例如下：

```
soundex('hello')='H400'
```

### SUBSTRING_INDEX

命令格式如下：

```
string substring_index(string a, string SEP, int count))
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

命令说明如下：

截取字符串a第count分隔符之前的字符串，如count为正则从左边开始截取，如果为负则从右边开始截取。

参数说明：a/sep为string类型，count为int类型。

返回值：

返回String类型，若输入为null，则返回null。

示例如下：

```
substring_index('https://help.aliyun.com', '.', 2)='https://help.
aliyun'
substring_index('https://help.aliyun.com', '.', -2)='aliyun.com'
substring_index('https://help.aliyun.com', null, 2)=null
```

## 1.10.6 其他函数

### CAST

命令格式如下：

```
cast(expr as <type>)
```

用途：将表达式的结果转换成目标类型，如cast('1' as bigint)将字符串'1'转为整数类型的1，如果转换不成功或不支持的类型转换会引发异常。

 说明：

- cast(double as bigint)，将double值转换成bigint。
- cast(string as bigint) 在将字符串转为bigint时，如果字符串中是以整型表达的数字，会直接转为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类型，再转为bigint类型。
- cast(string as datetime) 或 cast(datetime as string)时，会采用默认的日期格式yyyy-mm-dd hh:mi:ss。

**COALESCE**

命令格式如下：

```
coalesce(expr1, expr2, ...)
```

命令说明如下：

该函数用于返回列表中第一个非Null的值，如果列表中所有的值都是Null，则返回Null。

参数说明：

expr是要测试的值，所有这些值类型必须相同或为Null，否则会引发异常。

返回值：

返回值类型和参数类型相同。

 说明：

至少要有一个参数，否则引发异常。

**DECODE**

命令格式如下：

```
decode(expression, search, result[, search, result]...[, default])
```

命令说明如下：

该函数用于实现if-then-else分支选择的功能。

参数说明：

- expression：要比较的表达式。
- search：和expression进行比较的搜索项。
- result：search和expression的值匹配时的返回值。
- default：可选项，如果所有的搜索项都不匹配，则返回此default值，如果未指定，则返回Null。

返回值：

- 返回匹配的search。

- 如果没有匹配，返回default。
- 如果没有指定default，返回Null。

> 说明：
>
> - 至少要指定三个参数。
> - 所有的result类型必须一致，或为Null。不一致的数据类型会引发异常。所有的search和expression类型必须一致，否则报异常。
> - 如果decode中的search选项有重复时且匹配时，会返回第一个值。

示例如下：

```
select
decode(customer_id,
1, 'Taobao',
2, 'Alipay',
3, 'Aliyun',
Null, 'N/A',
'Others') as result
from sale_detail;
```

上面的decode函数实现了下面if-then-else语句中的功能：

```
if customer_id = 1 then
result := 'Taobao';
elsif customer_id = 2 then
result := 'Alipay';
elsif customer_id = 3 then
result := 'Aliyun';
...
else
result := 'Others';
end if;
```

> 说明：
>
> - 通常情况下，MaxCompute SQL在计算Null=Null时返回Null，但在decode函数中，Null与Null的值是相等的。
> - 上述示例中，当customer_id的值为Null时，decode函数返回N/A。

**GET_IDCARD_AGE**

命令格式如下：

```
get_idcard_age(idcardno)
```

命令说明如下：

该函数用于根据身份证号返回当前的年龄，当前年份减去身份证中标识的出生年份的差值。

参数说明：

idcardno：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，如果校验不通过会返回Null。

返回值：

返回Bigint类型，输入为Null，返回Null。如果当前年份减去出生年份差值大于100，返回Null。

**GET_IDCARD_BIRTHDAY**

命令格式如下：

```
get_idcard_birthday(idcardno)
```

命令说明如下：

该函数用于根据身份证号返回出生日期。

参数说明：

idcardno：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：

返回Datetime类型，输入为Null，返回Null。

**GET_IDCARD_SEX**

命令格式如下：

```
get_idcard_sex(idcardno)
```

命令说明如下：

该函数用于根据身份证号返回性别，值为M（男）或F（女）。

参数说明：

idcardno：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：

返回String类型，输入为Null，返回Null。

### GREATEST

命令格式如下：

```
greatest(var1, var2, …)
```

命令说明如下：

该函数用于返回输入参数中最大的一个。

参数说明：

var1，var2可以为Bigint，Double，Decimal，Datetime或String类型。若所有值都为Null，则返回Null。

返回值：

- 输入参数中的最大值，当不存在隐式转换时返回同输入参数类型。
- Null为最小值。

  当输入参数类型不同时：

  - Double，Bigint，Decimal，String之间的比较转为Double类型。
  - String，Datetime的比较转为Datetime类型。
  - 不允许其它的隐式转换。

### ORDINAL

命令格式如下：

```
ordinal(bigint nth, var1, var2, …)
```

命令说明如下：

该函数用于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：Bigint类型，指定要返回的位置为Null时，返回Null。
- var1，var2：类型可以为Bigint，Double，Datetime或String类型。

返回值：

- 排在第nth位的值，当不存在隐式转换时返回同输入参数类型。
- 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。
- 不允许其它的隐式转换。

- Null为最小。

示例如下：

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) = 2
```

**LEAST**

命令格式如下：

```
least(var1, var2, …)
```

命令说明如下：

该函数用于返回输入参数中最小的一个。

参数说明：

var1，var2可以为Bigint，Double，Decimal，Datetime或String类型。若所有值都为Null，则返回Null。

返回值：

- 输入参数中的最小值，当不存在隐式转换时返回同输入参数类型。
- 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和Double，Bigint，String之间比较时转为Decimal类型。
  - 不允许其它的隐式类型转换。
- Null为最小。

**MAX_PT**

命令格式如下：

```
max_pt(table_full_name)
```

命令说明如下：

对于分区的表，此函数返回该分区表的一级分区的最大值，按字母排序，且该分区下有对应的数据文件。

参数说明：

table_full_name：String类型，指定表名（必须带上project名，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。

示例如下：

假设tbl是分区表，该表对应的分区如下，且都有数据文件：

```
pt ='20120901'
pt ='20120902'
```

则以下语句中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');
```

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文件，则此分区不会做为返回值。

**UUID**

命令格式如下：

```
string uuid()
```

命令说明如下：

该函数用于返回一个随机ID，示例样式为29347a88-1e57-41ae-bb68-a9edbdd94212。

说明：

UUID返回的是一个随机的全局ID，其重复的概率很小。

### SAMPLE

命令格式如下：

```
boolean sample(x, y, column_name)
```

命令说明如下：

对所有读入的column_name的值，sample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：Bigint类型，表示哈希为x份，取第y份。
  - y可省略，省略时取第一份，如果省略参数中的y，则必须同时省略column_name。
  - x，y为整型常量，大于0，其它类型或小于等于0时抛异常，若y&gt，x也抛异常。x，y任一输入为Null时，返回Null。
- column_name：是采样的目标列。
  - column_name可以省略，省略时根据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，则报异常。

返回值：

返回Boolean类型。

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。如果不加column_name，则数据量比较少时输出不一定均匀，在这种情况下建议加上column_name，以获得比较好的输出结果。

示例如下：

假定存在表tbla，表内有列名为cola的列。

```
select * from tbla where sample (4, 1 , cola) = true;
-- 表示数值会根据cola hash为4份，取第1份
select * from tbla where sample (4, 2) = true;
```

```
-- 表示数值会对每行数据做随机哈希分配为4份，取第2份
```

**CASE WHEN表达式**

MaxCompute提供两种case when的语法格式，如下所示：

```
case value
when (_condition1) then result1
when (_condition2) then result2
...
else resultn
end
case
when (_condition1) then result1
when (_condition2) then result2
when (_condition3) then result3
...
else resultn
end
```

case when表达式可以根据表达式value的计算结果，灵活返回不同的值。

根据shop_name的不同情况得出所属区域，示例如下：

```
select
case
when shop_name is null then 'default_region'
when shop_name like 'hang%' then 'zj_region'
end as region
from sale_detail;
```

说明：

- 如果result类型只有Bigint，Double，统一转为Double后，再返回。
- 如果result类型中有String类型，统一转为String后，再返回。如果不能转则报错（如Boolean型）。
- 除此之外不允许其它类型之间的转换。

**IF表达式**

命令格式如下：

```
if(testCondition, valueTrue, valueFalseOrNull)
```

命令说明如下：

判断testCondition是否为真。如果为真，返回valueTrue，如果不满足则返回另一个值（valueFalse或者Null）。

参数说明：

- testCondition：要判断的表达式，Boolean类型。
- valueTrue：表达式testCondition为True时，返回的值。
- valueFalseOrNull：不满足表达式testCondition时，返回的值可以设为Null。

返回值：

返回值类型和参数valueTrue或者valueFalseOrNul的类型一致。

示例如下：

```
select if(1=2,100,200) from dual;
返回值：
+------------+
| _c0 |
+------------+
| 200 |
+------------+
```

**MaxCompute2.0扩展支持的其他函数类型**

**SPLIT**

命令格式如下：

```
split(str, pat)
```

用途：通过pat将str分割后返回数组。

参数说明：

- str：String类型，指被分隔的字符串。
- pat：String类型，分隔符，支持正则。

返回值：

`array <string >`，元素是str被pat分隔后的结果。

示例如下：

```
select split("a,b,c",",") from dual;
结果如下：
+------+
| _c0 |
+------+
| [a, b, c] |
```

```
+------+
```

**说明：**

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

**STR_TO_MAP**

函数声明:

```
str_to_map(text [, delimiter1 [, delimiter2]])
```

用途：使用delimiter1将text分割成K-V对，然后使用delimiter2分割每个K-V对的K和V。

参数说明

- text：String类型，指被分割的字符串；
- delimiter1: String类型，分隔符, 不指定时默认为’,’;
- delimiter2：String类型，分隔符, 不指定时默认为’=’;

返回值：map < string, string >, 元素是text被delimiter1和delimiter2分割后的K-V结果

示例：

```
select str_to_map('test1&1-test2&2','-','&');
```

返回结果：

```
+------------+
| a          |
+------------+
```

```
| {test1:1, test2:2} |
```

**EXPLODE**

命令格式如下：

```
explode (var)
```

命令说明如下：

该函数用于将一行数据转为多行的UDTF。

- 如果var是array，则将列中存储的array转为多行。
- 如果var是map，则将列中存储的map的每个key-value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`：`array<T>` 类型或者 `map<K, V>` 类型。

返回值：

返回转换后的行。

说明：

UDTF在使用上有以下限制：

- 在一个select中只能有一个UDTF，不可出现其它的列。
- 不可以与group by/cluster by/distribute by/sort by一起使用。

示例如下：

```
explode(array(null, 'a', 'b', 'c')) col
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

**MAP**

命令格式如下：

```
MAP map(K key1, V value1, K key2, V value2, ...)
```

命令说明如下：

该函数用于使用给定的key-value对建立map。

参数说明：

key/value

- 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。
- 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：

返回MAP 类型。

示例如下：

如表t_table的字段为(c1 bigint,c2 string,c3 string, c4 bigint ,c5 bigint),数据如下

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
| 1000       | k11 | k21 | 86         | 15         |
| 1001       | k12 | k22 | 97         | 2          |
| 1002       | k13 | k23 | 99         | 1          |
+------------+----+----+------------+------------+
```

执行sql：

```
select  map(c2,c4,c3,c5) from t_table;
```

结果如下

```
+------+
| _c0  |
+------+
| {k11:86, k21:15} |
```

```
| {k12:97, k22:2} |
| {k13:99, k23:1} |
+------+
```

### MAP_KEYS

命令格式如下：

```
ARRAY map_keys(map<K, V> )
```

命令说明如下：

该函数用于将参数map中的所有key作为数组返回。

参数说明：

map：map类型的数据。

返回值：

返回ARRAY类型，输入Null，则返回Null。

示例如下:

如表t_table_map的字段为(`c1 bigint,t_map map<string,bigint>` ),数据如下

```
+------------+-------+
| c1         | t_map |
+------------+-------+
| 1000       | {k11:86, k21:15} |
| 1001       | {k12:97, k22:2} |
| 1002       | {k13:99, k23:1} |
+------------+-------+
```

执行sql：

```
select  c1,map_keys(t_map) from t_table_map;
```

结果如下

```
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | [k11, k21] |
| 1001       | [k12, k22] |
| 1002       | [k13, k23] |
```

```
+------------+------+
```

### MAP_VALUES

命令格式如下：

```
ARRAY map_values(map<K, V>)
```

命令说明如下：

该函数用于将参数map中的所有values作为数组返回。

参数说明：

map：map类型的数据。

返回值：

返回ARRAY类型，输入Null，返回Null。

示例如下：

```
select map_values(map('a',123,'b',456));
--结果如下：
[123, 456]
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

### ARRAY

命令格式如下：

```
ARRAY array(value1,value2, ...)
```

命令说明如下：

该函数用于使用给定的value构造array。

参数说明：

value：value可为任意类型，但是所有value的类型必须一致。

返回值：

返回array类型。

示例如下：

如表t_table的字段为(c1 bigint,c2 string,c3 string, c4 bigint ,c5 bigint),数据如下

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
| 1000       | k11 | k21 | 86         | 15         |
| 1001       | k12 | k22 | 97         | 2          |
| 1002       | k13 | k23 | 99         | 1          |
+------------+----+----+------------+------------+
```

执行sql：

```
select array(c2,c4,c3,c5) from t_table;
```

结果如下：

```
+------+
| _c0  |
+------+
| [k11, 86, k21, 15] |
| [k12, 97, k22, 2] |
| [k13, 99, k23, 1] |
+------+
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

**SIZE**

命令格式如下：

```
INT size(map)
INT size(array)
```

命令说明如下：

- `size(map<K , V>)`返回给定map中K/V对数。
- `size(array<T>)`返回给定的array中的元素数目。

参数说明：

- `map<K, V>`：map类型的数据。
- `array<T>`：array类型的数据。

返回值：

返回int类型。

示例如下：

```
select size(map('a',123,'b',456)) from dual;--返回2
select size(map('a',123,'b',456,'c',789)) from dual;--返回3
select size(array('a','b')) from dual;--返回2
select size(array(123,456,789)) from dual;--返回3
```

说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

```
- setproject odps.sql.type.system.odps2=true;
```

对setproject的详细说明请参见：其他操作。

**ARRAY_CONTAINS**

命令格式如下：

```
boolean array_contains(ARRAY<T> a,value v)
```

命令说明如下：

该函数用于检测给定array a中是否包含v。

参数说明：

- a：array类型的数据。
- v：给出的v必须与array数组中的数据类型一致。

返回值：

返回Boolean类型。

示例如下：

如表t_table_array的字段为(c1 bigint, t_array array<string>),数据如下：

```
+------------+---------+
| c1         | t_array |
+------------+---------+
| 1000       | [k11, 86, k21, 15] |
| 1001       | [k12, 97, k22, 2] |
| 1002       | [k13, 99, k23, 1] |
+------------+---------+
```

执行sql：

```
select c1, array_contains(t_array,'1') from t_table_array;
```

结果如下：

```
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | false |
| 1001       | false |
| 1002       | true |
```

```
+------------+------+
```

### SORT_ARRAY

命令格式如下：

```
ARRAY sort_array(ARRAY<T>)
```

命令说明如下：

该函数用于为给定的数组排序。

参数说明：

ARRAY<T>：array类型的数据，数组中的数据可为任意类型。

返回值：

返回array类型。

示例如下：

```
select sort_array(array('a','c','f','b')),sort_array(array(4,5,7,2,5,8
)),sort_array(array('你','我','他')) from dual;
结果如下：
[a, b, c, f] [2, 4, 5, 5, 7, 8] [他, 你, 我]
```

执行sql

```
select sort_array(c1),sort_array(c2),sort_array(c3) from t_array;
```

返回结果:

```
[a, b, c, f] [2, 4, 5, 5, 7, 8] [他, 你, 我]
```

 说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。

- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

**POSEXPLODE**

命令格式如下：

```
posexplode(ARRAY<T>)
```

命令说明如下：

该函数用于将给定的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。

参数说明：

ARRAY：array类型的数据，数组中的数据可为任意类型。

返回值：

返回表生成的函数。

示例如下：

```
select posexplode(array('a','c','f','b')) from dual;
结果如下：
+------------+-----+
| pos | val |
+------------+-----+
| 0 | a |
| 1 | c |
| 2 | f |
| 3 | b |
+------------+-----+
```

**STRUCT**

命令格式如下：

```
STRUCT struct(value1,value2, ...)
```

函数说明：

该函数用于使用给定value列表建立struct。

参数说明：

value：各value可为任意类型。

返回值：

返回STRUCT<col1:T1, col2:T2, ...>类型。field的名称依次为col1，col2，…。

示例如下：

```
select struct('a',123,'ture',56.90) from dual;
结果如下：
{col1:a, col2:123, col3:ture, col4:56.9}
```

 说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

**NAMED_STRUCT**

命令格式如下：

```
STRUCT named_struct(string name1, T1 value1, string name2, T2 value2
, ...)
```

函数说明：

该函数用于使用给定的name/value列表建立struct。

参数说明：

- value：各value可为任意类型。
- name：指定的String类型的field名称。

返回值：

返回STRUCT<name1:T1, name2:T2, ...>类型，生成struct的field的名称依次为name1，name2，…。

示例如下：

```
select named_struct('user_id',10001,'user_name','LiLei','married','F
','weight',63.50) from dual;
结果如下：
{user_id:10001, user_name:LiLei, married:F, weight:63.5}
```

 说明：

目前MaxCompute SQL及新版本Mapreduce支持的Set命令分为以下两种方式：

- session级别：要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- project级别：即支持对project级别进行新类型打开。project的Owner可根据需要对project进行设置，命令为：

  ```
  - setproject odps.sql.type.system.odps2=true;
  ```

  对setproject的详细说明请参见：其他操作。

**INLINE**

命令格式如下：

```
inline(array<struct<f1:T1, f2:T2, ...>>)
```

命令说明如下：

该函数用于将给定的struct数组展开，每个元素对应一行，每行每个struct元素对应一列。

参数说明：

`STRUCT<f1:T1, f2:T2, ...>`：数组中的value可为任意类型。

返回值：

返回表生成的函数。

示例如下:

如表t_table的字段为(`t_struct struct<user_id:bigint,user_name:string,married:string,weight:double>`),表数据如下：

```
+----------+
| t_struct |
+----------+
| {user_id:10001, user_name:LiLei, married:N, weight:63.5} |
| {user_id:10002, user_name:HanMeiMei, married:Y, weight:43.5} |
+----------+
```

执行sql：

```
select inline(array(t_struct)) from t_table;
```

返回结果:

```
+------------+-----------+---------+------------+
| user_id    | user_name | married | weight     |
+------------+-----------+---------+------------+
| 10001      | LiLei     | N       | 63.5       |
| 10002      | HanMeiMei | Y       | 43.5       |
+------------+-----------+---------+------------+
```

**TRANS_ARRAY**

命令格式如下：

```
trans_array (num_keys, separator, key1,key2,…,col1, col2,col3) as (key1,key2,…,col1, col2)
```

命令说明如下：

用于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： Bigint类型常量，必须>=0。在转为多行时作为转置key的列的个数。
- Key：是指在将一行转为多行时，在多行中重复的列。
- separator：String类型常量，用于将字符串拆分成多个元素的分隔符。为空时报异常。
- keys：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： 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，如Hangzhou;Beijing;shanghai，是以（;）分隔的数组。

返回值：

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。

 说明：

UDTF使用上的限制如下：

- 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，不可以再出现其它的列。
- 不可以与group by/cluster by/distribute by/sort by一起使用。

示例如下：

t_table表中的数据如：

```
+----------+----------+------------+
| 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;
```

产生的数据如下所示：

```
+----------+----------+------------+
| 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
```

则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
```

# 1.11 UDF

## 1.11.1 UDF概述

UDF 全称为 User Defined Function，即用户自定义函数。MaxCompute 提供很多内建函数来满足您的计算需求，同时您还可以通过创建自定义函数来满足不同的计算需求。UDF 在使用上与普通的内建函数 类似，Java 和 MaxCompute 的数据类型的对应关系，请参见 参数与返回值类型。

如果您使用 *Maven*，可以从 *Maven* 库 中搜索 odps-sdk-udf，从而获取不同版本的 Java SDK，相关配置信息如下所示：

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-udf</artifactId>
   <version>0.20.7-public</version>
</dependency>
```

MaxCompute支持的UDF有三种：

| **UDF** 分类 | 描述 |
| --- | --- |
| User Defined Scalar Function（通常也称之为 UDF） | 用户自定义标量值函数（User Defined Scalar Function）。其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。 |
| UDTF（User Defined Table Valued Function） | 自定义表值函数，是用来解决一次函数调用输出多行数据场景的，也是唯一能返回多个字段的自定义函数。而 UDF 只能一次计算输出一条返回值。 |
| UDAF（User Defined Aggregation Function） | 自定义聚合函数，其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 SQL 中的 Group By 语句联用。具体语法请参见 聚合函数。 |

 说明：

- UDF 广义的说法代表了自定义标量函数，自定义聚合函数及自定义表函数三种类型的自定义函数的集合。狭义来说，仅代表用户自定义标量函数。文档会经常使用这一名词，请读者根据文档上下文判断具体含义。
- SQL 语句中有使用自定义的函数，提示内存不够。请配置 `set odps.sql.udf.joiner.jvm.memory=xxxx;`。原因是数据量太大并且有倾斜，任务超出默认设置的内存。

MaxCompute的UDF支持跨项目分享，即在project_a中可以使用project_b的UDF。具体的授权可以参考安全指南的授权文档中的“跨项目空间Table、Resource、Function分享示例”章节，使用时写法如`select other_project:udf_in_other_project(arg0, arg1) as res from table_t;`。

**UDF 示例**

UDF 的相关示例请参见 *UDF 示例*。

## 1.11.2 Java UDF

MaxCompute 的 UDF 包括：UDF，UDAF 和 UDTF 三种函数，本文将重点介绍如何通过 Java 实现这三种函数。

**参数与返回值类型**

MaxCompute2.0 版本升级后，Java UDF 支持的数据类型从原来的 Bigint，String，Double，Boolean 扩展了更多基本的数据类型，同时还扩展支持了 ARRAY，MAP，STRUCT 等复杂类型。

- Java UDF 使用新基本类型的方法，如下所示：
  - UDTF 通过 @Resolve 注解来获取 signature，如：`@Resolve("smallint->varchar(10)")`。
  - UDF 通过反射分析 evaluate 来获取 signature，此时 MaxCompute 内置类型与 Java 类型符合一一映射关系。
  - UDAF通过 @Resolve 注解来获取 signature，MaxCompute2.0支持在注解中使用新类型，如：`@Resolve("smallint->varchar(10)")`。
- Java UDF 使用复杂类型的方法，如下所示：
  - UDTF 通过 @Resolve annotation 来指定 sinature，如：`@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")`。
  - UDF 通过 evaluate 方法的 signature 来映射 UDF 的输入输出类型，此时参考 MaxCompute 类型与 Java 类型的映射关系。其中 array 对应 java.util.List，map 对应 java.util.Map，struct 对应 com.aliyun.odps.data.Struct。
  - UDAF通过 @Resolve 注解来获取 signature，MaxCompute2.0支持在注解中使用新类型，如：`@Resolve("smallint->varchar(10)")`。

说明：

- com.aliyun.odps.data.Struct 从反射看不出 field name 和 field type，所以需要用 @Resolve annotation 来辅助。即如果需要在 UDF 中使用 struct，要求在 UDF class 上也标注上 @Resolve 注解，这个注解只会影响参数或返回值中包含 com.aliyun.odps.data.Struct 的重载。
- 目前 *class* 上只能提供一个 *@Resolve annotation*#因此一个 *UDF* 中带有 *struct* 参数或返回值的重载只能有一个。

MaxCompute 数据类型与 Java 类型的对应关系，如下所示：

| MaxCompute Type | Java Type |
| --- | --- |
| Tinyint | java.lang.Byte |
| Smallint | java.lang.Short |
| Int | java.lang.Integer |
| Bigint | java.lang.Long |
| Float | java.lang.Float |
| Double | java.lang.Double |
| Decimal | java.math.BigDecimal |
| Boolean | java.lang.Boolean |
| String | java.lang.String |
| Varchar | com.aliyun.odps.data.Varchar |
| Binary | com.aliyun.odps.data.Binary |
| Datetime | java.util.Date |
| Timestamp | java.sql.Timestamp |
| Array | java.util.List |
| Map | java.util.Map |
| Struct | com.aliyun.odps.data.Struct |

说明：

- 在UDF中使用输入或输出参数的类型请务必使用Java Type，否则会报错ODPS-0130071。
- Java 中对应的数据类型以及返回值数据类型是对象，首字母请务必大写。

- SQL 中的 NULL 值通过 Java 中的 NULL 引用表示，因此 Java primitive type 是不允许使用的，因为无法表示 SQL 中的 NULL 值。
- 此处 Array 类型对应的 Java 类型是 List，而不是数组。

**UDF**

实现 UDF 需要继承 com.aliyun.odps.udf.UDF 类，并实现 evaluate 方法。evaluate 方法必须是非 static 的 public 方法 。Evaluate 方法的参数和返回值类型将作为 SQL 中 UDF 的函数签名。这意味着您可以在 UDF 中实现多个 evaluate 方法，在调用 UDF 时，框架会依据 UDF 调用的参数类型匹配正确的 evaluate 方法 。

**特别注意**：不同的jar包最好不要有类名相同但实现功能逻辑不一样的类。如，UDF(UDAF/UDTF)： udf1、 udf2分别对应资源udf1.jar、udf2.jar，如果两个jar包里都包含一个com.aliyun.UserFunction.class 类，当同一个sql中同时使用到这两个udf时，系统会随机加载其中一个类，那么就会导致udf执行行为不一致甚至编译失败。

UDF 的示例如下：

```
package org.alidata.odps.udf.examples;
  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();
  }
}
```

可以通过实现`void setup(ExecutionContext ctx)`和`void close()`来分别实现 UDF 的初始化和结束代码。

UDF 的使用方式与 MaxCompute SQL 中普通的内建函数相同，详情请参见 内建函数。

新版的MaxCompute支持定义Java UDF时，使用Writable类型作为参数和返回值。下面为MaxCompute类型和Java Writable类型的映射关系。

| MaxCompute Type | Java Writable Type |
| --- | --- |
| tinyint | ByteWritable |
| smallint | ShortWritable |
| int | IntWritable |
| bigint | LongWritable |

| MaxCompute Type | Java Writable Type |
| --- | --- |
| float | FloatWritable |
| double | DoubleWritable |
| decimal | BigDecimalWritable |
| boolean | BooleanWritable |
| string | Text |
| varchar | VarcharWritable |
| binary | BytesWritable |
| datetime | DatetimeWritable |
| timestamp | TimestampWritable |
| interval_year_month | IntervalYearMonthWritable |
| interval_day_time | IntervalDayTimeWritable |
| array | 暂不支持 |
| map | 暂不支持 |
| struct | 暂不支持 |

**其他 UDF 示例**

如以下代码，定义了一个有三个 overloads 的 UDF，其中第一个用了 array 作为参数，第二个用了 map 作为参数，第三个用了 struct。由于第三个 overloads 用了 struct 作为参数或者返回值，因此要求必须要对 UDF class 打上 `@Resolve` annotation，来指定 struct 的具体类型。

```
@Resolve("struct,string->string")
public class UdfArray extends UDF {
  public String evaluate(List vals, Long len) {
    return vals.get(len.intValue());
  }
  public String evaluate(Map map, String key) {
    return map.get(key);
  }
  public String evaluate(Struct struct, String key) {
  return struct.getFieldValue("a") + key;
 }
}
```

用户可以直接将复杂类型传入 UDF 中：

```
create function my_index as 'UdfArray' using 'myjar.jar';
```

```
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as
color_name from co
```

### UDAF

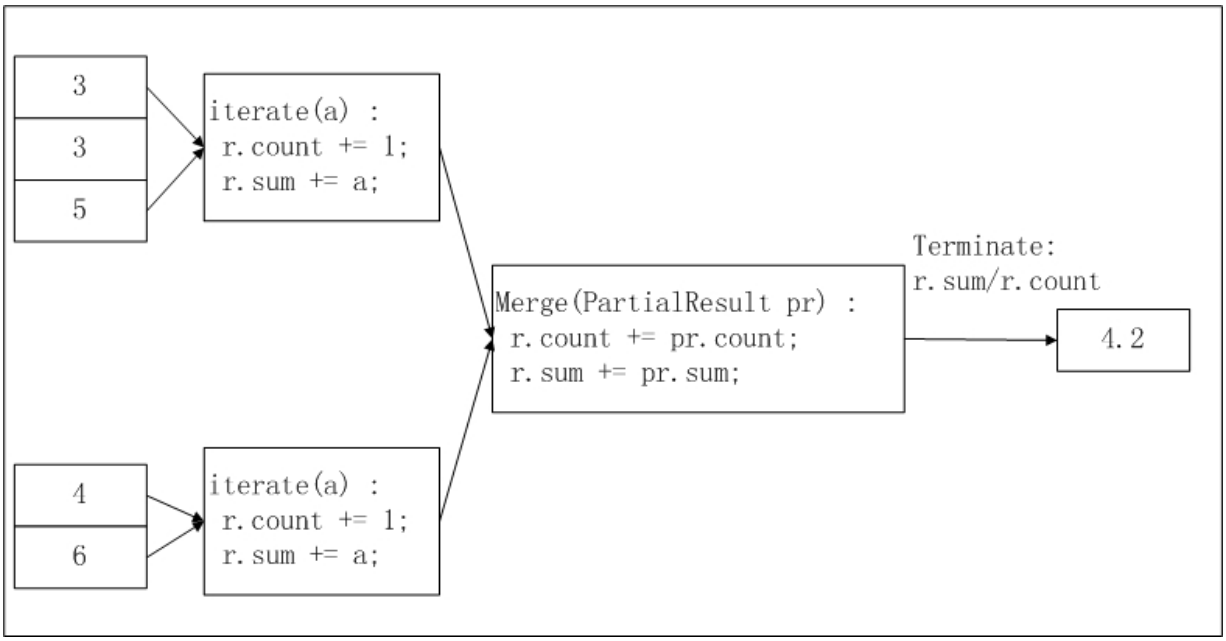
实现 Java UDAF 类需要继承 com.aliyun.odps.udf.Aggregator，并实现如下几个接口：

```
public abstract class Aggregator implements ContextFunction {
  @Override
  public void setup(ExecutionContext ctx) throws UDFException {

  }
  @Override
  public void close() throws UDFException {
  }
  /**
   * 创建聚合Buffer
   * @return Writable 聚合buffer
   */
  abstract public Writable newBuffer();
  /**
   * @param buffer 聚合buffer
   * @param args 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
   * @throws UDFException
   */
  abstract public void iterate(Writable buffer, Writable[] args)
throws UDFException;
  /**
   * 生成最终结果
   * @param buffer
   * @return Object UDAF的最终结果
   * @throws UDFException
   */
  abstract public Writable terminate(Writable buffer) throws
UDFException;
  abstract public void merge(Writable buffer, Writable partial) throws
 UDFException;
}
```

其中最重要的是 iterate，merge 和 terminate 三个接口，UDAF 的主要逻辑依赖于这三个接口的实现。此外，还需要您实现自定义的 Writable buffer。

以实现求平均值 avg 为例，下图简要说明了在 MaxCompute UDAF 中这一函数的实现逻辑及计算流程：




在上图中，输入数据被按照一定的大小进行分片（有关分片的描述请参见 *MapReduce*），每片的大小适合一个 worker 在适当的时间内完成。这个分片大小的设置需要您手动配置完成。

UDAF 的计算过程分为两个阶段：

- 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。
- 第二阶段：worker 汇总上一个阶段中每个分片内的信息。在最终输出时，r.sum / r.count 即是所有输入数据的平均值。

计算平均值的 UDAF 的代码示例，如下所示：

```
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;
  }
}
```

 说明：

- UDAF 在 SQL 中的使用语法与普通的内建聚合函数相同，详情请参见 聚合函数。
- 关于如何运行 UDTF 的方法与 UDF 类似，详情请参见 运行 *UDF*。
- String对应的Writable类型为Text。

**UDTF**

Java UDTF 需要继承 com.aliyun.odps.udf.UDTF 类。这个类需要实现 4 个接口，如下表所示：

| 接口定义 | 描述 |
| --- | --- |
| public void setup(ExecutionContext ctx) throws UDFException | 初始化方法，在UDTF处理输入数据前，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 | 这个方法由框架调用，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。 |

| 接口定义 | 描述 |
| --- | --- |
| | 输入参数以Object[]的形式传入，输出结果通过forward函数输出。用户需要在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，此方法由框架调用，并且只会被调用一次，即在处理完最后一条记录之后。 |
| public void forward(Object …o) throws UDFException | 用户调用forward方法输出数据，每次forward代表输出一条记录。对应SQL语句UDTF的as子句指定的列。 |

UDTF 的程序示例，如下所示：

```
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);
       }
     }
   }
```

**说明：**

以上只是程序示例，关于如何在 MaxCompute 中运行 UDTF 的方法与 UDF 类似，详情请参见：运行 *UDF*。

在 SQL 中可以这样使用这个 UDTF，假设在 MaxCompute 上创建 UDTF 时注册函数名为 user_udtf：

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
```

假设 my_table 的 col0，col1 的值如下所示：

```
+------+------+
| col0 | col1 |
+------+------+
| A B | 1 |
| C D | 2 |
```

```
+------+------+
```

则 select 出的结果，如下所示：

```
+----+----+
| c0 | c1 |
+----+----+
| A  | 1  |
| B  | 1  |
| C  | 2  |
| D  | 2  |
+----+----+
```

### 使用说明

UDTF 在 SQL 中的常用方式如下：

```
select user_udtf(col0, col1, col2) as (c0, c1) from my_table;
select user_udtf(col0, col1, col2) as (c0, c1) from (select * from
my_table distribute by key sort by key) t;
select reduce_udtf(col0, col1, col2) as (c0, c1) from (select col0,
col1, col2 from (select map_udtf(a0, a1, a2, a3) as (col0, col1, col2
) from my_table) t1 distribute by col0 sort by col0, col1) t2;
```

但使用 UDTF 有如下使用限制：

- 同一个 SELECT 子句中不允许有其他表达式。

  ```
  select value, user_udtf(key) as mycol ...
  ```

- UDTF 不能嵌套使用。

  ```
  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。

  ```
  select user_udtf(key) as mycol ... group by mycol
  ```

### 其他 UDTF 示例

在 UDTF 中，您可以读取 MaxCompute 的 资源。利用 UDTF 读取 MaxCompute 资源的示例，如下所示：

1. 编写 UDTF 程序，编译成功后导出 jar 包（udtfexample1.jar）。

   ```
   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_reso
urce1").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);
    }
}
```

2. 添加资源到 MaxCompute。

```
Add file file_resource.txt;
Add jar udtfexample1.jar;
Add table table_resource1 as table_resource1;
```

```
Add table table_resource2 as table_resource2;
```

3. 在 MaxCompute 中创建 UDTF 函数（my_udtf）。

```
create function mp_udtf as com.aliyun.odps.examples.udf.UDTFResource
 using
'udtfexample1.jar, file_resource.txt, table_resource1, table_reso
urce2';
```

4. 在 MaxCompute 创建资源表 table_resource1、table_resource2 和物理表 tmp1，并插入相应的数据。

5. 运行该 UDTF。

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from tmp1
;
返回：
+-------+------------+-------+
| a | b      | fileResourceLineCount |
+-------+------------+-------+
| 10    | 2          | fileResourceLineCount=3|tableResource1Record
Count=0|tableResource2RecordCount=0 |
| 10    | 2          | fileResourceLineCount=3|tableResource1Record
Count=0|tableResource2RecordCount=0 |
+-------+------------+-------+
```

**复杂数据类型示例**

如以下代码，定义了一个有三个 overloads 的 UDF，其中第一个用了 array 作为参数，第二个用了 map 作为参数，第三个用了 struct。由于第三个 overloads 用了 struct 作为参数或者返回值，因此要求必须要对 UDF class 打上 @Resolve annotation，来指定 struct 的具体类型。

```
@Resolve("struct<a:bigint>,string->string")
public class UdfArray extends UDF {
  public String evaluate(List<String> vals, Long len) {
    return vals.get(len.intValue());
  }
  public String evaluate(Map<String,String> map, String key) {
    return map.get(key);
  }
  public String evaluate(Struct struct, String key) {
    return struct.getFieldValue("a") + key;
  }
}
```

您可以直接将复杂类型传入 UDF 中，如下所示：

```
create function my_index as 'UdfArray' using 'myjar.jar';
```

```
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as
color_name from colors;
```

**HIVE UDF兼容示例**

MaxCompute 2.0支持了Hive风格的UDF，有部分的HIVE UDF、UDTF可以直接在MaxCompute上使用。

说明：

目前支持兼容的Hive版本为2.1.0; 对应Hadoop版本为2.7.2。UDF如果是在其他版本的Hive/Hadoop开发的，可能需要使用此Hive/Hadoop版本重新编译。

示例如下：

```
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.ObjectInsp
ectorFactory;
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";
  }
```

```
}
```

 说明：

对应HIVE UDF的使用请参考：

- *https://cwiki.apache.org/confluence/display/Hive/HivePlugins*
- *https://cwiki.apache.org/confluence/display/Hive/DeveloperGuide+UDTF*
- *https://cwiki.apache.org/confluence/display/Hive/GenericUDAFCaseStudy*

该udf可以将任意类型、数量的参数打包成array输出。假设输出jar包名为 test.jar：

```
--添加资源
Add jar test.jar;
--创建function
CREATE FUNCTION hive_collect as 'com.aliyun.odps.compiler.hive.Collect
' using 'test.jar';
--使用function
set odps.sql.hive.compatible=true;
select hive_collect(4y,5y,6y) from dual;
+------+
| _c0  |
+------+
| [4, 5, 6] |
+------+
```

 说明：

该udf可以支持所有的类型，包括array，map，struct等复杂类型。

使用兼容hive的udf需要注意：

- MaxCompute的add jar命令会永久地在project中创建一个resource，所以创建udf时需要指定jar包，无法自动将所有jar包加入classpath。
- 在使用兼容的HIVE UDF的时候，需要在sql前加set语句`set odps.sql.hive.compatible=true;`语句，set语句和sql语句一起提交执行。
- 在使用兼容的HIVE UDF时，还要注意MaxCompute的*JAVA沙箱*限制。

## 1.11.3 Python UDF

MaxCompute UDF包括UDF、UDAF和UDTF三种函数，本文将重点介绍如何通过Python实现这三种函数。

### 受限环境

MaxCompute UDF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，在这个环境中，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，您上传的代码必须都是纯Python实现，C扩展模块是被禁止的。

此外，Python的标准库中也不是所有模块都可用，涉及到上述功能的模块都会被禁止。具体标准库可用模块说明如下：

- 所有纯Python实现（不依赖扩展模块）的模块都可用。
- C实现的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

- _hashlib
- _heapq
- itertools
- _json
- _locale
- _lsprof
- math
- _md5
- _multibytecodec
- operator
- _random
- _sha256
- _sha512
- _sha
- _struct
- strop
- time
- unicodedata
- _weakref
- cPickle

- 部分模块功能受限。比如沙箱限制了您的代码最多可往标准输出和标准错误输出写出数据的大小，即`sys.stdout/sys.stderr`最多能写20Kb，多余的字符会被忽略。

第三方库

运行环境中还安装了除标准库以外比较常用的三方库，做为标准库的补充。支持的三方库还包括**numpy**。

说明：

三方库的使用同样受到禁止本地、网络IO或其他在受限环境下的限制，因此三方库中涉及到相关功能的API也是被禁止的。

### 参数与返回值类型

参数与返回值的指定方式，如下所示：

```
@odps.udf.annotate(signature)
```

Python UDF目前支持的MaxCompute SQL数据类型包括Bigint、String、Double、Boolean和Datetime。SQL语句在执行之前，必须确定所有函数的参数类型和返回值类型。因此对于Python这一动态类型语言，需要通过对UDF类加decorator的方式指定函数签名。

函数签名signature通过字符串指定，命令格式如下：

```
arg_type_list '->' type_list
arg_type_list: type_list | '*' | ''
type_list: [type_list ','] type
type: 'bigint' | 'string' | 'double' | 'boolean' | 'datetime'
```

- 箭头左边表示参数类型，右边表示返回值类型。
- 只有UDTF的返回值可以是多列，UDF和UDAF只能返回一列。
- *代表变长参数，使用变长参数，UDF/UDTF/UDAF可以匹配任意输入参数。

合法的signature示例如下：

```
'bigint,double->string'            # 参数为bigint、double，返回值为string

'bigint,boolean->string,datetime'  # UDTF参数为bigint、boolean，返回值为
string,datetime

'*->string'                        # 变长参数，输入参数任意，返回值为string

'->double'                         # 参数为空，返回值为double
```

Query语义解析阶段会检查到不符合函数签名的用法，抛出错误禁止执行。执行时，UDF函数的参数会以函数签名指定的类型传给您。您的返回值类型也要与函数签名指定的类型一致，否则检查到类型不匹配时也会报错。MaxCompute SQL数据类型对应Python类型如下：

| ODPS SQL Type | Bigint | String | Double | Boolean | Datetime |
|---|---|---|---|---|---|
| Python Type | int | str | float | bool | int |

说明：

- Datetime类型是以Int的形式传给用户代码的，值为epoch utc time起始至今的毫秒数。您可以通过Python标准库中的Datetime模块处理日期时间类型。

- Null值对应Python里的None。

此外，odps.udf.int(value,[silent=True])的参数也做了调整。增加了参数silent 。当silent为True时，如果value无法转为Int ，不会抛出异常，而是返回None 。

### UDF

实现Python UDF非常简单，只需要定义一个**new-style class**，并实现**evaluate**方法。示例如下：

```
from odps.udf import annotate

@annotate("bigint,bigint->bigint")
class MyPlus(object):

   def evaluate(self, arg0, arg1):
       if None in (arg0, arg1):
           return None
       return arg0 + arg1
```

**说明：**

Python UDF必须通过annotate指定函数签名。

### UDAF

- class odps.udf.BaseUDAF：继承此类实现Python UDAF。
- BaseUDAF.new_buffer()：实现此方法返回聚合函数的中间值的buffer。buffer必须是mutable object（比如list、dict），并且buffer的大小不应该随数据量递增，在极限情况下，buffer marshal过后的大小不应该超过2Mb。
- BaseUDAF.iterate(buffer[, args, ...])：实现此方法将args聚合到中间值buffer中。
- BaseUDAF.merge(buffer, pbuffer)：实现此方法将两个中间值buffer聚合到一起，即将pbuffer merge到buffer中。
- BaseUDAF.terminate(buffer)：实现此方法将中间值buffer转换为MaxCompute SQL的基本类型。

UDAF求平均值的示例如下：

```
@annotate('double->double')
class Average(BaseUDAF):

    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]
```

**UDTF**

- class odps.udf.BaseUDTF：Python UDTF的基类，用户继承此类，并实现process、close等方法。
- BaseUDTF.__init__()：初始化方法，继承类如果实现这个方法，则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，可以在这个方法中初始化所有状态。
- BaseUDTF.process([args, ...])：这个方法由MaxCompute SQL框架调用，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，便会输出一条记录。 forward的参数为SQL语句中指定的UDTF的输出参数。
- BaseUDTF.close()：UDTF的结束方法，此方法由MaxCompute SQL框架调用，并且只会被调用一次，即在处理完最后一条记录之后。

UDTF的示例如下：

```
#coding:utf-8
# explode.py

from odps.udf import annotate
from odps.udf import BaseUDTF


@annotate('string -> string')
class Explode(BaseUDTF):
   """将string按逗号分隔输出成多条记录
   """

   def process(self, arg):
       props = arg.split(',')
       for p in props:
           self.forward(p)
```

说明：

Python UDTF也可以不加annotate指定参数类型和返回值类型。这样，函数在SQL中使用时可以匹配任意输入参数，但返回值类型无法推导，所有输出参数都将认为是String类型。因此在调用forward时，就必须将所有输出值转成Str类型。

### 引用资源

Python UDF可以通过odps.distcache模块引用资源文件，目前支持引用文件资源和表资源。

- odps.distcache.get_cache_file(resource_name)：
  - 返回指定名字的资源内容。resource_name为Str类型，对应当前Project中已存在的资源名。如果资源名非法或者没有相应的资源，会抛出异常。
  - 返回值为file-like object ，在使用完这个object后，调用者有义务调用**close**方法释放打开的资源文件。

  使用get_cache_file的示例如下：

```
@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为Str类型，对应当前Project中已存在的资源表名。如果资源名非法或者没有相应的资源，会抛出异常。
  - 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tuple形式存在的表中的一条记录。

  使用get_cache_table的示例如下：

```
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)
```

## 1.11.4 MaxCompute UDF中运行Scipy

新版MaxCompute Isolation Session支持Python UDF。也就是说，Python UDF中已经可以运行二进制包。本文将为您介绍如何在MaxCompute UDF中运行Scipy。

**下载Scipy包并上传资源**

1. 从PyPI或其他镜像下载Scipy包。

   您需要下载后缀为cp27-cp27m-manylinux1_x86_64.whl的包，其他的包会无法加载，包括名为cp27-cp27mu的包，如下图中仅红框中的包可以直接使用。

| Filename, size & hash | File type | Python version | Upload date |
| --- | --- | --- | --- |
| scipy-1.1.0-cp27-cp27m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (16.8 MB) SHA256 | Wheel | cp27 | May 5, 2018 |
| scipy-1.1.0-cp27-cp27m-manylinux1_i686.whl (25.5 MB) SHA256 | Wheel | cp27 | May 5, 2018 |
| scipy-1.1.0-cp27-cp27m-manylinux1_x86_64.whl (30.8 MB) SHA256 | Wheel | cp27 | May 5, 2018 |
| scipy-1.1.0-cp27-cp27mu-manylinux1_i686.whl (25.5 MB) SHA256 | Wheel | cp27 | May 5, 2018 |
| scipy-1.1.0-cp27-cp27mu-manylinux1_x86_64.whl (30.8 MB) SHA256 | Wheel | cp27 | May 5, 2018 |

2. 下载后将文件名更改为scipy.zip，在MaxCompute Console中执行下述语句。

```
add archive scipy.zip;
```

此时scipy.zip即被创建为MaxCompute Archive资源。

> **说明：**
> 不建议您使用其他类型的资源，因为在执行时，MaxCompute会自动解压Archive类型的资源，从而省去手动解压的步骤。

### 从非Whl包生成Whl包

如果列出的包中包含Whl，则可以直接上传并跳过此步骤。如果列出的包不包含whl（例如仅有图中的scipy-0.19.0.zip），则需要在Linux环境中手动编译并打包为whl。打包前，需要确保下列命令返回cp27m而不是cp27mu。

```
python -c "import pip; print pip.pep425tags.get_abi_tag()"
```

如果返回值为cp27mu，您需要使用--enable-unicode=no选项编译一个可用的Python 2.7，再使用编译得到的Python。如果返回值正确，通常可以在该环境下使用pythonsetup.pybdist_wheel完成。

打包完成后，上传生成的whl包。

### 编写和创建UDF

1. 您需要编写一个UDF支持计算psi。

```
from odps.udf import annotate
from odps.distcache import get_cache_archive

def include_package_path(res_name):
    import os, sys
    archive_files = get_cache_archive(res_name)
    dir_names = sorted([os.path.dirname(os.path.normpath(f.name))
for f in archive_files
                        if '.dist_info' not in f.name], key=lambda v
: len(v))
    sys.path.append(os.path.dirname(dir_names[0]))

@annotate("double->double")
class MyPsi(object):
    def __init__(self):
        include_package_path('scipy.zip')

    def evaluate(self, arg0):
        from scipy.special import psi
        return float(psi(arg0))
```

get_cache_archive返回一个包含包中所有文件的文件对象。先取出所有的文件名，此后获得最短的路径作为包的路径，并加入sys.path。此后，便可以正常import scipy这个包。

> 说明：
> 因为MaxCompute会在执行前通过原有的沙箱检查UDF的输入/输出，因而include_package_path和import在函数外调用会报错。

2. 编写完成后，将代码保存为my_psi.py，并在MaxCompute Console中执行`addpymy_psi.py;`。

3. 在MaxCompute Console中执行下述命令创建函数。

```
create function my_psi as my_psi.MyPsi using my_psi.py,scipy.zip;
```

说明：

在创建函数时，不要忘记加上之前上传的包，例如上面的scipy.zip。

执行

创建UDF后，即可在MaxCompute Console中执行查询语句（暂不支持pypy，因此需禁用pypy）。

```
set odps.pypy.enabled=false;
set odps.isolation.session.enable = true;
select my_psi(sepal_length) from iris;
```

其他

如果包依赖了其他Python包，需要一同上传并同时加入到UDF依赖中。

使用0.7.4以上的PyODPS DataFrame可以简化使用二进制包的UDF的编写，无需手动调用include_package_path。

# 1.12 UDT

MaxCompute基于新一代的SQL引擎再推出新功能User Defined Type（简称UDT）。UDT允许在SQL中直接引用第三方语言的类或者对象，获取其数据内容或者调用其方法。

应用场景

UDT的常用场景如下：

- 场景1：某些用其他语言实现非常简单的功能，比如只需要调用一次Java内置类的方法即可实现的功能，而MaxCompute的内置函数却没有简单的方法实现。若使用UDF实现，整个过程又过于繁杂。
- 场景2：SQL中需要调用第三方库来实现相关功能。希望能够在SQL中直接调用，而不需要再wrap一层UDF。
- 场景3：Select Transform支持把脚本写到SQL语句中，可读性和代码维护大提升。但是某些语言无法这么用，比如Java源代码必须经过编译才能执行，希望类似这些语言也可直接写到SQL中。

**UDT概述**

MaxCompute中的UDT功能允许在SQL中直接引用第三方语言的类或者对象，获取其数据内容或者调用其方法。

在其他的SQL引擎中也有UDT的概念，但是和MaxCompute的概念有许多差异。很多SQL引擎中的概念比较像MaxCompute的struct复杂类型。而某些语言提供了调用第三方库的功能，如Oracle 的CREATE TYPE。相比之下，MaxCompute的UDT更像CREATE TYPE的概念，Type中不仅仅包含数据域，还包含方法。而且MaxCompute不需要用特殊的DDL语法来定义类型的映射，而是在SQL中直接使用。

示例如下：

```
set odps.sql.type.system.odps2=true;
-- 打开新类型，因为下面的操作会用到Integer，即int类型
SELECT java.lang.Integer.MAX_VALUE;
```

上述语句的输出结果如下：

```
+-----------+
| max_value |
+-----------+
| 2147483647 |
+-----------+
```

和Java语言一样，java.lang package可以省略，所以上述示例可以简写为：

```
set odps.sql.type.system.odps2=true;
SELECT Integer.MAX_VALUE;
```

可以看到，上述示例在select列表中直接写上了类似于Java表达式的表达式，而这个表达式按照Java的语义来执行，这个表达式表现出的能力就是MaxCompute的UDT。

UDT所提供的所有扩展能力，实际上用UDF都可以实现。如上述示例，若使用UDF实现，需要您进行下述操作。

1. 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;
   }
   }
   ```

2. 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';
```

3. 在SQL中使用。

```
select integer_max_value();
```

UDT简化了上述一系列的过程，让您能够轻松简单地用其他语言扩展SQL的功能。

### 实现原理

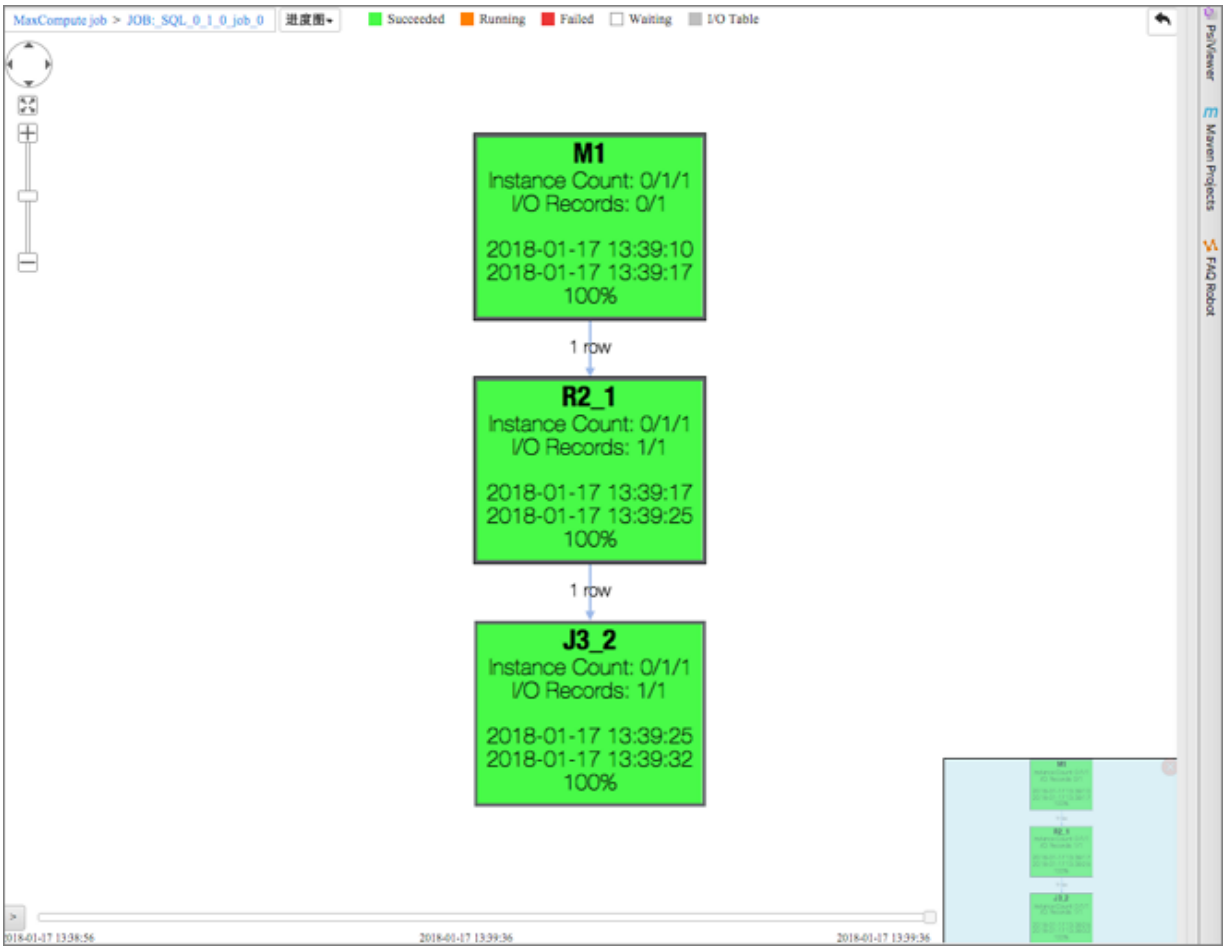
概述中的示例表现的是Java静态域访问的能力，而UDT的能力远不限于此。下述示例将为您介绍UDT的执行过程和具体功能。

```
-- 示例数据
@table1 := select * from values ('100000000000000000000') as t(x);
@table2 := select * from values (100L) as t(y);
-- 代码逻辑
@a := select new java.math.BigInteger(x) x from @table1;          --
new创建对象
@b := select java.math.BigInteger.valueOf(y) y from @table2;
 -- 静态方法调用
select /*+mapjoin(b)*/ x.add(y).toString() from @a a join @b b;
 -- 实例方法调用
```

输出结果如下：

```
100000000000000000100
```

上述示例还表现了一种用UDF比较不好实现的功能：子查询的结果允许UDT类型的列。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中再调用其他方法，甚至能参与数据shuffle。




如上图可知，该UDT共有三个STAGE：M1、R2和J3。如果您熟悉MapReduce原理便会知道，由于join的存在需要做数据reshuffle，所以会出现多个stage。一般情况下，不同stage不仅是在不同进程，甚至是在不同物理机器上运行的。

查看M1，会发现M1仅仅执行了new java.math.BigInteger(x) 这个操作。

查看J3，可以看到J3在不同阶段执行了java.math.BigInteger.valueOf(y)和x.add(y).toString()的操作。这几个操作不仅分阶段执行，而且在不同进程、不同物理机器上执行。但是UDT把这个过程封装起来，看起来和在同一个JVM中执行的效果几乎一样。

详细功能说明

- 目前UDT仅支持了Java语言，后续版本会加入对其他语言的支持。
- UDT同样允许用户上传自己的Jar包，并且直接引用。当前提供了一些flag以方便使用。
  - set odps.sql.session.resources指定引用的资源，可以指定多个，用逗号隔开。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。

如UDT概述中UDF的Jar包，用UDT来使用：

```
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一定要事先上传到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，如`java.math.BigInteger`，也可以使用`*`。暂不支持`static import`。

  如UDT概述中UDF的Jar包，用UDT来使用还可有如下写法：

```
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支持的操作如下所示：
  - 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，如new Integer[] { 1, 2, 3 }。
  - 方法调用，包括静态方法调用（因此能用工厂方法构建对象）。
  - 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。
- UDT支持类型转换，支持SQL的风格，如cast(1 as java.lang.Object)。但不支持Java风格的类型转换，如(Object)1。
- 暂不支持匿名类和lambda表达式（后续版本可能会支持）。
- 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）。

- Java SDK的类都是默认可用的。

说明：

目前runtime使用的JDK版本是JDK1.8，比该版本更新的JDK功能可能不支持。

- 所有的运算符都是MaxCompute SQL的语义，不是UDT的语义。如`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）。

  除string的相加操作比较容易混淆外，另一个比较容易混淆的是=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，通过等号判断的相等无法保证其行为（在UDT场景下，同一对象的概念是不能保证的，详情请参见下文的说明）。
- 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，这个映射关系在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，如`'123'.length() , 1L.hashCode()`。
  - UDT类型能够直接参与内置函数或者UDF的运算， 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设置`set odps.sql.type.system.odps2=true;`方可使用，否则会报错。

- UDT对泛型有比较完整的支持，如`java.util.Arrays.asList(new java.math.BigInteger('1'))`，编译器能够根据参数类型知道该方法的返回值是`java.util.List<java.math.BigInteger>`类型。

说明：

构造函数需要指定类型参数，否则使用java.lang.Object，这一点和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。

- UDT对同一对象的概念是模糊的。这是由数据的reshuffle导致的。从第一部分join的示例可以看出，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的两个引用后面可能分别引用了不同的对象（比如对象先被shuffle到两台机器，然后下次又shuffle回一起）。

  所以在使用UDT时，应该避免使用=operator来判断相等，而是使用equals方法。

  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。不能保证的是不同行或者不同列的对象的数据相关性。
- 目前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。

  并不是说UDT不能用在这些结构中，UDT可以在expression中间的任意阶段使用，只是不能作为最终输出。比如您虽然不能`group by new java.math.BigInteger('123')`，但是可以`group by new java.math.BigInteger('123').hashCode()`。因为hashCode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扩展了类型转换规则：
  - UDT对象能够被隐式类型转换为其基类对象。
  - UDT对象能够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，注意这时候可能会导致内容变化，比如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，而这个过程会真的导致数据内容的变化，甚至可能会有精度损失。
- 目前UDT对象不能落盘，这意味着不能将UDT对象insert到表中（实际上DDL不支持UDT，创建不出这样的表），当然，隐式类型转换变成了内置类型的除外。同时，屏显的最终结果也不能是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的时候，可以用这种方法来观察UDT的内容。

  您也可以设置`set odps.sql.udt.display.tostring=true;`这样MaxCompute会自动帮用户把所有的以UDT为最终输出的列wrap上`java.util.Objects.toString(...)`，从而方便调试。这个flag只对屏显语句生效，对insert语句不生效，所以专门用在调试中。

  内置类型支持binary，因此支持自己实现serialize的过程，将byte[]的数据落盘。下次读出来的时候再还原回来。

  某些类可能自带序列化和反序列化的方法，如protobuffer。目前UDT依旧不支持落盘，还是需要用户自己调用序列化反序列化方法，变成binary数据类型来落盘。

- UDT不仅能够实现scalar函数的功能，配合着内置函数*collect list*和*explode*，完全能够实现aggregator和table function的功能。

**UDT示例**

- 使用Java数组示例

```
set odps.sql.type.system.odps2=true;
set odps.sql.udt.display.tostring=true;
select
    new Integer[10],    -- 创建一个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);    -- 这个创建了一个 List<
Integer>，这个也能当做array<int>来用，所以这是另一个创建内置array数据的方法
from values (1,2,3,4) as t(c1, c2, c3, c4);
```

- 使用JSON示例

  UDT的runtime自带一个JSON的依赖（2.2.4），因此可以直接使用JSON。

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*,java,com.google.gson
.*; -- 同时import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，直接反序列化出来是List<Object>类
型，所以这里强转成了 List<String>，方便后续使用。
from @a;
```

  相比于内置函数*get_json_object*，上述用法不仅使用方便，在需要对Json字符串多个部分做内容提取时，先将JSON字符串反序列成格式化数据，大大提升工作效率。

  除JSON外，MaxCompute runtime自带的依赖还包括：commons-logging（1.1.1），commons-lang（2.5），commons-io（2.4），protobuf-java（2.4.1）。

- 复杂类型操作示例

  内置类型array和map与java.util.List和java.util.Map存在映射关系。结果如下：

  - Java中实现了java.util.List或者java.util.Map接口的类的对象，都可参与MaxComputeSQL的复杂类型操作。

- MaxComput中 array，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，但是别忘了array是支持的
    Objects.toString(a),        -- 过去不支持将array类型cast成string，现
在有绕过方法了
    array(1,2,3).subList(1, 2)             -- 求subList
from (select new ArrayList<Integer>(array(1,2,3)) as al, array(1,2,3
) as a) t;
```

- 聚合操作的实现示例

  UDT实现聚合的原理是，先用内置函数*collect_set*或*collect_list*函数将数据转变成List，之后对该List应用UDT的标量方法求得这一组数据的聚合值。

  例如下述示例实现对BigInteger求中位数（由于数据是java.math.BigInteger类型的，所以不能直接用内置的*median*函数）。

```
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
;  -- 求中位数的逻辑，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])).divide(new
 BigDecimal(2))) med from @b;
-- 最终结果
select med.toString() from @c;
```

  由于collect_list会先把所有数据都收集到一块，是没有办法实现partial aggregate的，所以这个做法的效率会比内置的aggregator或者UDAF低，所以在内置aggregator能实现的情况下，应尽量使用内置的aggregator。同时把一个group的所有数据都收集到一起的做法，会增加数据倾斜的风险。

  但是另一方面，如果UDAF本身的逻辑就是要将所有数据收集到一块（比如类似内置函数wm_concat的功能），此时使用上述方法，反而可能比UDAF（注意不是内置aggregator）高。

- 表值函数的实现示例

表值函数允许输入多行多列数据，输出多行多列数据。可以按照下述原理实现：

1. 对于输入多行多列数据，可以参考聚合函数实现的示例。
2. 要实现多行的输出，可以让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。
3. UDT本身就可以包含多个数据域，通过调用不同的getter方法来获取各个域的内容即可展开成多列。

下述示例实现将一个json字符串的内容展开出来的功能。

```
@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打成多行
@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)
```

**功能/性能/安全性**

UDT在功能方面的优势如下：

- 使用极其简单，不需要定义任何function。
- JDK的所有功能都可以拿来用，大大扩展了SQL的能力。
- 代码直接和SQL放在一块，便于管理。
- 其它类库拿来即用，代码重用率高。
- 可以使用面向对象的思想设计某些功能。

后续待完善功能如下：

- 支持无返回值的函数调用（或者有返回值，但是忽略返回值，直接取操作数本身，如调用List的add方法，结束后返回执行完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，如python。

性能方面，UDT执行过程和UDF非常接近，其性能与UDF几乎是一致的，而且计算引擎做了很多优化，在某些场景下UDT的性能更高。

- 对象在一个进程内实际上是不需要做列化反序列化的，只有跨进程的时候才需要。简单地说，就是在没有join或者aggregator等需要做数据reshuffle的情况下，UDT并没有序列化反序列化的开销。
- UDT的Runtime实现是基于codegen，而不是反射，所以不会存在反射带来的性能损失连续的多个UDT的操作，实际上会合并在一起，在一个FunctionCall里一起执行，如前面例子中`values[x].add(values[y]).divide(java.math.BigInteger.valueOf(2))` 这个看似存在多次UDT方法调用的操作，实际上只有一次调用。所以虽然UDT操作的单元都比较小，但是并不会因此造成多次函数调用的接口上的额外开销。

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。所以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。

# 1.13 UDJ

**UDJ简介**

目前MaxCompute内置了多种*Join*操作，包括inner/right join、 outer/left jion、outer/full join、outer/semi/anti-semi join等。这些内置的join操作功能强大，能够满足很大一部分需求，但是其标准的join实现，无法满足很多跨表操作的需求。

很多情况下，可以通过UDF（User Defined Function）描述您的代码框架，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。甚至在一些多表操作的场景上，您不得不放弃SQL而转向传统的完全自定义MR，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MR门槛都比较高，同时还会带来一些问题：

- 对于计算平台而言，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”，这点不利于生成最优的执行计划；
- 使用MR，不仅更大程度上剥夺了系统进行执行优化的可能性，而且由于MR绝大部分代码由Java完成，在执行效率上会远低于MaxCompute基于LLVM 代码生成器产生的深度优化native运行时。

目前基于MaxCompute 2.0计算引擎，MaxCompute在UDF框架中新近引入的一种新扩展机制：UDJ(User Defined Join)，来实现灵活的跨表、多表自定义操作，同时减少不得不通过MR等方式对分布式系统底层细节的操作。这是MaxCompute基于新一代体系机构发展NewSQL数据处理框架的重要一步。

**UDJ样例定义**

接下来用一个样例来详细介绍MaxCompute的UDJ使用。两个日志表，分别是payment和user_client_log。

- payment (user_id string,time datetime,pay_info string)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。
- user_client_log(user_id string,time datetime,content string)表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。

需求：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。

样例数据如下：

payment

| 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

| 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 |

其中user_client_log的一条记录

| user_id | time | content |
| --- | --- | --- |
| 8881237 | 2018-02-13 00:30:00 | click MpkvilgWSmhUuPn |

和payment中时间最接近的一条是

| user_id | time | pay_info |
| --- | --- | --- |
| 8881237 | 2018-02-13 08:30:00 | pYvotuLDIT |

因此，这两条记录合并为：

| | | |
|---|---|---|
| 8881237 | 2018-02-13 00:30:00 | click MpkvilgWSmhUuPn, pay pYvotuLDIT |

上面样例数据全部合并的结果如下：

| 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 |

### 使用内置JOIN

若使用标准join解决这个需求，会发现这个需求除了需要按照user_id进行关联以外，还需要知道payment中哪一条记录和user_client_log中的记录的时间差异值最小，勉强写SQL伪代码则类似于：

```
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，而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。

但在分布式系统中，Join操作实际上是将两个表按照某个(或多个)字段进行分组，并将同组数据集中到一处，而标准SQL中的Join对于关联后的操作有限，此时，若能提供一个通用编程语言接口，使用过程通过插件式的方式实现这个接口，在这个接口中将关联后的分组数据进行自定义处理并输出既可解决这个问题，这也是UDJ要解决的问题。

### 使用Java编写UDJ代码

接下来详细介绍如何通过UDJ实现这个内置join无法实现的需求。

前置条件

这是一个新功能，我们需要一个比较新的SDK：

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.30.0</version>
  <scope>provided</scope>
</dependency>
```

新的SDK中包含了一个新的抽象类UDJ，我们通过继承这个类，来实现UDJ的功能：

```
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;
/** For each record of right table, find the nearest record of left
table and
 * merge two records.
 */
@Resolve("->string,bigint,string")
public class PayUserLogMergeJoin extends UDJ {
  private Record outputRecord;
  /** Will be called prior to the data processing phase. User could
implement
   * this method to do initialization work.
   */
  @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)
    });
  }
  /** Override this method to implement join logic.
   * @param key Current join key
   * @param left Group of records of left table corresponding to the
current key
   * @param right Group of records of right table corresponding to the
 current key
   * @param output Used to output the result of UDJ
   */
  @Override
  public void join(Record key, Iterator<Record> left, Iterator<Record
> right, Yieldable<Record> output) {
    outputRecord.setString(0, key.getString(0));
    if (!right.hasNext()) {
      // Empty right group, do nothing.
      return;
    } else if (!left.hasNext()) {
```

```
      // Empty left group. Output all records of right group without
merge.
      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<>();
    // The left group of records will be iterated from the start to
the end
    // for each record of right group, but the iterator cannot be
reset.
    // So we save every records of left to an 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;
      // Iterate through all records of left, and find the pay record
that has
      // the minimal difference in terms of time.
      for (Record pay: pays) {
        long delta = Math.abs(logTime - pay.getDatetime(0).getTime());
        if (delta < minDelta) {
          minDelta = delta;
          nearestPay = pay;
        }
      }
      // Merge the log record with nearest pay record and output to
the result.
      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() {
  }
}
```

说明：

在本例中没有处理记录中的NULL值，为了使程序简洁便于演示，这里假设数据中没有NULL值。

从代码中可以看到，在每次调用UDJ的join方法时，两张表中各有一组对应着同一个key数据，提供给了我们。因此，只需遍历右表(user_client_log)的分组，对于每一个log记录，遍历一遍左表(payment)的分组，找出时间相差最小的记录，将日志内容合并然后输出即可。

这里假设同一个用户的支付记录数比较少，可以预先将左表分组全部加载到内存(通常情况下，内存足以存放一个用户在一天内产生的支付数据)。但若这个假设不成立应该如何解决？文章后面一节“使用SORT BY预排序”会解决这个问题。

### 在MaxCompute中创建UDJ

编写完UDJ的Java代码后，需要将这部分代码插件式的嵌入到MaxCompute SQL中进行执行。再此之前，需要将代码注册到MaxCompute。假设上述代码打包成了odps-udj-example.jar。

通过add jar命令将其当做jar资源上传到MaxCompute:

```
add jar odps-udj-example.jar;
```

通过create function语句注册UDJ函数，指定UDJ在SQL中的函数名pay_user_log_merge_join，以及关联上它对应的jar资源odps-udj-example.jar和在jar包中的类名 `com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin:`

```
create function pay_user_log_merge_join
  as 'com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin'
  using 'odps-udj-example.jar';
```

### 使用MaxCompute SQL进行UDJ查询

UDJ注册好了以后，就可以在MaxCompute SQL中使用了。

1. 创建示例源表：

```
create table payment (user_id string,time datetime,pay_info string);
create table user_client_log(user_id string,time datetime,content
string);
```

2. 制造演示数据：

```
--制造payment表数据
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')
;
--制造user_client_log表数据
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')
;
```

3. 在MaxCompute SQL中使用刚刚创建好的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)
;
```

UDJ的语法与标准Join语法类似，主要差异多了using字句：

- 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中分别用到的左右表的列；
- r是UDJ结果的别名，用于其他地方引用UDJ的结果；
- (user_id, time, content)是UDJ产生的结果的列名。

运行上面这条SQL，结果为：

```
+---------+------------+---------+
| 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 |
+---------+------------+---------+
```

如上述，通过使用UDJ完成了内置jion无法轻松完成的需求。

**UDJ预排序功能**

前面的UDJ代码抛出过这样的问题：为了找到payment中相差最小的一条记录，需要反复对payment表的iterator进行遍历，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，这方式可用，但在其它场景中，有时候同组内的数据可能非常大，大到无法在内存中放下，此时就需要通过其他方式解决。本小节将介绍通过SORT BY预排序解决这个场景。

当某个用户的支付数据量非常巨大导致无法将payment放在内存中时，仔细分析会发现组内所有数据如果已经按照time排好了序，那么这个问题就好解了，只需要比较两边iterator最”顶端”的数据，就可以实现这个功能。

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) {
      // The delta of time between two records is decreasing, we can
still
      // explore the left group to try to gain a smaller delta.
      lastPayRecord = payRecord.clone();
      prevDelta = delta;
      payRecord = left.next();
    } else {
      // Hit to the point of minimal delta. Check with the last pay
record,
      // output the merge result and prepare to process the next
record of
      // right group.
      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;
      }
    }
  }
}
```

SQL语句中，只需要对前面的例子稍作修改，在UDJ语句尾部增加SORT BY子句，指定UDJ组内左右表分别都按照各自的time字段进行排序（注意：UDJ代码修改后需要更新UDJ对应的jar包）：

```
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)
SORT BY p.time, u.time
;
```

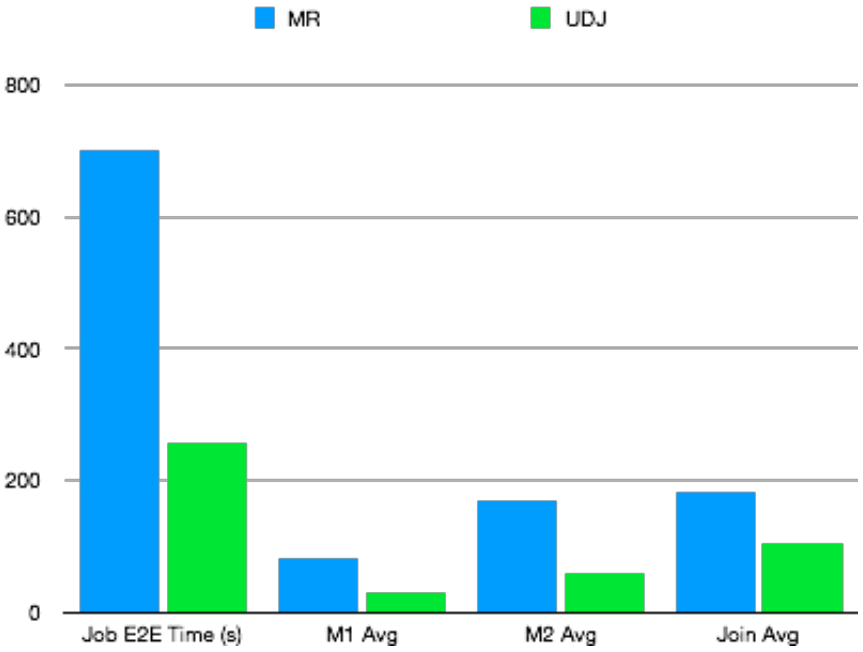
执行结果将和前面的方式执行结果一样。

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。

**UDJ的性能**

目前集团内在面对复杂的跨表计算需求时，因为缺乏UDJ这种机制，不得不借助MR来实现整套跨表计算的分布式流程。这类场景业务复杂的BU（比如搜索，广告等）尤为常见。

在此用一个线上线上真实的MR作业进行测试验证UDJ的性能。该MR作业实现一套比较复杂的算法将两个表合并一起，用UDJ对该MR进行改写，并且验证新的UDJ实现结果的正确性。性能方面，在并发度相同的情况下两者性能对比如下：




由图可见，UDJ接口的引入，一方面更加方便的描述对多表数据进行操作的复杂逻辑，一方面大幅度提高了性能（代码只有在UDJ内被调用，其上下游的逻辑（比如这个例子中的整个mapper逻辑）则完全通过MaxCompute高效的native运行时完成）。即便是在Java代码中，由于UDJ对MaxCompute运行时引擎和Java接口之间的数据交互逻辑做了深度的优化，通过UDJ实现的join逻辑，也比其对等的reducer更高效。

# 2 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MR Main则不受此限制），具体限制如下所示。

- 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。
  - 读取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访问到表的记录。
- 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，都会被禁止。
- Java反射限制：suppressAccessChecks权限被禁止，无法setAccessible某个private的属性或方法，以达到读取private属性或调用private方法的目的。

在代码中，直接使用下文访问本地文件的方法会报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，如下所示：

```
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()
```

# 3 SDK

## 3.1 Java SDK

本文将为您介绍较为常用的 MaxCompute 核心接口，更多详情请参见 *SDK Java Doc*。

您可以通过 Maven 管理配置新 SDK 的版本。Maven 的配置信息如下（最新版本可以随时到 *search.maven.org* 搜索 odps-sdk-core 获取）：

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-core</artifactId>
  <version>0.26.2-public</version>
</dependency>
```

MaxCompute 提供的 SDK 包整体信息，如下表所示：

| 包名 | 描述 |
| --- | --- |
| odps-sdk-core | MaxCompute 的基础功能，例如：对表，Project 的操作，以及 Tunnel 均在此包中 |
| odps-sdk-commons | 一些 Util 封装 |
| odps-sdk-udf | UDF 功能的主体接口 |
| odps-sdk-mapred | MapReduce 功能 |
| odps-sdk-graph | Graph Java SDK，搜索关键词“odps-sdk-graph” |

**AliyunAccount**

阿里云认证账号。输入参数为 accessId 及 accessKey，是阿里云用户的身份标识和认证密钥。此类用来初始化 MaxCompute。

**MaxCompute**

MaxCompute SDK 的入口，您可通过此类来获取项目空间下的所有对象集合，包括：*Projects*，*Tables*，*Resources*，*Functions*，*Instances*。

说明：

MaxCompute 原名 ODPS，因此在现有的 SDK 版本中，入口类仍命名为 ODPS。

您可以通过传入 AliyunAccount 实例来构造 MaxCompute 对象。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Table t : odps.tables()) {
        ....
    }
```

**Projects**

Projects 是 MaxCompute 中所有项目空间的集合。集合中的元素为 Project。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Project p = odps.projects().get("my_exists");
    p.reload();
    Map<String, String> properties = prj.getProperties();
    ...
```

**Project**

Project 是对项目空间信息的描述，可以通过 Projects 获取相应的项目空间。

**SQLTask**

SQLTask 是用于运行、处理 SQL 任务的接口。可以通过 run 接口直接运行 SQL。( 注意：每次只能提交运行一个**SQL**语句。)

run 接口返回 Instance 实例，通过 Instance 获取 SQL 的运行状态及运行结果。程序示例如下：

```
    import java.util.List;
    import com.aliyun.odps.Instance;
    import com.aliyun.odps.Odps;
    import com.aliyun.odps.OdpsException;
    import com.aliyun.odps.account.Account;
    import com.aliyun.odps.account.AliyunAccount;
    import com.aliyun.odps.data.Record;
    import com.aliyun.odps.task.SQLTask;
    public class testSql {
    private static final String accessId = "";
    private static final String accessKey = "";
    private static final String endPoint = "http://service.odps.aliyun
.com/api";
    private static final String project = "";
    private static final String sql = "select category from iris;";
    public static void
    main(String[] args) {
      Account account = new AliyunAccount(accessId, accessKey);
       Odps odps = new Odps(account);
```

```
        odps.setEndpoint(endPoint);
        odps.setDefaultProject(project);
        Instance i;
       try {
          i = SQLTask.run(odps, sql);
          i.waitForSuccess();
          List<Record> records = SQLTask.getResult(i);
          for(Record r:records){
             System.out.println(r.get(0).toString());
          }
       } catch (OdpsException e) {
          e.printStackTrace();
       }
    }
  }
```

说明：

如果您想创建表，需要通过 SQLTask 接口，而不是 Table 接口。您需要将表操作的语句传入SQLTask。

### Instances

Instances 是 MaxCompute 中所有实例（Instance）的集合，集合中的元素为 Instance。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Instance i : odps.instances()) {
        ....
    }
```

### Instance

Instance 是对实例信息的描述，可以通过 Instances 获取相应的实例。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Instance ins = odps.instances().get("instance id");
    Date startTime = instance.getStartTime();
    Date endTime = instance.getEndTime();
    ...
    Status instanceStatus = instance.getStatus();
    String instanceStatusStr = null;
    if (instanceStatus == Status.TERMINATED) {
      instanceStatusStr = TaskStatus.Status.SUCCESS.toString();
      Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
      for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
```

```
        if (status.getValue().getStatus() != TaskStatus.Status.SUCCESS
) {
          instanceStatusStr = status.getValue().getStatus().toString
();
          break;
        }
      }
    } else {
      instanceStatusStr = instanceStatus.toString();
    }
    ...
    TaskSummary summary = instance.getTaskSummary("instance name");
    String s = summary.getSummaryText();
```

**Tables**

Tables 是 MaxCompute 中所有表的集合，集合中的元素为 Table。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Table t : odps.tables()) {
        ....
    }
```

**Table**

Table 是对表信息的描述，可以通过 Tables 获取相应的表。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Table t = odps.tables().get("table name");
    t.reload();
    Partition part = t.getPartition(new PartitionSpec(tableSpec[1]));
    part.reload();
    ...
```

**Resources**

Resources 是 MaxCompute 中所有资源的集合。集合中的元素为 Resource。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Resource r : odps.resources()) {
        ....
```

```
    }
```

### Resource

Resource 是对资源信息的描述，可以通过 Resources 获取相应的资源。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Resource r = odps.resources().get("resource name");
    r.reload();
    if (r.getType() == Resource.Type.TABLE) {
     TableResource tr = new TableResource(r);
     String tableSource = tr.getSourceTable().getProject() + "."
         + tr.getSourceTable().getName();
     if (tr.getSourceTablePartition() != null) {
       tableSource += " partition(" + tr.getSourceTablePartition().
toString()
           + ")";
     }
     ....
    }
```

创建文件资源的示例，如下所示：

```
    String projectName = "my_porject";
    String source = "my_local_file.txt";
    File file = new File(source);
    InputStream is = new FileInputStream(file);
    FileResource resource = new FileResource();
    String name = file.getName();
    resource.setName(name);
    odps.resources().create(projectName, resource, is);
```

创建表资源的示例，如下所示：

```
    TableResource resource = new TableResource(tableName, tablePrj,
partitionSpec);
    //resource.setName(INVALID_USER_TABLE);
    resource.setName("table_resource_name");
    odps.resources().update(projectName, resource);
```

### Functions

Functions 是 MaxCompute 中所有函数的集合。集合中的元素为 Function。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Function f : odps.functions()) {
        ....
```

```
    }
```

**Function**

Function 是对函数信息的描述，可以通过 Functions 获取相应的函数。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Function f = odps.functions().get("function name");
    List<Resource> resources = f.getResources();
```

创建函数的示例，如下所示：

```
    String resources = "xxx:xxx";
    String classType = "com.aliyun.odps.mapred.open.example.WordCount
";
    ArrayList<String> resourceList = new ArrayList<String>();
    for (String r : resources.split(":")) {
      resourceList.add(r);
    }
    Function func = new Function();
    func.setName(name);
    func.setClassType(classType);
    func.setResources(resourceList);
    odps.functions().create(projectName, func);
```

# 3.2 Python SDK

PyODPS是MaxCompute的Python版本的SDK，它提供了对MaxCompute对象的基本操作和DataFrame框架，可以轻松地在MaxCompute上进行数据分析。更多详情请参见*Github*项目和包括所有接口、类的细节等内容的详细文档。

- 关于PyODPS的更多详情请参见*PyODPS*云栖社区专辑。
- 如果您想了解更多关于PyODPS的开发内容，请参见*PyODPS*开发文档。
- 欢迎各位开发者参与到PyODPS的生态开发中，详情请参见*GitHub*文档。
- 欢迎提交issue和merge request，加快PyODPS生态成长，更多详情请参见代码。
- 钉钉技术交流群：**11701793**

**安装**

PyODPS支持Python2.6以上（包括Python3），系统安装pip后，只需运行`pip install pyodps`，PyODPS的相关依赖便会自动安装。

### 快速开始

首先，用阿里云账号初始化一个MaxCompute的入口，如下所示：

```
from odps import ODPS
odps = ODPS('**your-access-id**', '**your-secret-access-key**', '**
your-default-project**',
            endpoint='**your-end-point**')
```

根据上述操作初始化后，便可对表、资源、函数等进行操作。

### 项目空间

项目空间是MaxCompute的基本组织单元，类似于Database的概念。

您可通过`get_project`获取到某个项目空间，如下所示：

```
project = odps.get_project('my_project')  # 取到某个项目
project = odps.get_project()              # 取到默认项目
```

说明：

- 如果不提供参数，则获取到默认项目空间。
- 通过`exist_project`，可以查看某个项目空间是否存在。
- 表是MaxCompute的数据存储单元。

### 表操作

通过调用`list_tables`可以列出项目空间下的所有表，如下所示：

```
for table in odps.list_tables():
    # 处理每张表
```

通过调用`exist_table`可以判断表是否存在，通过调用`get_table`可以获取表。

```
>>> t = odps.get_table('dual')
>>> 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.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>]
```

### 创建表的Schema

初始化的方法有两种，如下所示：

- 通过表的列和可选的分区来初始化。

```
>>> from odps.models import Schema, Column, Partition
>>> columns = [Column(name='num', type='bigint', comment='the column
')]
>>> partitions = [Partition(name='pt', type='string', comment='the
partition')]
>>> schema = Schema(columns=columns, partitions=partitions)
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

- 通过调用Schema.from_lists，虽然调用更加方便，但显然无法直接设置列和分区的注释。

```
>>> schema = Schema.from_lists(['num'], ['bigint'], ['pt'], ['string
'])
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

### 创建表

您可以使用表的Schema来创建表，操作如下所示：

```
>>> table = odps.create_table('my_new_table', schema)
>>> table = odps.create_table('my_new_table', schema, if_not_exists=
True)  # 只有不存在表时才创建
>>> table = o.create_table('my_new_table', schema, lifecycle=7)  # 设置
生命周期
```

也可以使用逗号连接的字段名 字段类型字符串组合来创建表，操作如下所示：

```
>>> # 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)
```

在未经设置的情况下，创建表时，只允许使用bigint、double、decimal、string、datetime、boolean、map和array类型。

如果您的服务位于公共云，或者支持tinyint、struct等新类型，可以设置`options.sql.use_odps2_extension = True`，以打开这些类型的支持，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>')
```

### 获取表数据

您可通过以下三种方法获取表数据。

- 通过调用`head`获取表数据，但仅限于查看每张表开始的小于1万条的数据，如下所示：

```
>>> t = odps.get_table('dual')
>>> for record in t.head(3):
>>>     print(record[0])  # 取第0个位置的值
>>>     print(record['c_double_a'])  # 通过字段取值
>>>     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
```

- 通过在table上执行`open_reader`操作，打开一个reader来读取数据。您可以使用with表达式，也可以不使用。

```
# 使用with表达式
>>> 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，这里可以改造成并行操作
>>>         # 处理一条记录
>>>
>>> # 不使用with表达式
>>> reader = t.open_reader(partition='pt=test')
>>> count = reader.count
>>> for record in reader[5:10]
>>>     # 处理一条记录
```

- 通过使用Tunnel API读取表数据，`open_reader`操作其实也是对Tunnel API的封装。

### 写入数据

类似于`open_reader`，table对象同样可以执行`open_writer`来打开writer，并写数据。如下所示：

```
>>> # 使用 with 表达式
>>> with t.open_writer(partition='pt=test') as writer:
>>>     writer.write(records)  # 这里records可以是任意可迭代的records，默认写到block 0
>>>
>>> with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer:
 # 这里同是打开两个block
>>>     writer.write(0, gen_records(block=0))
>>>     writer.write(1, gen_records(block=1))  # 这里两个写操作可以多线程并行，各个block间是独立的
>>>
>>> # 不使用 with 表达式
>>> writer = t.open_writer(partition='pt=test', blocks=[0, 1])
>>> writer.write(0, gen_records(block=0))
>>> writer.write(1, gen_records(block=1))
>>> writer.close()  # 不要忘记关闭 writer，否则数据可能写入不完全
```

同样，向表中写入数据也是对Tunnel API的封装，更多详情请参见数据上传下载通道。

### 删除表

删除表的操作，如下所示：

```
>>> odps.delete_table('my_table_name', if_exists=True)  #  只有表存在时删除
>>> t.drop()  # Table对象存在的时候可以直接执行drop函数
```

### 表分区

- 基本操作

  遍历表的全部分区，如下所示：

  ```
  >>> 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，如下所示：

  ```
  >>> table.exist_partition('pt=test,sub=2015')
  ```

  获取分区，如下所示：

  ```
  >>> partition = table.get_partition('pt=test')
  >>> print(partition.creation_time)
  ```

```
2015-11-18 22:22:27
>>> partition.size
0
```

- 创建分区

```
>>> t.create_partition('pt=test', if_not_exists=True) # 不存在的时候才创建
```

- 删除分区

```
>>> t.delete_partition('pt=test', if_exists=True)  # 存在的时候才删除
>>> partition.drop()  # Partition对象存在的时候直接drop
```

### SQL

PyODPS支持MaxCompute SQL的查询，并可以读取执行的结果。

- 执行SQL

```
>>> odps.execute_sql('select * from dual')  #  同步的方式执行，会阻塞直到SQL执行完成
>>> instance = odps.run_sql('select * from dual')  # 异步的方式执行
>>> instance.wait_for_success()  # 阻塞直到完成
```

- 读取SQL执行结果

  运行SQL的instance能够直接执行`open_reader`的操作，一种情况是SQL返回了结构化的数据。

```
>>> with odps.execute_sql('select * from dual').open_reader() as reader:
>>>     for record in reader:
>>>         # 处理每一个record
```

另一种情况是SQL可能执行的比如`desc`，这时通过`reader.raw`属性取到原始的SQL执行结果。

```
>>> with odps.execute_sql('desc dual').open_reader() as reader:
>>>     print(reader.raw)
```

### Resource

资源在MaxCompute上常用在UDF和MapReduce中。

列出所有资源还是可以使用`list_resources`，判断资源是否存在使用`exist_resource`。删除资源时，可以调用`delete_resource`，或者直接对于Resource对象调用`drop`方法。

在PyODPS中，主要支持两种资源类型，一种是文件，另一种是表。

- 文件资源

文件资源包括基础的file类型、以及py、jar和archive。

> 说明：
> 在DataWorks中，py格式的文件资源请以file形式上传，详情请参见*Python UDF文档*。

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、以及一个file-like的对象（或者是字符串对象）来创建，示例如下：

```
resource = odps.create_resource('test_file_resource', 'file',
file_obj=open('/to/path/file'))  # 使用file-like的对象
resource = odps.create_resource('test_py_resource', 'py', file_obj='
import this')  # 使用字符串
```

读取和修改文件资源

对文件资源调用open方法，或者在MaxCompute入口调用open_resource都能打开一个资源，打开后的对象会是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
>>>     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 odps.open_resource('test_file_resource', mode='r+') as fp:
 # 读写模式打开
>>>     fp.read()
>>>     fp.tell()  # 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，文件内容会被先清空。
- a：追加模式，只能写入内容到文件末尾。
- r+：读写模式，能任意读写内容。
- 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，打开如说一些压缩文件等等就需要以这种模式， 因此`rb`就是指以二进制读模式打开文件，`r+b`是指以二进制读写模式打开。

- 表资源

  创建表资源

```
>>> odps.create_resource('test_table_resource', 'table', table_name
='my_table', partition='pt=test')
```

  更新表资源

```
>>> table_resource = odps.get_resource('test_table_resource')
>>> table_resource.update(partition='pt=test2', project_name='
my_project2')
```

**DataFrame**

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。完整的DataFrame文档请参见*DataFrame*。

DataFrame的示例如下：

**说明：**

在所有步骤开始前，需要创建MaxCompute对象。

```
>>> o = ODPS('**your-access-id**', '**your-secret-access-key**',
             project='**your-project**', endpoint='**your-end-point
**'))
```

此处以movielens 100K作为示例，假设已经有三张表，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）。

只需传入Table对象，便可创建一个DataFrame对象。如下所示：

```
>>> from odps.df import DataFrame
```

```
>>> users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

通过dtypes属性来查看这个DataFrame有哪些字段，分别是什么类型，如下所示：

```
>>> users.dtypes
```

通过head方法，可以获取前N条数据，方便快速预览数据。如下所示：

```
>>> 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 |

有时候，并不需要都看到所有字段，便可以从中筛选出一部分。如下所示：

```
>>> users[['user_id', 'age']].head(5)
```

| | user_id | age |
|---|---|---|
| 0 | 1 | 24 |
| 1 | 2 | 53 |
| 2 | 3 | 23 |
| 3 | 4 | 24 |
| 4 | 5 | 33 |

有时候只是排除个别字段。如下所示：

```
>>> 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 |

排除掉一些字段的同时，想要通过计算得到一些新的列，比如将sex为M的置为True，否则为False，并取名叫sex_bool。如下所示：

```
>>> 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 |

若想知道年龄在20到25岁之间的人有多少个，如下所示：

```
>>> users.age.between(20, 25).count().rename('count')
943
```

若想知道男女用户分别有多少，如下所示：

```
>>> users.groupby(users.sex).count()
```

| | sex | count |
|---|---|---|
| 0 | F | 273 |
| 1 | M | 670 |

若想将用户按职业划分，从高到底，获取人数最多的前10个职业，如下所示：

```
>>> 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 |

DataFrame API提供了value_counts方法来快速达到同样的目的。如下所示：

```
>>> users.occupation.value_counts()[:10]
```

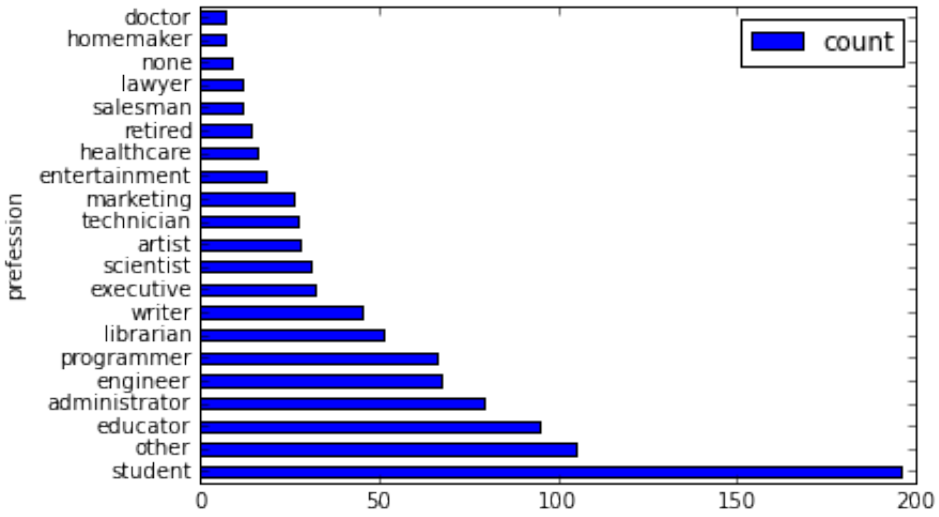
| | occupation | count |
|---|---|---|
| 0 | student | 196 |
| 1 | other | 105 |
| 2 | educator | 95 |
| 3 | administrator | 79 |
| 4 | engineer | 67 |
| 5 | programmer | 66 |
| 6 | librarian | 51 |
| 7 | writer | 45 |
| 8 | executive | 32 |
| 9 | scientist | 31 |

使用更直观的图来查看这份数据，如下所示：

```
>>> %matplotlib inline
```

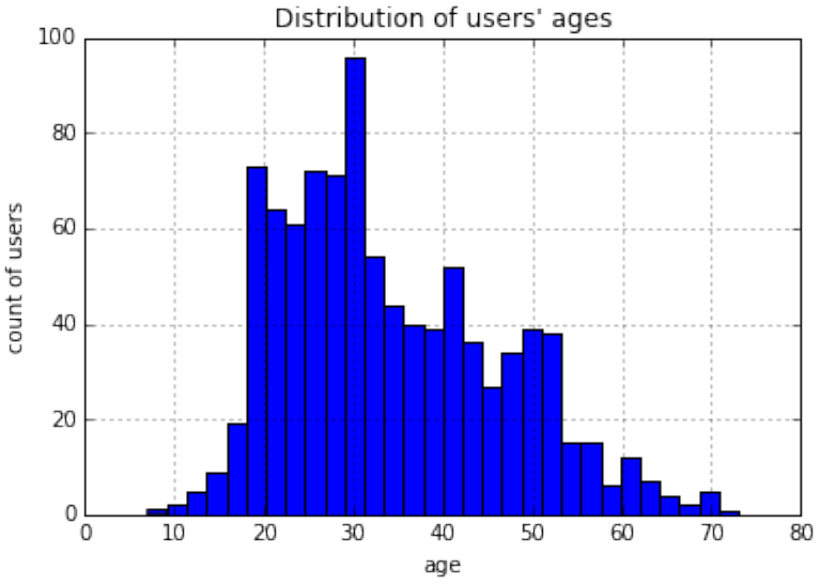
使用横向的柱状图来可视化，如下所示：

```
>>> users['occupation'].value_counts().plot(kind='barh', x='occupation
',
ylabel='prefession')
```




将年龄分成30组，查看各年龄分布的直方图，如下所示：

```
>>> users.age.hist(bins=30, title="Distribution of users' ages",
xlabel='age', ylabel='count of users')
```



使用join把这三张表进行联合后，把它保存成一张新的表。如下所示：

```
>>> 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
}
```

把0到80岁的年龄，分成8个年龄段，如下所示：

```
>>> 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')]
```

取分组和年龄唯一的前10条数据来进行查看，如下所示：

```
>>> 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 |

对各个年龄分组下，用户的评分总数和评分均值进行查看，如下所示：

```
>>> 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 |

**Configuration**

PyODPS提供了一系列的配置选项，可通过`odps.options`获得。可配置的MaxCompute选项，如下所示：

- 通用配置

| 选项 | 说明 | 默认值 |
|---|---|---|
| end_point | MaxCompute 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 | 1 |
| 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 |

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| 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 |
| 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MaxCompute2.0语言扩展 | False |

- 数据上传/下载配置

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| tunnel.endpoint | Tunnel Endpoint | None |
| tunnel.use_instance_tunnel | 使用Instance Tunnel获取执行结果 | True |
| tunnel.limited_instance_tunnel | 限制Instance Tunnel获取结果的条数 | True |
| 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 |

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| 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 |

- PyODPS ML配置

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| 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 | True |
| ml.model_volume | 在使用ModelTransfer时使用的Volume名称 | pyodps_volume |

## 3.3 PyODPS DataFrame中使用pandas、scipy和scikit-learn

*PyODPS DataFrame*提供了类似pandas的接口来操作MaxCompute数据，同时也支持在本地使用pandas和使用数据库来执行。

PyODPS DataFrame不仅支持类似pandas的map和apply方法，也提供了MapReduce API来扩展pandas语法以适应大数据环境。

PyODPS的自定义函数是序列化到MaxCompute上执行，MaxCompute的Python环境仅包含numpy第三方包。现在，MaxCompute在sprint 27及更高版本的isolation，可以实现在自定义函数中使用pandas、scipy或scikit-learn等包含c代码的库。

> 说明：
> PyODPS需要0.7.4及以上版本 。

**上传第三方包**

> 说明：
> 您只需上传一次第三方包，当MaxCompute资源有了这些包，可直接跳过此步。

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。

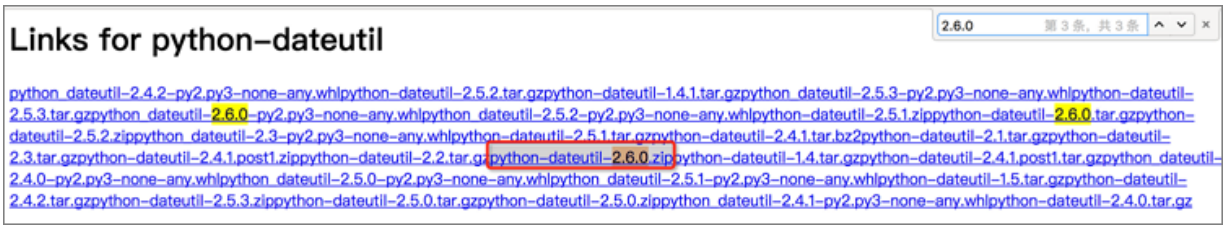
其次，要想在MaxCompute上运行，需要包含所有的依赖包，这个是比较繁琐的。各个包的依赖情况如下表所示。

| 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。

您可进入*python-dateutils*找到*python-dateutil-2.6.0.zip*进行下载。




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。

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。

其他包的操作同上，需下载资源如下表所示。

| 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 |

 说明：

您也可以使用PyODPS的资源上传接口来完成资源的上传，同样只需操作一遍。

**编写代码验证**

1. 写一个简单的函数，里面用到所有的库，最好是在函数中import这些第三方库。

```
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
```

 说明：

上述代码只是示例，目标是用到上文所说的所有的包。

2. 写完函数后，写一个简单的map。

 说明：

运行时要确保打开isolation，如果在project级别没有打开，也可在运行时打开一个可以设置全局的选项。

```
from odps import options

options.sql.settings = {'odps.isolation.session.enable': True}
```

您也可以在execute方法上指定本次执行打开isolation。

同样，您可以在全局通过options.df.libraries指定用到的包，也可以在execute时指定。这里需要指定所有的包，包括依赖。

3. 调用定义的函数。

```
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].sepal_length.map(test).execute(hints=hints, libraries
=libraries)
```

总结

对于要用到的第三方库及其依赖，如果已经上传，可以直接编写代码，并指定用到的libraries即可。否则，需要按照上述操作上传第三方库。

**PyODPS相关资源**

- 相关文档请参见*PyODPS使用指南*。
- 相关代码请参见*aliyun-odps-python-sdk*。

# 4 处理非结构化数据

## 4.1 前言

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，能够调度大量的节点做并行计算，同时对分布式计算中的 failover、重试等均有一套行之有效的处理管理机制。

MaxCompute SQL 作为分布式数据处理的主要入口，为快速方便处理/存储 EB 级别的离线数据提供强有力的支持。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，在这样的背景下，MaxCompute 计算框架也在不断的演化，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 面对的主要是以 cfile 列格式，存储在内部 MaxCompute 表格中的结构化数据。而对于 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），需要首先通过各种工具导入 MaxCompute 表，然后进行计算。数据导入的过程，具有较大的局限性。以 OSS 为例子，想要在 MaxCompute 中处理 OSS 上的数据，通常有以下两种做法：

- 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。
- 写 UDF，在 UDF 里直接调用 OSS SDK 访问 OSS 数据。

但这两种做法都有不足之处：

- 第一种需要在 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。
- 第二种通常需要申请 UDF 网络访问权限，还要开发者自己控制作业并发数和数据如何分片的问题。

本节将介绍一种外部表的功能，支持旨在提供处理除了 MaxCompute 现有表格以外的其他数据的能力。在这个框架中，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。

这里的 **各种数据** 涵盖两个维度：

**多样的数据存储介质**：插件式的框架可以对接多种数据存储介质，比如 OSS、OTS。

**多样的数据格式**：MaxCompute 表是结构化的数据，而外部表可以不限于结构化数据。

- 完全无结构数据，比如图像、音频、视频文件、raw binaries 等。
- 半结构化数据，比如 csv、tsv 等隐含一定 schema 的文本文件。
- 非 cfile 的结构化数据，比如 orc/parquet 文件，甚至 hbase/OTS 数据。

接下来将通过两个简单的示例，来帮助您深入了解非结构化数据的处理，详情请参见 访问 *OSS* 非结构化数据 和 访问 *OTS* 非结构化数据。

## 4.2 OSS的STS模式授权

创建外部表时Location访问OSS的账号支持传入明文AccessKeyId和AccessKeySecret，但这样做有泄露账号的风险。在某些场景下，这种风险是不可容忍的，因此MaxCompute提供了更为安全的方式来访问OSS。

MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）来解决账号的安全问题。您可通过以下两种方式授予权限：

- 当MaxCompute和OSS的owner是同一个账号时 ，可以直接在RAM控制台进行一键授权操作。
- 自定义授权。

1. 首先需要在RAM中授权MaxCompute访问OSS的权限。创建角色，角色名如AliyunODPSDefaultRole或AliyunODPSRoleForOtherUser，并将策略内容设置为：

```
--当MaxCompute和OSS的Owner是同一个账号
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"Service": [
"odps.aliyuncs.com"
]
}
}
],
"Version": "1"
}
--当MaxCompute和OSS的Owner不是同一个账号
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"Service": [
"MaxCompute的Owner云账号id@odps.aliyuncs.com"
]
}
}
],
```

```
"Version": "1"
}
```

2. 授予角色访问OSS必要的权限 AliyunODPSRolePolicy 。如下所示：

```
{
"Version": "1",
"Statement": [
{
"Action": [
 "oss:ListBuckets",
 "oss:GetObject",
 "oss:ListObjects",
 "oss:PutObject",
 "oss:DeleteObject",
 "oss:AbortMultipartUpload",
 "oss:ListParts"
],
"Resource": "*",
"Effect": "Allow"
}
]
}
--可自定义其他权限
```

3. 再将权限`AliyunODPSRolePolicy`授权给该角色。

**说明：**

授权完成后，查看角色详情获取Role的Ran信息，后续创建OSS外部表时需指定这个Ran信息。

## 4.3 访问OSS非结构化数据

本文将为您介绍如何在MaxCompute上轻松访问OSS的数据。

**STS模式授予权限**

MaxCompute需要直接访问OSS的数据，前提需要将OSS的数据相关权限赋给MaxCompute的访问账号，您可通过以下两种方式授予权限。

- **当MaxCompute和OSS的owner是同一个账号时**，可以直接登录阿里云账号后，点击此处完成一键授权。
- 自定义授权。

1. 首先需要在*RAM*中授予MaxCompute访问OSS的权限。登录*RAM*控制台（若MaxCompute和OSS不是同一个账号，此处需由OSS账号登录进行授权），通过控制台中的角色管理创建角色 ，角色名如AliyunODPSDefaultRole或AliyunODPSRoleForOtherUser。

2. 修改角色策略内容设置，如下所示。

```
--当MaxCompute和OSS的Owner是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
--当MaxCompute和OSS的Owner不是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号id@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

3. 授予角色访问OSS必要的权限**AliyunODPSRolePolicy**，如下所示。

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "oss:ListBuckets",
   "oss:GetObject",
   "oss:ListObjects",
   "oss:PutObject",
   "oss:DeleteObject",
   "oss:AbortMultipartUpload",
   "oss:ListParts"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
--可自定义其他权限
```

4. 将权限**AliyunODPSRolePolicy**授权给该角色。

### 内置extractor访问OSS数据

访问外部数据源时，需要用户自定义不同的Extractor，同时您也可以使用MaxCompute内置的Extractor，来读取按照约定格式存储的*OSS*数据。只需要创建一个外部表，便可把这张表作为源表进行查询。

假设有一份CSV数据存在*OSS*上，endpoint为`oss-cn-shanghai-internal.aliyuncs.com`，bucket为`oss-odps-test`，数据文件的存放路径为*`/demo/vehicle.csv`*。

### 创建外部表

创建外部表，语句如下：

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.CsvStorageHandler' -- (1)
WITH SERDEPROPERTIES (
 'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
) -- (2)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/'; -- (3)(4)
```

上述语句，说明如下：

- `com.aliyun.odps.CsvStorageHandler`是内置的处理CSV格式文件的`StorageHandler`，它定义了如何读写CSV文件。您只需指明这个名字，相关逻辑已经由系统实现。
- `odps.properties.rolearn`中的信息是RAM中`AliyunODPSDefaultRole`的`Arn`信息。您可以通过RAM控制台中的角色详情获取。
- LOCATION必须指定一个OSS目录，默认系统会读取这个目录下所有的文件。
  - 建议您使用OSS提供的内网域名，否则将产生OSS流量费用。
  - 建议您存放OSS数据的区域对应您开通MaxCompute的区域。由于MaxCompute只有在部分区域部署，我们不承诺跨区域的数据连通性。

- OSS的连接格式为oss://oss-cn-shanghai-internal.aliyuncs.com/Bucket名称/目录名称/。目录后不要加文件名称，如下的集中用法都是错误的：

```
http://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/
 -- 不支持http连接
https://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/
 -- 不支持https连接
oss://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo
 -- 连接地址错误
oss://oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/vehicle.csv  -- 不必指定文件名
```

- 外部表只是在系统中记录了与OSS目录的关联，当Drop这张表时，对应的LOCATION数据不会被删除。

如果想查看创建好的外部表结构信息，可以执行语句：

```
desc extended <table_name>;
```

在返回的信息里，除了跟内部表一样的基础信息外，Extended Info包含外部表StorageHandler、Location等信息。

**查询外部表**

外部表创建成功后，便可如同普通表一样使用这个外部表。假设/demo/vehicle.csv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
```

执行如下SQL语句：

```
select recordId, patientId, direction from ambulance_data_csv_external
 where patientId > 25;
```

**说明：**

目前外部表只能通过MaxCompute SQL操作，MaxCompute MapReduce目前无法操作外部表。

这条语句会提交一个作业，调用内置csv extractor，从OSS读取数据进行处理。输出结果如下：

```
+------------+------------+-----------+
| recordId   | patientId  | direction |
+------------+------------+-----------+
```

```
| 1           | 51          | S          |
| 3           | 48          | NE         |
| 4           | 30          | W          |
| 5           | 47          | S          |
| 7           | 53          | N          |
| 8           | 63          | SW         |
| 10          | 31          | N          |
+-------------+-------------+------------+
```

### 自定义Extractor访问OSS

当OSS中的数据格式比较复杂，内置的Extractor无法满足需求时，需要自定义Extractor来读取OSS文件中的数据。

例如有一个txt数据文件，并不是CSV格式，记录之间的列通过|分隔。比如`/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv`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
```

- 定义**Extractor**

  写一个通用的Extractor，将分隔符作为参数传进来，可以处理所有类似格式的text文件。如下所示：

  ```
  /**
   * Text extractor that extract schematized records from formatted
  plain-text(csv, tsv etc.)
   **/
  public class TextExtractor extends Extractor {
    private InputStreamSet inputs;
    private String columnDelimiter;
    private DataAttributes attributes;
    private BufferedReader currentReader;
    private boolean firstRead = true;
    public TextExtractor() {
      // default to ",", this can be overwritten if a specific
  delimiter is provided (via DataAttributes)
      this.columnDelimiter = ",";
    }
    // no particular usage for execution context in this example
    @Override
    public void setup(ExecutionContext ctx, InputStreamSet inputs,
  DataAttributes attributes) {
      this.inputs = inputs; //  inputs 是一个 InputStreamSet，每次调用
  next() 返回一个 InputStream，这个 InputStream 可以读取一个 OSS 文件的所有
  内容。
  ```

```
    this.attributes = attributes;
    // check if "delimiter" attribute is supplied via SQL query
    String columnDelimiter = this.attributes.getValueByKey("
delimiter"); //delimiter 通过 DDL 语句传参。
    if ( columnDelimiter != null)
    {
      this.columnDelimiter = columnDelimiter;
    }
    // note: more properties can be inited from attributes if needed
  }
  @Override
  public Record extract() throws IOException {//extactor() 调用返回一
条 Record，代表外部表中的一条记录。
    String line = readNextLine();
    if (line == null) {
      return null; // 返回 NULL 来表示这个表中已经没有记录可读。
    }
    return textLineToRecord(line); // textLineToRecord 将一行数据按照
delimiter 分割为多个列。
  }
  @Override
  public void close(){
    // no-op
  }
}
```

textLineToRecord将数据分割的完整实现请参见此处。

**定义StorageHandler**

StorageHandler作为External Table自定义逻辑的统一入口。

```
package com.aliyun.odps.udf.example.text;
public class TextStorageHandler extends OdpsStorageHandler {
  @Override
  public Class<? extends Extractor> getExtractorClass() {
    return TextExtractor.class;
  }
  @Override
  public Class<? extends Outputer> getOutputerClass() {
    return TextOutputer.class;
  }
}
```

**编译打包**

将自定义代码编译打包，并上传到MaxCompute。

```
add jar odps-udf-example.jar;
```

- 创建**External**表

  与使用内置Extractor相似，首先需要创建一张外部表，不同的是在指定外部表访问数据的时候，需要使用自定义的StorageHandler。

创建外部表语句如下:

```
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' --
STORED BY 指定自定义 StorageHandler 的类名。
  with SERDEPROPERTIES (
'delimiter'='\\|',  --SERDEPROPERITES 可以指定参数，这些参数会通过
DataAttributes 传递到 Extractor 代码中。
'odps.properties.rolearn'='acs:ram::xxxxxxxxxxxxx:role/aliyunodps
defaultrole'
)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/SampleData/CustomTxt/AmbulanceData/'
USING 'odps-udf-example.jar'; --同时需要指定类定义所在的jar包。
```

- 查询外部表

  执行如下SQL语句：

```
select recordId, patientId, direction from ambulance_data_txt_e
xternal where patientId > 25;
```

**自定义Extractor访问非文本文件数据**

在前面我们看到了通过内置与自定义的Extractor可以轻松处理存储在OSS上的CSV等文本数据。接下来以语音数据（wav格式文件）为例，为您介绍如何通过自定义的Extractor访问并处理OSS上的非文本文件。

**这里从最终执行的SQL开始，介绍以MaxCompute SQL为入口，处理存放在OSS上的语音文件的使用方法。**

创建外部表SQL如下：

```
CREATE EXTERNAL TABLE IF NOT EXISTS speech_sentence_snr_external
(
sentence_snr double,
id string
)
STORED BY 'com.aliyun.odps.udf.example.speech.SpeechStorageHandler'
WITH SERDEPROPERTIES (
    'mlfFileName'='sm_random_5_utterance.text.label' ,
    'speechSampleRateInKHz' = '16'
)
```

```
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
dev/SpeechSentenceTest/'
USING 'odps-udf-example.jar,sm_random_5_utterance.text.label';
```

如上所示，同样需要创建外部表，然后通过外部表的Schema定义了希望通过外部表从语音文件中抽取出来的信息：

- 一个语音文件中的语句信噪比（SNR）：sentence_snr。
- 对应语音文件的名字：id。

创建外部表后，通过标准的Select语句进行查询，则会触发Extractor运行计算。此处便可感受到，在读取处理OSS数据时，除了可以对文本文件做简单的反序列化处理，还可以通过自定义Extractor实现更复杂的数据处理抽取逻辑。比如：在此示例中，通过自定义的`com.aliyun.odps.udf.example.speech.SpeechStorageHandler`中封装的Extractor，实现了对语音文件计算平均有效语句信噪比的功能，并将抽取出来的结构化数据直接进行SQL运算（WHERE sentence_snr > 10），最终返回所有信噪比大于10的语音文件以及对应的信噪比值。

在OSS地址`oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/dev/SpeechSentenceTest/`上，存储了原始的多个WAV格式的语音文件，MaxCompute框架将读取该地址上的所有文件，并在必要的时候进行文件级别的分片，自动将文件分配给多个计算节点处理。每个计算节点上的Extractor则负责处理通过InputStreamSet分配给该节点的文件集。具体的处理逻辑则与用户单机程序相仿，您不需关心分布计算中的种种细节，按照类单机方式实现其用户算法即可。

定制化的`SpeechSentenceSnrExtractor`主体逻辑，说明如下。

首先在`setup`接口中读取参数，进行初始化，并且导入语音处理模型（通过resource引入）。

```
public SpeechSentenceSnrExtractor(){
    this.utteranceLabels = new HashMap<String, UtteranceLabel>();
  }
  @Override
  public void setup(ExecutionContext ctx, InputStreamSet inputs,
DataAttributes attributes){
    this.inputs = inputs;
    this.attributes = attributes;
    this.mlfFileName = this.attributes.getValueByKey(MLF_FILE_A
TTRIBUTE_KEY);
    String sampleRateInKHzStr = this.attributes.getValueByKey(
SPEECH_SAMPLE_RATE_KEY);
    this.sampleRateInKHz = Double.parseDouble(sampleRateInKHzStr);
    try {
      // read the speech model file from resource and load the model
into memory
      BufferedInputStream inputStream = ctx.readResourceFileAsStream(
mlfFileName);
      loadMlfLabelsFromResource(inputStream);
      inputStream.close();
```

```
    } catch (IOException e) {
      throw new RuntimeException("reading model from mlf failed with
exception " + e.getMessage());
    }
  }
```

Extractor()接口中，实现了对语音文件的具体读取和处理逻辑，对读取的数据根据语音模型进行信噪比的计算，并且将结果填充成[snr, id]格式的Record。

上述示例对实现进行了简化，同时也没有包括涉及语音处理的算法逻辑，具体实现请参见MaxCompute SDK在开源社区中提供的样例代码。

```
@Override
  public Record extract() throws IOException {
    SourceInputStream inputStream = inputs.next();
    if (inputStream == null){
      return null;
    }
    // process one wav file to extract one output record [snr, id]
    String fileName = inputStream.getFileName();
    fileName = fileName.substring(fileName.lastIndexOf('/') + 1);
    logger.info("Processing wav file " + fileName);
    String id = fileName.substring(0, fileName.lastIndexOf('.'));
    // read speech file into memory buffer
    long fileSize = inputStream.getFileSize();
    byte[] buffer = new byte[(int)fileSize];
    int readSize = inputStream.readToEnd(buffer);
    inputStream.close();
    // compute the avg sentence snr
    double snr = computeSnr(id, buffer, readSize);
    // construct output record [snr, id]
    Column[] outputColumns = this.attributes.getRecordColumns();
    ArrayRecord record = new ArrayRecord(outputColumns);
    record.setDouble(0, snr);
    record.setString(1, id);
    return record;
  }
  private void loadMlfLabelsFromResource(BufferedInputStream
fileInputStream)
          throws IOException {
    //  skipped here
  }
  // compute the snr of the speech sentence, assuming the input buffer
 contains the entire content of a wav file
  private double computeSnr(String id, byte[] buffer, int validBuffe
rLen){
    // computing the snr value for the wav file (supplied as byte
buffer array), skipped here
  }
```

执行查询，如下所示：

```
select sentence_snr, id
    from speech_sentence_snr_external
```

```
where sentence_snr > 10.0;
```

获得计算结果，如下所示：

```
-------------------------------------------------------------
| sentence_snr |                    id                     |
-------------------------------------------------------------
|   34.4703    |          J310209090013_H02_K03_042        |
-------------------------------------------------------------
|   31.3905    | tsh148_seg_2_3013_3_6_48_80bd359827e24dd7_0 |
-------------------------------------------------------------
|   35.4774    | tsh148_seg_3013_1_31_11_9d7c87aef9f3e559_0  |
-------------------------------------------------------------
|   16.0462    | tsh148_seg_3013_2_29_49_f4cb0990a6b4060c_0  |
-------------------------------------------------------------
|   14.5568    |   tsh_148_3013_5_13_47_3d5008d792408f81_0   |
-------------------------------------------------------------
```

综上所述，通过自定义Extractor，便可在SQL语句上分布式地处理多个OSS上的语音数据文件。同样的方法，也可以方便地利用MaxCompute的大规模计算能力，完成对图像，视频等各种类型非结构化数据的处理。

**数据的分区**

在前面的例子中，一个外部表关联的数据通过LOCATION上指定的OSS目录来实现，而在处理的时候，MaxCompute是读取目录下的所有数据，**包括子目录中的所有文件**。在数据量比较大，尤其是对于随着时间不断积累的数据目录，对全目录扫描可能带来不必要的I/O以及数据处理时间。 解决这个问题通常有两种做法：

- 直接的方法：您对数据存放地址做好规划，考虑使用多个EXTERNAL TABLE来描述不同部分的数据，让每个EXTERNALTABLE的LOCATION指向数据的一个子集。
- 数据分区方法：EXTERNAL TABLE与内部表一样，**支持分区表的功能**，可以通过这个功能来对数据做系统化的管理。

本章节主要介绍EXTERNAL TABLE的分区功能。

- **分区数据在OSS上的标准组织方式和路径格式**

  与MaxCompute内部表不同，对于存放在外部存储上(如OSS)上面的数据，MaxComput没有数据的管理权，因此如果需要使用分区表功能，在OSS上数据文件的存放路径必须符合一定的格式，路径格式如下：

  ```
  partitionKey1=value1\partitionKey2=value2\...
  ```

  **场景示例**

将每天产生的LOG文件存放在OSS上，并需要通过MaxCompute进行数据处理，数据处理时需按照粒度为“天”来访问一部分数据。假设这些LOG文件为CSV格式且可以用内置extractor访问（复杂自定义格式用法也类似），那么外部分区表定义数据如下：

```
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'
  WITH SERDEPROPERTIES (
 'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrol
e'
)
  LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/
log_data/';
```

如上建表语句，和前面的例子区别在于定义EXTERNAL TABLE时，通过PARTITIONED BY的语法指定该外部表为分区表，该例子是一个三层分区分区表，分区的key分别是year，month和day。

为了让分区生效，在OSS上存储数据时需要遵循location的路径格式。如有效的路径存储layout：

```
osscmd ls oss://oss-odps-test/log_data/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=01/logfile.1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data/year
=2016/month=07/day=10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data/year
=2016/month=08/day=08/logfile
...
```

**说明：**

因为数据是离线准备的，即通过osscmd或者其他OSS工具上载到OSS存储服务，所以数据路径格式也在上载时决定。

通过**ALTER TABLE ADD PARTITIONDDL**语句，即可把这些分区信息引入MaxCompute。

对应的DDL语句：

```
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')
...
```

 说明：

以上这些操作与标准的MaxCompute内部表操作一样，分区的详情请参见分区。在数据准备好并且PARTITION信息引入MaxCompute之后，即可通过SQL语句对OSS外表数据的分区进行操作。

此时分析数据时，可以指定指需分析某天的数据，如只想分析2016年6月1号当天，有多少不同的IP出现在LOG里面，可以通过如下语句实现。

```
SELECT count(distinct(ip)) FROM log_table_external WHERE year = '
2016' AND month = '06' AND day = '01';
```

该语句对log_table_external这个外表对应的目录，将只访问`log_data/year=2016/month=06/day=01`子目录下的文件（logfile和logfile.1），不会对整个log_data/目录作全量数据扫描，避免大量无用的I/O操作。

同样如果只希望对2016年下半年的数据做分析，则执行如下语句。

```
SELECT count(distinct(ip)) FROM log_table_external
WHERE year = '2016' AND month > '06';
```

只访问OSS上面存储的下半年的LOG数据。

- 分区数据在OSS上的自定义路径

  如果事先存在OSS上的历史数据，但是又不是根据`partitionKey1=value1\partitionKey2=value2\...`路径格式来组织存放，也需要通过MaxCompute的分区方式来进行访问计算时，MaxCompute也提供了通过自定义路径来引入partition的方法。

  假设OSS数据路径只有简单的分区值（而无分区key信息），也就是数据的layout为：

```
osscmd ls oss://oss-odps-test/log_data_customized/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/01/logfile.1
```

```
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/07/10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/08/08/logfile
...
```

外部表建表DDL可参看前面的示例，同样在建表语句里指定好分区key。

不同的子目录指定到不同的分区，可通过类似如下自定义分区路径的DDL语句实现。

```
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '
06', day = '01')
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/
log_data_customized/2016/06/01/';
```

在ADD PARTITION的时候增加了LOCATION信息，从而实现自定义分区数据路径后，即使数据存放不符合推荐的`partitionKey1=value1\partitionKey2=value2\...`格式，也能正确的实现对子目录数据的分区访问了。

## 4.4 处理OSS的开源格式数据

访问*OSS*非结构化数据为您介绍了如何在MaxCompute上访问存储在OSS上的文本、音频、图像等格式的数据。本文将为您介绍对于存储在OSS上的各种流行的开源数据格式（ORC、PARQUET、SEQUENCEFILE、RCFILE、AVRO和TEXTFILE）如何通过非结构化框架在MaxCompute进行处理。

非结构框架会直接调用开源社区的实现来进行开源数据格式的解析，并且与MaxCompute系统无缝对接。

**说明：**

处理OSS的开源格式数据前，需要首先对OSS进行*STS*模式授权。

### 创建External Table

MaxCompute非结构化数据框架通过External Table与各种数据的关联，关联OSS上开源格式数据的External Table建表的DDL语句格式如下所示。

```
DROP TABLE [IF EXISTS] <external_table>;
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
[ROW FORMAT SERDE '<serde class>'
  [WITH SERDEPROPERTIES ('odps.properties.rolearn'='${roleran}' [,'
name2'='value2',...])]
]
STORED AS <file format>
```

```
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

**说明：**

该语法格式与Hive的语法相当接近，但需注意以下问题。

- STORED AS关键字，在该语法格式中不是普通非结构化外表用的STORED BY关键字，这是目前在读取开源兼容数据时独有的。

  STORED AS后面接的是文件格式名字， 比如ORC/PARQUET/RCFILE/SEQUENCEFILE/TEXTFILE等。
- 外部表的column schemas必须与具体OSS上存储存储数据的schema相符合。
- ROW FORMAT SERDE非必选选项，只有在使用一些特殊的格式上，比如TEXTFILE时才需要使用。
- WITH SERDEPROPERTIES当关联OSS权限使用*STS*模式授权时，需要该参数指定**odps.properties.rolearn**属性，属性值为RAM中具体使用的Role的Arn的信息。

  若不用STS模式，则无需指定该属性，直接在Location传入明文AccessKeyId和AccessKeySecret。
- Location若关联OSS需使用明文AK，写法如下所示。

```
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${
bucket}/${userPath}/'
```

**关联OSS的PARQUET数据示例**

假设有一些PARQUET文件存放在一个OSS路径上，每个文件都是PARQUET格式，存放的schema为16列（4列Bigint、4列Double和8列String）的数据，建表DDL语句如下所示。

```
CREATE EXTERNAL TABLE tpch_lineitem_parquet
(
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenumber bigint,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  l_linestatus string,
  l_shipdate string,
  l_commitdate string,
  l_receiptdate string,
  l_shipinstruct string,
  l_shipmode string,
  l_comment string
```

```
)
STORED AS PARQUET
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/parquet_data/';
```

关联**OSS**的**TEXTFILE**数据分区表示例

如果数据是每行以JSON格式，存储成OSS上TEXTFILE文件，同时数据在OSS通过多个目录组织，这时可以使用MaxCompute分区表和数据关联，建表DDL语句如下所示。

```
CREATE EXTERNAL TABLE tpch_lineitem_textfile
(
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenumber bigint,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  l_linestatus string,
  l_shipdate string,
  l_commitdate string,
  l_receiptdate string,
  l_shipinstruct string,
  l_shipmode string,
  l_comment string
)
PARTITIONED BY (ds string)
ROW FORMAT serde 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/text_data/';
```

如果OSS表目录下面的子目录是以Partition Name方式组织，示例如下。

```
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data/ds=20170102/'
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data/ds=20170103/'
...
```

则可以使用以下DDL语句ADD PARTITION。

```
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170102");
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170103");
```

如果OSS分区目录不是按这种方式组织，或者根本不在表目录下，示例如下。

```
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data_20170102/;
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data_20170103/;
```

```
...
```

则可以使用以下DDL语句ADD PARTITION。

```
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170102")
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/text_data_20170102/';
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170103")
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/text_data_20170103/';
...
```

**读取以及处理OSS的开源格式数据**

对比前文的两个创建外部表示例，可以看出对于不同文件类型，只要简单修改STORED AS后的格式名。在下述示例中将只集中描述对上面PARQUET数据对应的外表（tpch_lineitem_parquet）的处理。如果要处理不同的文件类型，只要在DDL创建外表时指定是PARQUET/ORC/TEXTFILE/RCFILE/TEXTFILE即可，处理数据的语句一样。

- **直接读取以及处理OSS的开源数据**

  创建数据外表进行关联后，直接对外表就可以进行与普通MaxCompute表一样的操作，如下所示。

  ```
  SELECT l_returnflag, l_linestatus,
  SUM(l_extendedprice*(1-l_discount)) AS sum_disc_price,
  AVG(l_quantity) AS avg_qty,
  COUNT(*) AS count_order
  FROM tpch_lineitem_parquet
  WHERE l_shipdate <= '1998-09-02'
  GROUP BY l_returnflag, l_linestatus;
  ```

  外表tpch_lineitem_parquet被当作一个普通的内部表一样使用，不同在于MaxCompute内部计算引擎是直接从OSS读取对应的PARQUET数据进行处理。

  前文创建的关联textfile的外部分区表tpch_lineitem_textfile，因为使用了`ROW FORMAT` + `STORED AS`，需要手动设置flag（只使用STORED AS，odps.sql.hive.compatible默认为TRUE），再进行读取，否则会有报错。

  ```
  SELECT * FROM tpch_lineitem_textfile LIMIT 1;
  FAILED: ODPS-0123131:User defined function exception - Traceback:
  com.aliyun.odps.udf.UDFException: java.lang.ClassNotFoundException:
  com.aliyun.odps.hive.wrapper.HiveStorageHandlerWrapper
  --需要手动设置hive兼容flag
  set odps.sql.hive.compatible=true;
  SELECT * FROM tpch_lineitem_textfile LIMIT 1;
  +------------+------------+------------+--------------+------------
  +-----------------+------------+------------+--------------
  +--------------+------------+--------------+--------------
  +----------------+------------+-----------+
  ```

```
| l_orderkey | l_partkey  | l_suppkey  | l_linenumber | l_quantity
 | l_extendedprice | l_discount | l_tax      | l_returnflag
 | l_linestatus | l_shipdate | l_commitdate | l_receiptdate |
l_shipinstruct | l_shipmode | l_comment |
+------------+------------+------------+--------------+------------
+-----------------+------------+------------+--------------
+--------------+------------+--------------+--------------
+----------------+------------+-----------+
| 5640000001 | 174458698  | 9458733    | 1            | 14.0
  | 23071.58       | 0.08       | 0.06       | N            | O
         | 1998-01-26 | 1997-11-16   | 1998-02-18    | TAKE BACK
RETURN | SHIP       | cuses nag silently. quick |
+------------+------------+------------+--------------+------------
+-----------------+------------+------------+--------------
+--------------+------------+--------------+--------------
+----------------+------------+-----------+
```

**说明：**

直接使用外表，每次读取数据都需要涉及外部OSS的I/O操作，且MaxCompute系统本身针对内部存储做的许多高性能优化都用不上，如此一来性能上就会有所损失。 因此如果是需要对数据进行反复计算以及对计算的高效性比较敏感的场景，推荐使用下文介绍的用法：先将数据导入MaxCompute内部再进行计算。

- **将OSS的开源数据导入MaxCompute再进行计算**

  首先创建一个与外部表schema一样的内部表tpch_lineitem_internal，然后将OSS上的开源数据导入MaxCompute内部表，以MaxCompute内部数据存储格式进行存储。

```
CREATE TABLE tpch_lineitem_internal LIKE tpch_lineitem_parquet;
INSERT OVERWRITE TABLE tpch_lineitem_internal;
SELECT * FROM tpch_lineitem_parquet;
```

  接下来直接对内部表进行同样的操作。

```
SELECT l_returnflag, l_linestatus,
SUM(l_extendedprice*(1-l_discount)) AS sum_disc_price,
AVG(l_quantity) AS avg_qty,
COUNT(*) AS count_order
FROM tpch_lineitem_internal
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus;
```

  通过此方式将数据先导入MaxCompute系统进行存储，对同样数据进行计算处理会更高效。

## 4.5 输出到OSS的非结构化数据

访问OSS非结构化数据为您介绍了MaxCompute如何通过外部表的关联进行访问并处理存储在OSS的非结构化数据，实际上MaxCompute的非结构化框架也支持通过insert方式将MaxCompute的数据直接输出到OSS，MaxCompute也是通过外部表关联OSS，进行数据输出。

输出数据到OSS通常是两种情况：

- MaxCompute内部表输出到关联OSS的外部表。
- MaxCompute处理外部表后结果直接输出到关联OSS的外部表。

与访问OSS数据一样，MaxCompute支持通过内置StorageHandler 和自定义StorageHandler进行输出。

### 通过内置StorageHandler输出到OSS

使用MaxCompute内置的StorageHandler ，可以非常方便的按照约定格式输出数据到OSS进行存储。我们只需要创建一个外部表，指明内置的StorageHandler，就能以这张表为关联，相关逻辑由系统实现。

目前MaxCompute支持2个内置StorageHandler：

- com.aliyun.odps.CsvStorageHandler ，定义如何读写csv格式数据，数据格式约定：英文逗号`,`为列分隔符，换行符为`\n`。
- com.aliyun.odps.TsvStorageHandler，定义如何读写csv格式数据，数据格式约定：`\t`为列分隔符，换行符为`\n`。
- 创建**EXTERNAL TABLE**

```
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED BY '<StorageHandler>'
[WITH SERDEPROPERTIES (  'odps.properties.rolearn'='${roleran}') ]
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

  - STORED BY，如果需求输出到OSS上的数据文件是TSV文件，则用内置`com.aliyun.odps.TsvStorageHandler`；如果需求输出到OSS上的数据文件是CSV文件，则用内置`com.aliyun.odps.CsvStorageHandler`。
  - WITH SERDEPROPERTIES，当关联OSS权限使用“STS模式授权”的“自定义授权”时，需要该参数指定’odps.properties.rolearn’属性，属性值为RAM 中具体使用的自定义role的Arn的信息。

**说明：**

STS模式授权可参看《访问OSS非结构化数据》中的对应内容。

— LOCATION，指定对应OSS存储的文件路径。若*WITH SERDEPROPERTIES*中不设置'odps.properties.rolearn'属性，且授权方式是采用明文AK，则*LOCATION*为

```
LOCATION
    'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${bucket
}/${userPath}/'
```

- 通过对External Table的 INSERT 操作实现数据输出到OSS。

 说明：

insert到OSS的单个文件的大小不能超过5G。

通过External Table关联上OSS存储路径后，可以对External Table做标准的SQL INSERT OVERWRITE/INSERT INTO操作既可将数据输出到OSS。

```
INSERT OVERWRITE|INTO TABLE <external_tablename> [PARTITION (
partcol1=val1, partcol2=val2 ...)]
select_statement
FROM <from_tablename>
[WHERE where_condition];
```

— *from_tablename*：可以是内部表，也可以是外部表（包括关联的OSS或OTS的外部表）。

— INSERT将按照外部表'STORED BY'指定 'StorageHandler'的格式（即TSV或CSV）写到OSS上。

INSERT 操作成功完成后，就可以看到OSS上的对应LOCATION产生了一系列文件。

如：external table 对应的location是*oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/tsv_output_folder/* 则，在OSS对应路径中可以看到生成一系列文件：

```
osscmd ls oss://oss-odps-test/tsv_output_folder/
2017-01-14 06:48:27 39.00B Standard oss://oss-odps-test/tsv_output
_folder/.odps/.meta
2017-01-14 06:48:12 4.80MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_0_0-0.tsv
2017-01-14 06:48:05 4.78MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_1_0-0.tsv
2017-01-14 06:47:48 4.79MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_2_0-0.tsv
...
```

可以看到，通过前面LOCATION指定的oss-odps-test这个OSS bucket下的tsv_output_folder文件夹下产生了一个'.odps'文件夹，这其中将有一些'.tsv'文件，以及一个'.meta'文件。 类似的文件结构是MaxCompute往OSS上输出所特有的：

- 通过MaxCompute对一个OSS地址，使用INSERT INTO/OVERWRITE 外部表来做输出操作，所有的数据将在指定的LOCATION下的'.odps'文件夹产生。
- '.odps'文件夹中的'.meta'文件为MaxCompute额外写出的宏数据文件，其中用于记录当前文件夹中有效的数据。 正常情况下，如果INSERT操作成功完成的话，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这个宏数据进行解析。 即使是在作业中途失败或被kill的情况下，对于INSERT OVERWRITE操作，再跑一次成功即可。
- 如果是分区表，将在tsv_output_folder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然后分区子目录里才是'.odps'文件夹。如`test/tsv_output_folder/一级分区名=分区值/n级分区名=分区值/.odps/20170113224724561g9m6csz7/M1_2_0-0.tsv`。

对于MaxCompute内置的TSV/CSV StorageHandler 处理来说，产生的文件数目与对应SQL stage的并发度是相同。

若`INSER OVERWITE ... SELECT ... FROM ...;`的操作在源数据表(from_tablename) 上分配了1000个mapper，那么最后将产生了1000个TSV/CSV文件。

### 通过自定义StorageHandler输出到OSS

除了使用内置的StorageHandler来实现在OSS上输出TSV/CSV常见文本格式，MaxCompute非结构化框架提供了通用的SDK，支持对外输出自定义数据格式文件。

与内置StorageHandler一样需要先"创建EXTERNAL TABLE"，再"通过对External Table的INSERT 操作实现数据输出到OSS"。不同点在于创建外部表时，STORED BY是需要指定自定义的StorageHandler。

说明：

MaxCompute非结构化框架通过StorageHandler这个接口来描述对各种数据存储格式的处理。具体来说，StorageHandler作为一个wrapper class, 让您指定自定义的Exatractor(用于数据的读入，解析，处理等) 以及Outputer(用于数据的处理和输出等)。 自定义的StorageHandler 应该继承OdpsStorageHandler，实现getExtractorClass以及getOutputerClass 两个接口。

接下来我们用访问*OSS*非结构化数据中"自定义 Extractor 访问 OSS"的'TextStorageHandler '例子来介绍MaxCompute如何通过自定义StorageHandler 将数据输出到OSS的txt文件，且以'|'为列分隔符，以'\n'为换行符。

说明：

*MaxCompute Studio*配置好 *MaxCompute Java Module*后，可以在examples中看到对应的示例代码。或者点击此处也看到完整代码。

- 定义**Outputer**

  输出逻辑都必须实现Outputer接口：

```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.io.OutputStreamSet;
import com.aliyun.odps.io.SinkOutputStream;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.Outputer;
import java.io.IOException;
public class TextOutputer extends Outputer {
    private SinkOutputStream outputStream;
    private DataAttributes attributes;
    private String delimiter;
    public TextOutputer (){
        // default delimiter, this can be overwritten if a delimiter
 is provided through the attributes.
        this.delimiter = "|";
    }
    @Override
    public void output(Record record) throws IOException {
        this.outputStream.write(recordToString(record).getBytes());
    }
    // no particular usage of execution context in this example
    @Override
    public void setup(ExecutionContext ctx, OutputStreamSet
outputStreamSet, DataAttributes attributes) throws IOException {
        this.outputStream = outputStreamSet.next();
        this.attributes = attributes;
    }
    @Override
    public void close() {
        // no-op
    }
    private String recordToString(Record record){
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < record.getColumnCount(); i++)
        {
            if (null == record.get(i)){
                sb.append("NULL");
            }
            else{
                sb.append(record.get(i).toString());
            }
            if (i != record.getColumnCount() - 1){
                sb.append(this.delimiter);
            }
        }
        sb.append("\n");
        return sb.toString();
    }
```

```
}
```

Outputer接口有三个：setup, output和close, 这和Extractor的setup, extract和close三个接口基本上对称。 其中setup()和close()在一个outputer中只会调用一次。 可以在setup里面做初始化准备工作，另外通常需要把setup()传递进来的这三个参数保存成ouputerd的class variable, 方便之后output()或者close()接口中使用。 而close()这个接口用于代码的扫尾工作。

通常情况下大部分的数据处理发生在output(Record)这个接口内。 MaxCompute系统根据当前outputer分配处理的每个输入Record调用一次 output(Record)。 假设在一个output(Record) 调用返回的时候，代码已经消费完这个Record, 因此在当前output(Record)返回后，系统会将Record所使用的内存作它用，所以当Record中的信息在跨多个output()函数调用被使用时，需要调用当前处理的record.clone()方法，将当前record保存下来。

- 定义**Extractor**

Exatractor用于数据的读入，解析，处理等，如果输出的表最终不需要再通过MaxCompute进行读取等，可以无需定义。

```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.Column;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.io.InputStreamSet;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.Extractor;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
/**
 * Text extractor that extract schematized records from formatted
plain-text(csv, tsv etc.)
 **/
public class TextExtractor extends Extractor {
    private InputStreamSet inputs;
    private String columnDelimiter;
    private DataAttributes attributes;
    private BufferedReader currentReader;
    private boolean firstRead = true;
    public TextExtractor() {
        // default to ",", this can be overwritten if a specific
delimiter is provided (via DataAttributes)
        this.columnDelimiter = ",";
    }
    // no particular usage for execution context in this example
    @Override
    public void setup(ExecutionContext ctx, InputStreamSet inputs,
DataAttributes attributes) {
        this.inputs = inputs;
        this.attributes = attributes;
        // check if "delimiter" attribute is supplied via SQL query
```

```
        String columnDelimiter = this.attributes.getValueByKey("
delimiter");
        if ( columnDelimiter != null)
        {
            this.columnDelimiter = columnDelimiter;
        }
        System.out.println("TextExtractor using delimiter [" + this.
columnDelimiter + "].");
        // note: more properties can be inited from attributes if
needed
    }
    @Override
    public Record extract() throws IOException {
        String line = readNextLine();
        if (line == null) {
            return null;
        }
        return textLineToRecord(line);
    }
    @Override
    public void close(){
        // no-op
    }
    private Record textLineToRecord(String line) throws IllegalArg
umentException
    {
        Column[] outputColumns = this.attributes.getRecordColumns();
        ArrayRecord record = new ArrayRecord(outputColumns);
        if (this.attributes.getRecordColumns().length != 0){
            // string copies are needed, not the most efficient one
, but suffice as an example here
            String[] parts = line.split(columnDelimiter);
            int[] outputIndexes = this.attributes.getNeededIndexes
();
            if (outputIndexes == null){
                throw new IllegalArgumentException("No outputIndexes
 supplied.");
            }
            if (outputIndexes.length != outputColumns.length){
                throw new IllegalArgumentException("Mismatched
output schema: Expecting "
                        + outputColumns.length + " columns but get "
 + parts.length);
            }
            int index = 0;
            for(int i = 0; i < parts.length; i++){
                // only parse data in columns indexed by output
indexes
                if (index < outputIndexes.length && i == outputInde
xes[index]){
                    switch (outputColumns[index].getType()) {
                        case STRING:
                            record.setString(index, parts[i]);
                            break;
                        case BIGINT:
                            record.setBigint(index, Long.parseLong(
parts[i]));
                            break;
                        case BOOLEAN:
                            record.setBoolean(index, Boolean.
parseBoolean(parts[i]));
                            break;
```

```
                        case DOUBLE:
                            record.setDouble(index, Double.
parseDouble(parts[i]));
                            break;
                        case DATETIME:
                        case DECIMAL:
                        case ARRAY:
                        case MAP:
                        default:
                            throw new IllegalArgumentException("Type
 " + outputColumns[index].getType() + " not supported for now.");
                    }
                    index++;
                }
            }
        }
        return record;
    }
    /**
     * Read next line from underlying input streams.
     * @return The next line as String object. If all of the
contents of input
     * streams has been read, return null.
     */
    private String readNextLine() throws IOException {
        if (firstRead) {
            firstRead = false;
            // the first read, initialize things
            currentReader = moveToNextStream();
            if (currentReader == null) {
                // empty input stream set
                return null;
            }
        }
        while (currentReader != null) {
            String line = currentReader.readLine();
            if (line != null) {
                return line;
            }
            currentReader = moveToNextStream();
        }
        return null;
    }
    private BufferedReader moveToNextStream() throws IOException {
        InputStream stream = inputs.next();
        if (stream == null) {
            return null;
        } else {
            return new BufferedReader(new InputStreamReader(stream
));
        }
    }
}
```

详情请参见访问*OSS*非结构化数据文档。

- 定义**StorageHandler**

```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.udf.Extractor;
import com.aliyun.odps.udf.OdpsStorageHandler;
```

```
import com.aliyun.odps.udf.Outputer;
public class TextStorageHandler extends OdpsStorageHandler {
    @Override
    public Class<? extends Extractor> getExtractorClass() {
        return TextExtractor.class;
    }
    @Override
    public Class<? extends Outputer> getOutputerClass() {
        return TextOutputer.class;
    }
}
```

若表无需读取可不用指定Extractor接口。

- 编译打包

  将自定义代码编译打包，并作为jar 资源上传到MaxCompute。如打的jar包名为‘odps-TextStorageHandler.jar’,则上传为MaxCompute Resource如下：

```
add jar odps-TextStorageHandler.jar;
```

- 创建**external**表

  与使用内置StorageHandler类似，需要建立一个外部表，不同的是这次需要指定数据输出到外部表时候，使用自定义的StorageHandler。

```
CREATE EXTERNAL TABLE IF NOT EXISTS output_data_txt_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.examples.unstructured.text.TextStorag
eHandler'
WITH SERDEPROPERTIES(
    'delimiter'='|'
    [,'odps.properties.rolearn'='${roleran}'])
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/'
USING 'odps-TextStorageHandler.jar';
```

说明：

若需用‘odps.properties.rolearn’属性，详情请参见访问*OSS*非结构化数据的STS模式授权的自定义授权。若不用，可以参考一键授权或者在LOCATION上用明文AK。

- 通过对**External Table**的**INSERT**操作实现数据输出到**OSS**

通过自定义StorageHandler 创建External Table关联上OSS存储路径后，可以对External Table做标准的SQL INSERT OVERWRITE/INSERT INTO操作既可将数据输出到OSS,方式与内置StorageHandler一样：

```
INSERT OVERWRITE|INTO TABLE <external_tablename> [PARTITION (
partcol1=val1, partcol2=val2 ...)]
select_statement
FROM <from_tablename>
[WHERE where_condition];
```

insert操作执行成功后，与内置StorageHandler一样，可以在OSS对应LOCATION路径看到生成一系列文件于'.odps'文件夹中。

## 4.6 访问OTS非结构化数据

表格存储（Table Store）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。您可以通过*TableStore*文档对其进行了解。

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，也就是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。

前文为您介绍了如何访问*OSS*非结构化数据，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。

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Table Store数据需要有一个安全的授权通道。在此问题上，MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）来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Table Store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。
- 自定义授权

  1. 首先在RAM控制台中授予MaxCompute访问Table Store的权限。

     登录 *RAM*控制台（若MaxCompute和Table Store不是同一个账号，此处需由Table Store账号登录进行授权），创建角色，角色名叫AliyunODPSDefaultRole或AliyunODPSRoleForOtherUser。

  2. 修改策略内容设置，如下所示：

```
--当MaxCompute和Table Store的Owner是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
--当MaxCompute和Table Store的Owner不是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号的UID@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
```

```
}
```

**说明：**

您可单击右上角的登录账号，进入账号管理页面查看云账号的UID。


3. 编辑该角色的授权策略AliyunODPSRolePolicy，如下所示：

```
{
"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"
}
]
}
--还可自定义其他权限
```

4. 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，即可轻松实现对TableStore数据的处理。本节将以下述示例为例，来为您说明MaxCompute对接TableStore的一些概念和实现。

建外部表语句如下所示：

```
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
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler' -- (1)
WITH SERDEPROPERTIES ( -- (2)
'tablestore.columns.mapping'=':o_orderkey,:o_orderdate,o_custkey,
o_orderstatus,o_totalprice', -- ①
'tablestore.table.name'='ots_tpch_orders' -- ②
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
 --③
)
LOCATION 'tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.
com'; -- （3）
```

语句说明如下所示：

（1）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。

（2）SERDEPROPERITES是提供参数选项的接口，在使用TableStoreStorageHandler时，有两个必须指定的选项，分别是下文介绍的tablestore.columns.mapping、tablestore.table.name和odps.properties.rolearn。

①tablestore.columns.mapping选项：必选项，用来描述MaxCompute将访问的Table Store表的列，包括主键和属性列。

- 以:打头的用来表示Table Store主键，例如此语句中的:o_orderkey和:o_orderdate，其他的均为属性列。
- Table Store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Table Store表的所有主键，对于属性列则没有必要全部提供，可以只提供需要通过MaxCompute来访问的属性列。

②tablestore.table.name：需要访问的Table Store表名。如果指定的Table Store表名错误（不存在），则会报错，MaxCompute不会主动去创建Table Store表。

③odps.properties.rolearn中的信息是RAM中AliyunODPSDefaultRole的Arn信息。您可以通过RAM控制台中的角色详情进行获取。

（3）LOCATION clause：用来指定Table Storeinstance名字、endpoint等具体信息。这里的Table Store数据的安全访问建立在前文介绍的RAM/STS授权的前提上。

如果您想要查看创建好的外部表结构信息，可以执行如下语句：

```
desc extended <table_name>;
```

在返回的信息里，除了跟内部表一样的基础信息外，Extended Info包含外部表StorageHandler 、Location等信息。

**查询外部表**

创建External Table后，Table Store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Table Store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;
```

由上可见，使用常见的MaxCompute SQL语法，访问Table Store的所有细节由MaxComput 内部处理。这包括在列名的选择上，比如上述SQL中，使用的列名是odps_orderkey，odps_totalprice等，而不是原始Table Store中的主键名o_orderkey或属性列名o_totalprice，因为在创建External Table的DDL语句中，已经做了对应的mapping。当然您也可根据自己的需求在创建External Table时选择保留原始的TableStore主键/列名。

如果需要对一份数据做多次计算，相较每次从Table Store去远程读数据，有个更高效的办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。同时因为存储在MaxCompute内部，访问速度会比访问外部的Table Store更快，尤其适用于需要进行多次计算的热点数据。

**MaxCompute导出数据到Table Store**

说明：

MaxCompute不会主动创建外部的Table Store表，所以在对Table Store表进行数据输出之前，必须保证该表已经在Table Store上创建过（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Table Storeb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，现在希望对internal_orders中的数据进行一定处理后再写回 Table Store，可通过对外部表做**INSERT OVERWITE TABLE**操作来实现，如下所示：

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custk
ey, 'SHIPPED'), CEIL(odps_totalprice)
FROM internal_orders;
```

对于Table Store这种KV数据的NoSQL存储介质，从MaxCompute的输出将只影响相对应主键所在的行，比如示例中只影响所有odps_orderkey + odps_orderdate这两个主键值能对应行上的数据。而且在这些Tabele Store行上面，也只会去更新在创建External Table（ots_table_external）时指定的属性列，而不会去修改未在External Table中出现的数据列。

# 5 安全指南

## 5.1 目标用户

本章节文档主要面向 MaxCompute 项目空间所有者（Owner）、管理员以及对 MaxCompute 多租户数据安全体系感兴趣的用户。

MaxCompute 多租户的数据安全体系，主要包括如下内容：

- 用户认证。
- 项目空间的用户与授权管理。
- 跨项目空间的资源分享。
- 项目空间的数据保护。

## 5.2 快速开始

### 5.2.1 添加用户并授权

场景描述：Jack是项目空间prj1的管理员，一个新加入的项目组成员Alice（已拥有云账号: alice@aliyun.com）申请加入项目空间prj1，并申请如下权限：查看Table列表，提交作业，创建表。

操作步骤：（由项目空间管理员来操作）

```
    use prj1;
    add user aliyun$alice@aliyun.com;  --添加用户
    grant List, CreateTable, CreateInstance on project prj1 to user
aliyun$alice@aliyun.com; --使用grant语句对用户授权
```

### 5.2.2 添加角色并通过ACL授权

场景描述：Jack是项目空间prj1的管理员，有三个新加入的项目组成员：Alice, Bob, Charlie，他们的角色是数据审查员。他们要申请如下权限：查看Table列表，提交作业，读取表userprofile。

对于这个场景的授权，项目空间管理员可以使用基于对象的 *ACL授权* 机制来完成。

操作方法：

```
    use prj1;
    add user aliyun$alice@aliyun.com; --添加用户
    add user aliyun$bob@aliyun.com;
    add user aliyun$charlie@aliyun.com;
    create role tableviewer; --创建角色
    grant List, CreateInstance on project prj1 to role tableviewer;
 --对角色赋权
```

```
    grant Describe, Select on table userprofile to role tableviewer;
    grant tableviewer to aliyun$alice@aliyun.com; --对用户赋予角色
tableviewer
    grant tableviewer to aliyun$bob@aliyun.com;
    grant tableviewer to aliyun$charlie@aliyun.com;
```

### 5.2.3 设置项目保护模式

场景描述：Jack是项目空间prj1的管理员。该项目空间有很多敏感数据，比如用户身份号码和购物记录。而且还有很多具有自主知识产权的数据挖掘算法。Jack希望能将项目空间中的这些敏感数据和算法保护好，项目中用户只能在项目空间中访问，数据只能在项目空间内流动，不允许流出到项目空间之外。

操作方法：Jack需要如下操作

```
    use prj1;
    set ProjectProtection=true; --开启项目空间的数据保护机制
```

一旦当项目空间开启 项目空间的数据保护 机制后，无法将项目空间中的数据转移到项目空间之外，所有的数据只能在项目空间内部流动。

但是在某些情况下，可能由于业务需要，用户Alice需要将某些数据表导出到项目空间之外，并且也经过空间管理员的审核通过。针对这类情况，MaxCompute提供了一种机制来支持受保护项目空间的数据流出，即设置TrustedProject。将prj2设置为prj1的可信项目空间，设置后将允许prj1中的所有数据流出到prj2。

```
    use prj1;
    add trustedproject prj2;
```

## 5.3 用户认证

目前，MaxCompute 支持 云账号 和 **RAM** 账号 两种账号体系。

 说明：

MaxCompute 仅能识别 RAM 的账号体系，不能识别 RAM 的权限体系。即您可以将自身的任意 RAM 子账号加入 MaxCompute 的某一个项目中，但 MaxCompute 在对该 RAM 子账号做权限验证时，并不会考虑 RAM 中的权限定义。

在默认情况下，MaxCompute 项目仅能识别阿里云账号系统，您可以通过 `list accountproviders;`查看该项目所支持的账号系统。

通常情况下仅会看到阿里云账号，如果您想添加对 RAM 账号的支持，可以执行`add accountprovider ram;`。添加成功后，您可再次通过`list accountproviders;`查看所支持的账号系统是否有所变化。

### 申请云账号

如果您还没有云账号，请访问 阿里云官网，申请一个属于您的云账号。

**说明：**

申请云账号时，需要一个有效的电子邮箱地址，而且此邮箱地址将被当作云账号。比如：Alice 可以使用她的 alice@aliyun.com 邮箱来注册一个云账号，那么她的云账号就是 alice@aliyun.com。

### 申请 AccessKey

拥有云账号之后，您即可登录访问 *AccessKeys* 页面，以创建或管理当前云账号的 AccessKey 列表。

一个 AccessKey 由两部分组成：AccessKeyId 和 AccessKeySecret。AccessKeyId 用于检索 AccessKey，而 AccessKeySecret 用于计算消息签名，所以需要严格保护以防泄露。当一个 AccessKey 需要更新时，您可以创建一个新的 AccessKey，然后禁用老的 AccessKey。

### 使用云账号登录 MaxCompute

当使用 odpscmd 登录时，需要在配置文件 conf/odps_config.ini 中配置 AccessKey 的相关信息，如下所示：

```
project_name=myproject
access_id=<这里输入Access ID，不带尖括号>
access_key=<这里输入Access Key，不带尖括号>
end_point=http://service.odps.aliyun-inc.com/api
```

**说明：**

在阿里云网站上禁用或解禁一个 AccessKey 时，目前需要 15 分钟后才能完全生效。

## 5.4 用户管理

任意非项目空间 Owner 用户必须被加入 MaxCompute 项目空间中，并被授予相对应权限，方能操作 MaxCompute 中的数据、作业、资源及函数。本文将介绍项目空间 Owner 如何将其他用户（包括 RAM 子账号）加入和移出 MaxCompute，如何给用户授权。

如果您是项目空间 Owner，建议您仔细阅读本文；如果您是普通用户，建议您向 Owner 提出申请，被加入对应的项目空间后再阅读后续章节。

本文的操作均在客户端运行，Linux 系统下运行 **./bin/odpscmd**，Windows 下运行 **./bin/odpscmd.bat**。

### 添加用户

当项目空间的 Owner Alice 决定对另一个用户授权时，Alice 需要先将该用户添加到自己的项目空间中来，**只有添加到项目空间中的用户才能够被授权**。

添加用户的命令如下：

```
add user
```

云账号的<username>既可以是在 www.aliyun.com 上注册过的有效邮箱地址，也可以是执行此命令的云账号的某个 RAM 子账号，示例如下：

```
add user ALIYUN$odps_test_user@aliyun.com;
add user RAM$ram_test_user;
```

假设 Alice 的云账号为 alice@aliyun.com，那么当 Alice 执行上述两条语句后，通过`list users;`命令即可看到如下结果：

```
RAM$alice@aliyun.com:ram_test_user
ALIYUN$odps_test_user@aliyun.com
```

这表明云账号 odps_test_user@aliyun.com 以及 Alice 通过 RAM 创建的子账号 ram_test_user 已经被加入到了该项目空间中。

### 添加 RAM 子账号

添加 RAM 子账号有以下两种方式：

- 通过DataWorks进行操作，详情请参见 如何添加成员及授权。
- 通过 MaxCompute 客户端常用命令进行操作，详情如下：

> **说明：**
>
> - MaxCompute 只允许主账号将自身的 RAM 子账号加入到项目空间中，不允许加入其它云账号的 RAM 子账号，因此在`add user`时，无需在 RAM 子账号前指定主账号名称，MaxCompute 默认判定命令的执行者即是子账号对应的主账号。

- MaxCompute 只能够识别 RAM 的账号体系，不能识别 RAM 的权限体系。即用户可以将自身的任意 RAM 子账号加入 MaxCompute 的某一个项目中，但 MaxCompute 在对该 RAM 子账号做权限验证时，并不会考虑 RAM 中的权限定义。

默认情况下，MaxCompute 项目只能够识别阿里云账号系统，用户可以通过`list accountproviders;`命令查看该项目所支持的账号系统，通常情况下仅会看到 ALIYUN 账号，例如下所示：

```
odps@ ****>list accountproviders;
ALIYUN
```

**说明：**

只有项目空间的Owner 有权限进行`accountproviders`的相关操作。

由上可见，只能看到`ALIYUN`账号体系，如果想添加对 RAM 账号的支持，可以执行`add accountprovider ram;`如下所示：

```
odps@ odps_pd_inter>add accountprovider ram;
OK
```

添加用户成功后，此用户仍不能操作 MaxCompute，需要对用户授予一定权限，用户才能在所拥有的权限范围内操作 MaxCompute。关于授权的更多操作，请参见 授权。

### 用户授权

添加用户后，项目空间 Owner 或者项目空间管理员需要给该用户进行授权，只有用户获得权限后，才能执行操作。

MaxCompute 提供了 ACL 授权，跨项目空间数据分享及项目空间数据保护等多种策略。下面列举两个常见场景，更多详情请参见 *ACL*授权。

**场景一：**

假设 Jack 是项目空间 prj1 的管理员，一个新加入的项目组成员 Alice（已拥有云账号：alice@aliyun.com）申请加入项目空间 prj1，并申请查看 Table 列表，提交作业和创建表的权限。项目空间的 admin role 或者该项目空间 owner 在客户端执行如下命令：

```
use prj1; --进入项目空间 prj1
add user aliyun$alice@aliyun.com; --添加用户
grant List, CreateTable, CreateInstance on project prj1 to user aliyun
$alice@aliyun.com; --使用grant语句对用户授权
```

**场景二：**

假设用户云账号为 bob@aliyun.com，已经被添加到某个项目空间（$user_project_name），需要给它授予建表、获取表信息和执行的权限。

项目空间的 admin role或者该项目空间 owner 在客户端可以执行如下命令：

```
grant CreateTable on PROJECT $user_project_name to USER ALIYUN$bob@
aliyun.com;
  -- 向 bob@aliyun.com 授予名为“$user_project_name”的 project 的
CreateTable（创建表）权限
grant Describe on Table $user_table_name to USER ALIYUN$bob@aliyun.com
;
  -- 向 bob@aliyun.com 授予名为“$user_table_name”的 Table 的 Describe
（获取表信息）权限
grant Execute on Function $user_function_name to USER ALIYUN$bob@
aliyun.com;
  -- 向 bob@aliyun.com 授予名为 "$user_function_name" 的 Function 的
Execute（执行）权限
```

**给 RAM 子账号授权**

通过`list accountproviders;`来查看账号支持情况，如下所示：

```
odps@ ****>list accountproviders;
ALIYUN, RAM
```

由上可见，这个项目空间已经能够支持RAM账号体系，即可以向这个项目空间添加 RAM 子账号并授予某张表的`Describe`权限，如下所示：

```
odps@ ****>add user ram$bob@aliyun.com:Alice;
OK: DisplayName=RAM$bob@aliyun.com:Alice
odps@ ****>grant Describe on table src to user ram$bob@aliyun.com:
Alice;
OK
```

此时，*bob@aliyun.com*的 RAM 子账号*Alice*就可以通过自己的**AccessKeyID**及**AccessKeySecret**登录 MaxCompute ，并对表*src*进行`desc`操作。

**说明：**

- 获取 RAM 子账号`AccessKeyID`及`AccessKeySecret`的相关操作请参见 *RAM* 介绍。
- 更多 MaxCompute 添加/删除用户的操作请参见本文相关内容。
- 更多有关授权的操作请参见 授权。

**删除用户**

当一个用户离开此项目团队时，Alice 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，该用户将不再拥有任何访问项目空间资源的权限。

移除用户的命令，如下所示：

```
remove user
```

说明：

- 当一个用户被移除后，该用户不再拥有访问该项目空间资源的任何权限。
- 移除一个用户之前，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。关于角色的介绍请参考 角色管理。
- 当一个用户被移除后，与该用户有关的 *ACL* 授权 仍然会被保留。一旦该用户以后被再添加到该项目空间时，该用户的历史的 ACL 授权访问权限将被重新激活。
- MaxCompute 目前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。

Alice 执行下述两条命令，即可移除相关用户：

```
remove user ALIYUN$odps_test_user@aliyun.com;
remove user RAM$ram_test_user;
```

查看用户是否移除，命令如下：

```
LIST USERS;
```

查看结果将不会看到这两个账号。此时，表明这两个账号已经被移出项目空间。

### 删除 RAM 子账号

同样，您也可以通过`remove user`命令删除自身的 RAM 子账号。示例如下：

```
odps@ ****>revoke describe on table src from user ram$bob@aliyun.com:
Alice;
OK
-- 回收子账号 Alice 权限
odps@ ****>remove user ram$bob@aliyun.com:Alice;
Confirm to "remove user ram$bob@aliyun.com:Alice;" (yes/no)? yes
OK
-- 删除子账号
```

如果您是项目空间的 owner，也可以通过`remove accountprovider`将 RAM 账号系统从当前项目中删除，如下所示：

```
odps@ ****>remove accountprovider ram;
Confirm to "remove accountprovider ram;" (yes/no)? yes
OK
odps@ ****>list accountproviders;
```

```
ALIYUN
```

## 5.5 角色管理

角色（Role）是一组访问权限的集合，当需要对一组用户赋予相同的权限时，可以使用角色来授权。基于角色的授权可以大大简化授权流程，降低授权管理成本。当需要对用户授权时，应当优先考虑是否应该使用角色来完成。

每一个项目空间在创建时，会自动创建一个 admin 的角色，并且为该角色授予了确定的权限：可以访问项目空间内的所有对象、对用户或角色进行管理、对用户或角色进行授权。与项目空间 Owner 相比，admin 角色不能将 admin 权限指派给用户，不能设定项目空间的安全配置，不能修改项目空间的鉴权模型，admin 角色所对应的权限不能被修改。

角色管理相关命令如下：

```
create role <rolename> --创建角色
drop role <rolename> --删除角色
grant <rolename> to <username> --给用户指派某种角色
revoke <rolename> from <username> --撤销角色指派
```

说明：

- 多个用户可以同时存在于一个角色下，一个用户也可以隶属于多个角色。
- DataWorks中成员角色类型对应的 MaxCompute 角色，以及各角色的平台权限详情，请参见 项目管理中的项目成员管理模块。

**创建角色**

创建角色的命令格式，如下所示：

```
CREATE ROLE ;
```

示例如下：

假设要创建一个 player 角色，需在客户端输入如下命令：

```
create role player;
```

### 添加用户到角色

添加用户到角色的命令格式，如下所示：

```
GRANT <roleName> TO <full_username> ;
```

示例如下：

假设要将用户 bob@aliyun.com 加入 player 角色中，需在客户端输入如下命令：

```
grant player to bob@aliyun.com;
```

### 给角色授权

给角色授权的语句与给用户授权相似，更多详情请参见 用户授权。

**说明：**

给角色授权后，该角色下的所有用户拥有相同的权限。

示例如下：

假设 Jack 是项目空间 prj1 的管理员，有三个新加入的项目组成员：Alice，Bob 和 Charlie，他们的角色是数据审查员。他们要申请如下权限：查看 Table 列表，提交作业和读取表 userprofile。

对于这个场景的授权，项目空间管理员可以使用基于对象的 *ACL*授权机制来完成。

操作如下：

```
    use prj1;
    add user aliyun$alice@aliyun.com; --添加用户
    add user aliyun$bob@aliyun.com;
    add user aliyun$charlie@aliyun.com;
    create role tableviewer; --创建角色
    grant List, CreateInstance on project prj1 to role tableviewer;
 --对角色赋权
    grant Describe, Select on table userprofile to role tableviewer;
    grant tableviewer to aliyun$alice@aliyun.com; --对用户赋予角色
tableviewer
    grant tableviewer to aliyun$bob@aliyun.com;
```

```
grant tableviewer to aliyun$charlie@aliyun.com;
```

删除角色中的用户

删除角色中的用户的命令格式，如下所示：

```
REVOKE <roleName> FROM <full_username>;
```

示例如下：

假设将用户 bob@aliyun.com 从 player 角色中删除，需在客户端输入如下命令：

```
revoke player from bob@aliyun.com;
```

删除角色

删除角色的命令格式，如下所示：

```
DROP ROLE <roleName>;
```

示例如下：

假设要删除 player 角色，需在客户端输入如下命令：

```
drop role player;
```

说明：

删除一个角色时，MaxCompute 会检查该角色内是否还存在其他用户。若存在，则删除该角色失败。只有在该角色的所有用户都被撤销时，删除角色才会成功。

## 5.6 授权

添加用户 后，项目空间 Owner 或者项目空间管理员需要给该用户进行授权，只有用户获得权限后，才能执行操作。所谓授权，即授予用户对 MaxCompute 中的表，任务，资源等客体的某种操作权限，包括：读、写、查看等。

MaxCompute 提供了 ACL 授权，跨项目空间数据分享及项目空间数据保护等多种策略。授权操作一般涉及到三个要素：主体（Subject，可以是用户也可以是角色），客体（Object）和操作（Action）。在 MaxCompute 中，主体是指用户或角色，客体是指项目空间中的各种类型对象。

ACL 授权中，MaxCompute 的客体包括：*Project*，*Table*，*Function*，*Resource*，*Instance*，操作与特定对象类型有关，不同类型的对象支持的操作也不尽相同。

MaxCompute 项目空间支持如下的对象类型及操作：

| 客体（**Object**） | 操作（**Action**） | 说明 |
| --- | --- | --- |
| Project | Read | 查看项目空间自身（不包括项目空间的任何对象）的信息，如 CreateTime 等 |
| Project | Write | 更新项目空间自身（不包括项目空间的任何对象）的信息，如 Comments |
| Project | List | 查看项目空间所有类型的对象列表 |
| Project | CreateTable | 在项目空间中创建 Table |
| Project | CreateInstance | 在项目空间中创建 Instance |
| Project | CreateFunction | 在项目空间中创建 Function |
| Project | CreateResource | 在项目空间中创建 Resource |
| Project | All | 具备上述所有权限 |
| Table | Describe | 读取 Table 的元信息 |
| Table | Select | 读取 Table 的数据 |
| Table | Alter | 修改 Table 的元信息，添加删除分区 |
| Table | Update | 覆盖或添加 Table 的数据 |
| Table | Drop | 删除 Table |
| Table | All | 具备上述所有权限 |
| Function | Read | 读取，及执行权限 |
| Function | Write | 更新 |
| Function | Delete | 删除 |
| Function | Execute | 执行 |
| Function | All | 具备上述所有权限 |
| Resource | Read | 读取 |
| Resource | Write | 更新 |
| Resource | Delete | 删除 |
| Resource | All | 具备上述所有权限 |
| Instance | Read | 读取 |
| Instance | Write | 更新 |
| Instance | All | 具备上述所有权限 |

说明：

- 上述权限描述中 Project 类型对象的 CreateTable 操作，Table 类型的 Select、Alter、Update、Drop 操作需要与 Project 对象的 CreateInstance 操作权限配合使用。
- 单独使用上述几种权限而没有指派 CreateInstance 权限是无法完成对应操作的。这与 MaxCompute 的内部实现相关。同样，Table 的 Select 权限也要与 CreateInstance 权限配合使用，当跨项目操作如在项目A里select项目B的table，则需要有项目A的CreateInstance和项目B的table select权限 。
- 在添加用户或创建角色之后，需要对用户或角色进行授权 。MaxCompute 授权是一种基于对象的授权。通过授权的权限数据（即访问控制列表, Access Control List）被看作是该对象的一种子资源。只有当对象已经存在时，才能进行授权操作。当对象被删除时，通过授权的权限数据会被自动删除。

- 类似**SQL92**的授权方法

  MaxCompute 支持的授权方法类似于 SQL92 定义的 GRANT/REVOKE 语法，它通过简单的授权语句来完成对已存在的项目空间对象的授权或撤销授权。授权语法如下：

```
grant actions on object to subject
revoke actions on object from subject
actions ::= action_item1, action_item2, ...
object ::= project project_name | table schema_name |
           instance inst_name | function func_name |
            resource res_name
subject ::= user full_username | role role_name
```

  熟悉 SQL92 定义的 GRANT/REVOKE 语法或者熟悉 Oracle 数据库安全管理的用户容易发现，MaxCompute 的 ACL 授权语法并不支持 [WITH GRANT OPTION] 授权参数。也就是说，当用户 A 授权用户 B 访问某个对象时，用户 B 无法将权限进一步授权给用户 C。那么，所有的授权操作都必须由具有以下三种身份之一的用户来完成：

  - 项目空间 Owner。
  - 项目空间中拥有 admin 角色的用户。
  - 项目空间中对象创建者。

- 使用 **ACL** 授权的应用示例

  假设云账号用户 alice@aliyun.com 是新加入到项目空间 test_project_a 的成员，Allen 是加入到 bob@aliyun.com 中的 RAM 子账号。在 test_project_a 中，他们需要提交作业、创建数据表、查看项目空间已存在的对象。

管理员执行的授权操作，如下所示：

```
    use test_project_a; --打开项目空间
    add user aliyun$alice@aliyun.com; --添加用户
    add user ram$bob@aliyun.com:Allen; --添加RAM子账号
    create role worker; --创建角色
    grant worker TO aliyun$alice@aliyun.com; --角色指派
    grant worker TO ram$bob@aliyun.com:Alice;  --角色指派
    grant CreateInstance, CreateResource, CreateFunction, CreateTabl
e, List ON PROJECT test_project TO ROLE worker; --对角色授权
```

- 跨项目空间**Table**、**Resource**、**Function**分享示例

  接前一个示例，aliyun$alice@aliyun.com 和 ram$bob@aliyun.com:Allen 在test_project_a 拥有了一定的权限后，这两个用户还需查询test_project_b中的table prj_b_test_table，且需要用到test_project_b中的UDF prj_b_test_udf。

  test_project_b的管理员执行的授权操作如下：

```
    use test_project_b; --打开项目空间
    add user aliyun$alice@aliyun.com; --添加用户
    add user ram$bob@aliyun.com:Allen; --添加RAM子账号
    create role prj_a_worker; --创建角色
    grant prj_a_worker TO aliyun$alice@aliyun.com; --角色指派
    grant prj_a_worker TO ram$bob@aliyun.com:Alice;  --角色指派
    grant Describe , Select  ON TABLE prj_b_test_table TO ROLE
prj_a_worker; --对角色授予table权限
    grant Read  ON Function prj_b_test_udf TO ROLE prj_a_worker;
 --对角色授予udf权限
    grant Read  ON Resource prj_b_test_udf_resource TO ROLE
prj_a_worker; --对角色授予实现udf的Resource的权限
--授权后，这两个用户在test_project_a中查询表和使用udf的方式如下：
use test_project_a;
select test_project_b:prj_b_test_udf(arg0, arg1) as res from
test_project_b.prj_b_test_table;
```

说明：

若是在test_project_a 中创建udf，则授权时只需进行Resource授权后，使用写法如下：

```
create function function_name as 'com.aliyun.odps.compiler.udf.
PlaybackJsonShrinkUdf' using 'test_project_b/resources/odps-compiler-
playback.jar' -f;
```

## 5.7 权限查看

MaxCompute 支持从多种维度查看权限，具体包括查看指定用户的权限、查看指定角色的权限、以及查看指定对象的授权列表。

在展现用户权限或角色权限时，MaxCompute 使用了如下的标记字符：A、C、D、G，它们的含义如下：

- A：表示 Allow，即允许访问。
- D：表示 Deny，即拒绝访问。
- C：表示 with Condition，即为带条件的授权，只出现在 policy 授权体系中。
- G：表示 with Grant option，即可以对 object 进行授权。

展现权限的示例如下：

```
    odps@test_project> show grants for aliyun$odpstest1@aliyun.com;
    [roles]
    dev
    Authorization Type: ACL
    [role/dev]
    A       projects/test_project/tables/t1: Select
    [user/odpstest1@aliyun.com]
    A       projects/test_project: CreateTable | CreateInstance |
CreateFunction | List
    A       projects/test_project/tables/t1: Describe | Select
    Authorization Type: Policy
    [role/dev]
    AC      projects/test_project/tables/test_*: Describe
    DC      projects/test_project/tables/alifinance_*: Select
    [user/odpstest1@aliyun.com]
    A       projects/test_project: Create* | List
    AC      projects/test_project/tables/alipay_*: Describe | Select
    Authorization Type: ObjectCreator
    AG      projects/test_project/tables/t6: All
    AG      projects/test_project/tables/t7: All
```

查看指定用户的权限

```
    show grants; --查看当前用户自己的访问权限
    show grants for <username>; --查看指定用户的访问权限，仅由 ProjectOwner
 和 Admin 才能有执行权限 。
```

示例如下：

假设需要查看用户云账号 bob@aliyun.com 在当前项目空间的权限，需要在客户端可以执行如下命令：

```
show grants for ALIYUN$bob@aliyun.com;
```

查看RAM子帐号权限：

```
show grants for RAM$主帐号:子帐号;
```

示例如下：

```
show grants for RAM$bob@aliyun.com:Alice;
```

**查看指定角色的权限**

```
describe role ; --查看指定角色的访问权限角色指派
```

**查看指定对象的授权列表**

```
show acl for  [on type ];--查看指定对象上的用户和角色授权列表
```

说明：

当省略 *[on type <objectType>]* 时，默认的 type 为 Table。

## 5.8 项目空间的安全配置

MaxCompute 是一个支持多租户的数据处理平台，不同的租户对数据安全需求不尽相同。为了满足不同租户对数据安全的灵活需求，MaxCompute 支持项目空间级别的安全配置，ProjectOwner 可以定制适合自己的外部账号支持和鉴权模型。

MaxCompute 支持多种正交的授权机制，如 ACL 授权、隐式授权（如对象创建者自动被赋予访问对象的权限）。但是并非所有用户都需要使用这些安全机制，您可以根据自己的业务安全需求或使用习惯，合理设置本项目空间的鉴权模型。

```
    show SecurityConfiguration
      --查看项目空间的安全配置
    set CheckPermissionUsingACL=true/false
      --激活/冻结ACL授权机制，默认为true
    set ObjectCreatorHasAccessPermission=true/false
      --允许/禁止对象创建者默认拥有访问权限，默认为true
    set ObjectCreatorHasGrantPermission=true/false
      --允许/禁止对象创建者默认拥有授权权限，默认为true
    set ProjectProtection=true/false
```

```
--开启/关闭项目空间的数据保护机制，禁止/允许数据流出项目空间
```

> 说明：
>
> 您也可以通过DataWorks进行可视化操作，完成项目空间的相关安全配置，详情请参见 项目配置。

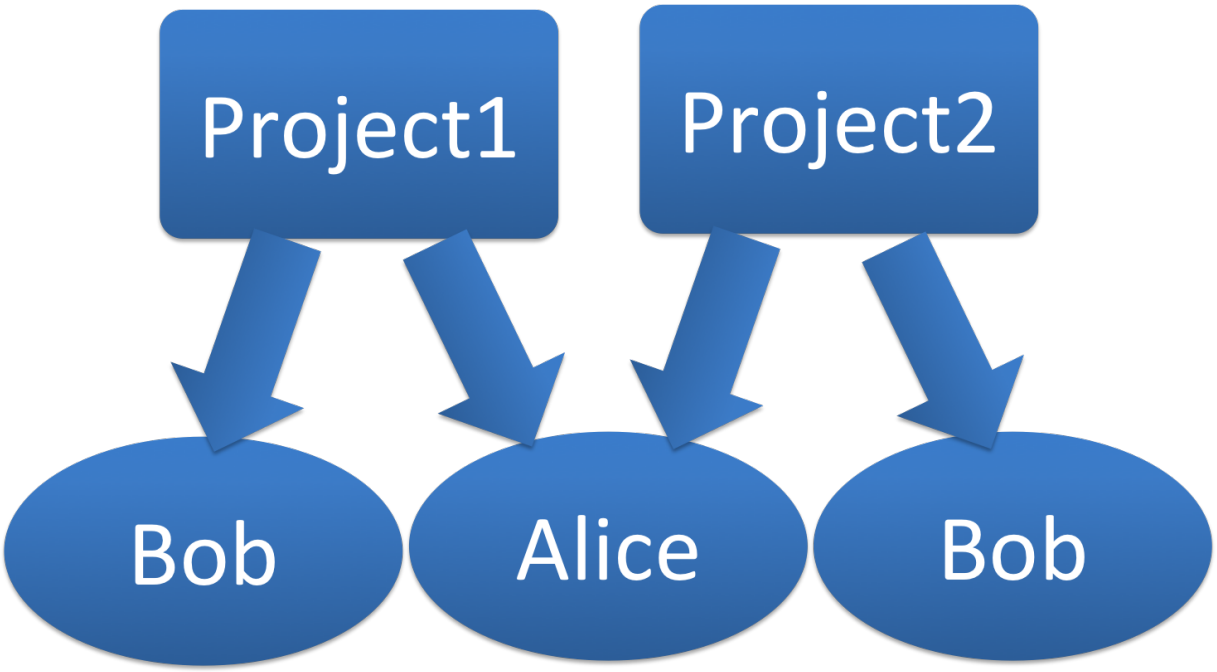
## 5.9 项目空间的数据保护

**背景和动机**

在现实中，有一些公司(比如金融机构、军工企业)对数据安全非常敏感，比如不允许员工将工作带回家，而只允许在公司内部进行操作。而且公司的所有电脑上的USB存储接口也都是禁用的。这样做的目的是禁止员工将敏感数据泄漏出去。

作为MaxCompute项目空间管理员，您是否也有类似的安全需求呢？——“不允许用户将数据转移到项目空间之外”。

比如，当项目空间prj1的Owner遇到下图所示的这种情形时，是否会担心用户Alice将她能访问的数据转移到prj2中去呢？因为她可以访问prj2，而prj2又不受控制。



更具体地说，假设Alice已经被您授予了访问myprj.table1的Select权限，同时Alice也被prj2的管理员授予了CreateTable的权限。

那么Alice将可以通过如下的任何一种方法将数据转移到prj2中去：

- 提交SQL：

```
create table prj2.table2 as select * from myprj.table1;
```

- 编写MapReduce将myprj.table1读出，并写入prj2.table2。

如果您项目空间中的数据非常敏感，绝对不允许流出到其他项目空间中去，您希望MaxCompute能将上述导致数据流出的操作统统禁止，可以吗？没问题。

**数据保护机制**

MaxCompute提供的项目空间保护机制正好可以满足上述需要。您只需要在您的项目空间中做如下设置：

```
    set projectProtection=true
      --设置ProjectProtection规则：数据只能流入，不能流出
```

设置ProjectProtection后，您的项目空间中的数据流向就会得到控制 ——“数据只能流入，不能流出”。即上述的两种操作将失效，因为它们都违背了ProjectProtection规则。

默认时，ProjectProtection不会被设置，值为false。

同时在多个项目空间中拥有访问权限的用户就可以自由地使用任意支持跨Project的数据访问操作来转移项目空间的数据。如果对项目空间中的数据高度敏感，则需要管理员自行设置ProjectProtection保护机制。

**开启数据保护机制后的数据流出方法**

在您的项目空间被设置了ProjectProtection之后，您可能很快就会遇到这样的需求：Alice向您提出申请，她的确需要将某张表的数据导出您的项目空间。

而且经过您的审查之后，那张表也的确没有泄漏您关心的敏感数据。为了不影响Alice的正常业务需要，MaxCompute为您提供了在ProjectProtection被设置之后的两种数据导出途径。

- 设置**TrustedProject**

  若当前项目空间处于受保护状态，如果将数据流出的目标空间设置为当前空间的TrustedProject，

那么向目标项目空间的数据流向将不会被视为触犯ProjectProtection规则。如果多个项目空间之间两两互相设置为TrustedProject，

那么这些项目空间就形成了一个TrustedProject Group，数据可以在这个Project Group内流动，但禁止流出到Project Group之外。

管理TrustedProject的命令如下：

```
list trustedprojects;
  --查看当前project中的所有TrustedProjects
add trustedproject <projectname>;
  --在当前project中添加一个TrustedProject
remove trustedproject <projectname>;
  --在当前project中移除一个TrustedProject
```

- 资源分享与数据保护

  在MaxCompute中，基于*package*的资源分享机制与ProjectProtection数据保护机制是正交的，但在功能上却是相互制约的。

  MaxCompute规定：资源分享优先于数据保护。换句话说，如果一个数据对象是通过资源分享方式授予其他项目空间用户访问，那么该数据对象将不受ProjectProtection规则的限制。

关于最佳实践

如果要防止数据从项目空间的流出，在设置`ProjectProtection=true`之后，还需检查如下配置：

- 确保没有添加trustedproject。如果有设置，则需要评估可能的风险；
- 确保没有使用package数据分享。如果有设置，则需要确保package中没有敏感数据。

# 5.10 跨项目空间的资源分享

## 5.10.1 基于Package的跨项目空间的资源分享

假设您是项目空间的Owner或管理员（admin角色），某个主账号下多个项目空间，其中项目空间prj1里有一批资源（包括tables、Resources、自定义functions）可以分享给其他项目空间使用，但是若把其他项目空间的user都add到这个prj1项目空间并逐个进行授权操作，需要操作非常多的步骤，对于prj1项目空间来说引入很多跟本项目业务无关（假设存在）的user非常不方便管理。

那么你可以使用本节介绍的跨项目空间的资源分享功能。

如果资源需要精细控制单人使用，且申请人是本业务项目团队成员，那么建议你使用项目空间的用户与授权管理功能：项目空间的用户与授权管理

Package是一种跨项目空间共享数据及资源的机制，主要用于解决跨项目空间的用户授权问题。

如果不使用Package，对于下面的场景我们无法有效的解决：

Alifinance项目空间的成员若要访问Alipay项目空间的数据，则需要Alipay项目空间管理员执行繁琐的授权操作：首先需要将Alifiance项目空间中的用户添加到Alipay项目空间中，再分别对这些新加入的用户进行普通授权。

实际上，Alipay项目空间管理员并不期望对Alifiance项目空间中的每个用户都进行授权管理，而更期望有一种机制能使得Alifiance项目空间管理员能对许可的对象进行自主授权控制。

使用Package之后，Alipay项目空间管理员可以对Alifinance需要使用的对象进行打包授权(也就是创建一个Package)，然后许可Alifinance项目空间可以安装这个Package。在Alifinance项目空间管理员安装Package之后，就可以自行管理Package是否需要进一步授权给自己Project下的用户。

## 5.10.2 Package的使用方法

**Package的使用方法**

Package的使用涉及到两个主体：Package创建者和Package使用者。

- Package创建者项目空间是资源提供方，它将需要分享的资源及其访问权限进行打包，然后许可Package使用方来安装使用。
- Package使用者项目空间是资源使用方，它在安装资源提供方发布的Package之后，便可以直接跨项目空间访问资源。

下面分别介绍Package创建者和Package使用者所涉及的操作。

**Package创建者**

- 创建**Package**

```
create package ;
```

说明：

- 仅project的owner有权限进行该操作。
- 目前创建的package名称不能超过128个字符。

- 将分享的资源添加到**Package**

```
    add project_object to package package_name [with privileges
privileges];--将对象添加到Package
    remove project_object from package package_name;--将对象从Package
移除
    project_object ::= table table_name |
                   instance inst_name |
                   function func_name |
                   resource res_name
    privileges ::= action_item1, action_item2, ...
```

说明：

- 目前支持的对象类型不包括Project类型，也就是不允许通过Package在其他Project中创对象。
- 添加到Package中的不仅仅是对象本身，还包括相应的操作权限。当没有通过[with privileges privileges]来指定操作权限时，默认为只读权限，即Read/Describe/Select。“对象及其权限”被看作一个整体，添加后不可被更新。若有需要，只能删除和重新添加。

- 许可其他项目空间使用**Package**

```
allow project <prjname> to install package <pkgname> [using label <
number>];
```

- 撤销其他项目空间使用**Package**的许可

```
disallow project <prjname> to install package <pkgname>;
```

- 删除**Package**

```
delete package <pkgname>;
```

- 查看已创建和已安装的**Package**列表

```
show packages;
```

- 查看**Package**详细信息

```
describe package <pkgname>;
```

**Package**使用者

- 安装**Package**

```
install package <pkgname>;
```

对于安装Package来说，要求pkgName的格式为：`<projectName>.<packageName>`。

**说明：**

仅project的Owner有权限进行该操作。

- 卸载**Package**

```
uninstall package <pkgname>;
```

对于卸载Package来说，要求pkgName的格式为：`<projectName>.<packageName>`

- 查看**Package**

```
    show packages;
    --查看已创建和已安装的package列表
    describe package <pkgname>;
    --查看package详细信息
```

- 使用方项目给本项目其他成员授权访问**Package**

被安装的Package是一种独立的MaxCompute对象类型。若要访问Package里的资源(即其他项目空间分享给您的资源)，您必须拥有对该Package的Read权限。

如果请求者没有Read权限，则需要向ProjectOwner或Admin申请。ProjectOwner或Admin可以通过ACL授权机制来完成。

比如，如下的ACL授权允许云账号用户LIYUN$odps_test@aliyun.com访问Package里的资源：

```
    use prj2;
    install package prj1.testpkg;
    grant read on package prj1.testpackage to user aliyun$odps_test@
aliyun.com;
```

**场景示例**

场景描述：Jack是项目空间prj1的管理员。John是项目空间prj2的管理员。由于业务需要， Jack希望将其项目空间prj1中的某些资源(比如datamining.jar及sampletable表)分享给John的项目空间prj2。 如果项目空间prj2的用户Bob需要访问这些资源，那么管理员John可以通过ACL给Bob自主授权，无需Jack参与。

操作步骤：

1. 项目空间prj1的管理员Jack在项目空间prj1中创建资源包(package)。

```
    use prj1;
    create package datamining; --创建一个package
    add resource datamining.jar to package datamining; --添加资源到
package
    add table sampletable to package datamining; --添加table到package
```

```
    allow project prj2 to install package datamining; --将package分享
给项目空间prj2
```

2. 项目空间prj2管理员Jhon在项目空间prj2中安装package。

```
    use prj2;
    install package prj1.datamining; --安装一个package
    describe package prj1.datamining; --查看package中的资源列表
```

3. Jhon对package给Bob进行自主授权。

```
    use prj2;
    grant Read on package prj1.datamining to user aliyun$bob@aliyun.
com; --通过ACL授权Bob使用package
```

## 5.11 列级别访问控制

基于标签的安全（LabelSecurity）是项目空间级别的一种强制访问控制策略（Mandatory Access Control, MAC），它的引入是为了让项目空间管理员能更加灵活地控制用户对列级别敏感数据的访问。

**理解MaxCompute中的MAC与DAC差异**

在MaxCompute中，强制访问控制机制(MAC)独立于自主访问控制机制(DAC)。为了便于理解，MAC与DAC的关系可以用下面的例子来做个类比。

对于一个国家来说（类比MaxCompute的一个项目空间），这个国家公民要想开车(类比读数据操作)，必须先申请获得驾照(类比申请SELECT权限)。这些就属于DAC考虑的范畴。

但由于这个国家交通事故率一直居高不下，于是该国新增了一条法律：禁止酒驾。此后，所有想开车的人除了持有驾照之外，还必须不能喝酒。类比MaxCompute，这个禁止酒驾就相当于禁止读取敏感度高的数据。这就属于MAC考虑的范畴。

**数据的敏感等级分类**

LabelSecurity需要将数据和访问数据的人进行安全等级划分。在政府和金融机构，一般将数据的敏感度标记分为四类：0级 (不保密, Unclassified), 1级 (秘密, Confidential), 2级 (机密, Sensitive), 3级 (高度机密, Highly Sensitive)。MaxCompute也遵循这一分类方法。ProjectOwner需要定义明确的数据敏感等级和访问许可等级划分标准，默认时所有用户的访问许可等级为0级，数据安全级别默认为0级。

LabelSecurity对敏感数据的粒度可以支持列级别，管理员可以对表的任何列设置敏感度标记(Label)，一张表可以由不同敏感等级的数据列构成。

对于view，也支持和表同样的设置，即管理员可以对view设置label等级，view的等级和它对应的基表的label等级是独立的，在view创建时，默认的等级也是0。

**LabelSecurity默认安全策略**

在对数据和user分别设置安全等级标记之后，LabelSecurity的默认安全策略如下：

- (No-ReadUp) 不允许user读取敏感等级高于用户等级的数据，除非有显式授权。
- (Trusted-User) 允许user写任意等级的数据，新创建的数据默认为0级（不保密）。

**说明：**

- 在一些传统的强制访问控制系统中，为了防止数据在项目空间内部的任意分发，一般还支持更多复杂的安全策略，比如不允许用户写敏感等级不高于用户等级的数据(No-WriteDown)。但在MaxCompute平台中，考虑到项目空间管理员对数据敏感等级的管理成本，默认安全策略并不支持No-WriteDown. 如果项目空间管理员有类似需求，可以通过修改项目空间安全配置(`Set ObjectCreatorHasGrantPermission=false`)以达到控制目的。
- 如果是为了控制数据在不同项目空间之间的流动，则可以将项目空间设置为受保护状态(ProjectProtection)。设置之后，只允许用户在项目空间内访问数据，这样可以有效防止数据流出道项目空间之外。

项目空间中的LabelSecurity安全机制默认是关闭的，ProjectOwner可以自行开启。

LabelSecurity安全机制一旦开启，上述的默认安全策略将被强制执行。当用户访问数据表时，除了必须拥有Select权限外，还必须获得读取敏感数据的相应许可等级。或者说，通过LabelSecurity只是通过CheckPermission的必要而非充分条件。

**LabelSecurity操作**

- **LabelSecurity机制的开/关**

```
Set LabelSecurity=true|false;
  --开启或关闭LabelSecurity机制，默认为false。
  --此操作必须由ProjectOwner完成，其他操作则可以由Admin角色完成。
```

- **给user设置安全许可标签**

```
SET LABEL <number> TO USER <username>; -- number取值范围:[0, 9]，该操作只能由ProjectOwner或Admin角色完成
--举例:
ADD USER aliyun$yunma@aliyun.com; --添加用户，默认的安全许可标签为0级
ADD USER ram$yunma@aliyun.com:Allen;--添加yunma@aliyun.com的RAM子账号用户Allen
```

```
 SET LABEL 3 TO USER aliyun$yunma@aliyun.com;
   --设置yunma的安全许可标签为3级，他能访问敏感等级不超过3级的数据
 SET LABEL 1 TO USER ram$yunma@aliyun.com:Allen;
   --设置yunma的子账号Allen的安全许可标签为1级，他能访问敏感等级不超过1级的数据
```

- 给数据设置敏感等级标签

```
 SET LABEL <number> TO TABLE tablename[(column_list)]; -- number取值范围:[0, 9]，该操作只能由ProjectOwner或Admin角色完成
 --举例:
 SET LABEL 1 TO TABLE t1; --设置表t1的label为1级
 SET LABEL 2 TO TABLE t1(mobile, addr); --将t1的mobile,addr两列的label设置为2级
 SET LABEL 3 TO TABLE t1; --设置表t1的label为3级. 注意此时mobile,addr两列的label仍为2级
```

说明：

从上面例子可以看出，显式地对列设置的标签会覆盖对表设置的标签，而不会考虑标签设置的顺序以及敏感等级的高低。

- 显式授权低级别用户访问特定的高敏感级数据表

```
 --授权：
 GRANT LABEL <number> ON TABLE <tablename>[(column_list)] TO USER <username> [WITH EXP <days>]; --默认过期时间是180天
 --撤销授权：
 REVOKE LABEL ON TABLE <tablename>[(column_list)] FROM USER <username>;
 --清理过期的授权：
 CLEAR EXPIRED GRANTS;
 --举例：
 GRANT LABEL 2 ON TABLE t1 TO USER ram$yunma@aliyun.com:Allen WITH EXP 1; --显式授权Allen访问t1表中敏感度不超过2级的数据，授权有效期为1天
 GRANT LABEL 3 ON TABLE t1(col1, col2) TO USER ram$yunma@aliyun.com:Allen WITH EXP 1; --显式授权alice访问t1(col1, col2)中敏感度不超过3级的数据，授权有效期为1天
 REVOKE LABEL ON TABLE t1 FROM USER ram$yunma@aliyun.com:Allen; --撤销alice对t1表的敏感数据访问
```

说明：

目前取消用户对table的label权限，会同时取消该用户对table字段的label的权限

- 查看一个用户能访问哪些敏感数据集

```
 SHOW LABEL [<level>] GRANTS [FOR USER <username>];
   --省略 [FOR USER <username>]时，查看当前用户所能访问的敏感数据集
```

```
   --省略<level>时，将显示所有label等级的授权；若指定<level>，则只显示指定等级的授权
```

- 查看一个敏感数据表能被哪些用户访问

```
 SHOW LABEL [<level>] GRANTS ON TABLE <tablename>;
   --显示指定table上的Label授权
```

- 查看一个用户对一个数据表的所有列级别的**Label**权限

```
 SHOW LABEL [<level>] GRANTS ON TABLE <tablename> FOR USER <username>;
   --显示指定用户对指定table上列级别的Label授权
```

- 查看一个表中所有列的敏感等级

```
DESCRIBE <tablename>;
```

- 控制**package**安装者对**package**中敏感资源的许可访问级别

```
 ALLOW PROJECT <prjName> TO INSTALL PACKAGE <pkgName> [USING LABEL <number>];
   --由package创建者授权，授予package安装者对package中敏感资源的许可访问级别
```

说明：

- 省略[USING LABEL <number>]时，默认为0级，即只可以访问非敏感数据。
- 跨项目空间访问敏感数据时，package安装者的项目空间中的所有用户都将使用此命令中许可的访问级别

**LabelSecurity**应用场景示例

- 限制项目空间中所有非**Admin**用户对一张表的某些敏感的列的读访问

  场景说明：user_profile是某项目空间中的一张含有敏感数据的表，它包含有100列，其中有5列包含敏感数据：id_card, credit_card, mobile, user_addr, birthday. 当前的DAC机制中已经授权了所有用户对该表的Select操作。ProjectOwner希望除了Admin之外，所有用户都不允许访问那5列敏感数据。

  ProjectOwner操作步骤如下：

```
set LabelSecurity=true;
  --开启LabelSecurity机制
set label 2 to table user_profile(mobile, user_addr, birthday);
  --将指定列的敏感等级设置为2
set label 3 to table user_profile(id_card, credit_card);
```

```
--将指定列的敏感等级设置为3
```

说明：

以上操作执行之后，所有非Admin用户都将无法访问那5列数据。如果有人因业务需要确实要访问这些敏感数据，那么需要ProjectOwner或Admin的授权。

Alice是项目空间中的一员，由于业务需要，她要申请访问user_profile的mobile列的数据，需要访问1周时间。项目空间管理员操作步骤如下：

```
GRANT LABEL 2 ON TABLE user_profile TO USER ALIYUN$alice@aliyun.com
WITH EXP 7;
```

说明：

敏感等级为2的数据一共有三列：mobile, user_addr, birthday。那么，当上述授权成功后，Alice将能访问这三列数据。这里存在轻微的过渡授权。如果管理员合理设置数据列的敏感度，那么这种过渡授权是可以避免的。

- **限制项目空间中已获得敏感数据访问许可的用户在项目空间内对敏感数据的复制与肆意传播**

  场景说明：在上一个场景中，Alice由于业务需要而获得了等级为2的敏感数据的访问权限。但管理员仍然担心Alice可能会将user_profile表中的敏感等级为2的那些数据复制到她自己新建的另一张表user_profile_copy中，从而进一步将user_profile_copy表自主授权给Bob访问。如何限制Alice的这种行为？

  考虑到安全易用性和管理成本，LabelSecurity的默认安全策略允许WriteDown，即允许用户向敏感等级不高于用户等级的数据列写入数据，因此MaxCompute还无法从根本上解决此问题。但这里可以限制用户的自主授权行为：允许对象创建者只能访问自己创建的数据，而不允许自主授权该对象给其他用户。操作如下：

```
SET ObjectCreatorHasAccessPermission=true;
  --允许对象创建者操作对象
SET ObjectCreatorHasGrantPermission=false;
  --不允许对象创建者授权对象给其他用户
```

# 6 Lightning

## 6.1 Lightning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，让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。

您可使用主流BI工具（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，只需为运行的查询付费。

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

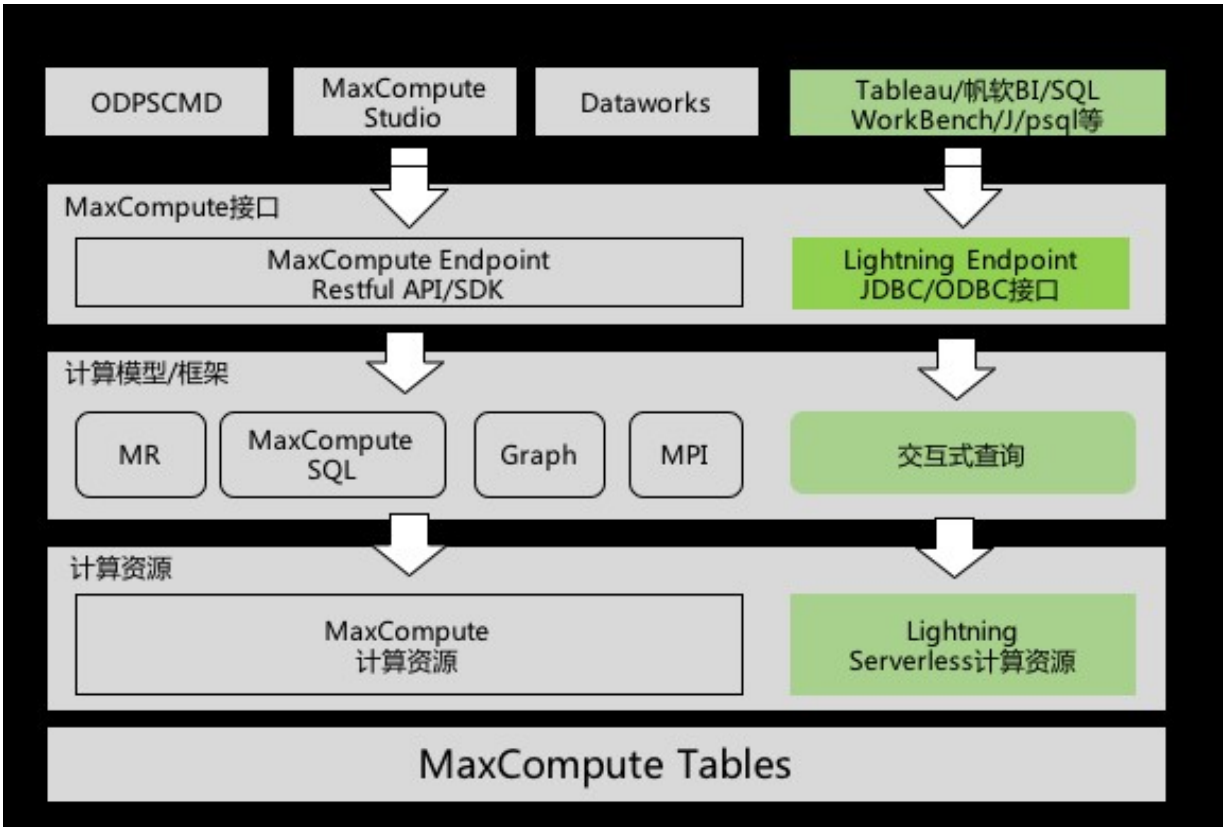
  MaxCompute Lightning是在MaxCompute已有的计算资源之外提供的无服务器的计算服务，您不需要设置、管理或运维MaxCompute Lightning资源，通过MaxCompute Lightning连接后即可开展查询。

  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，使用者可以直接对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（比如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。使用者往往是掌握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。

## 6.2 开通Lightning服务

MaxCompute Lightning是MaxCompute提供的交互式查询服务，您需要先开通并创建MaxCompute项目方可使用MaxCompute Lightning。

MaxCompute Lightning服务目前处于公测阶段，未对全网用户开放。如需使用，您可以通过我们在阿里云官网上提供的公测试用申请页面申请公测期间的服务开通。

商业化后，默认为MaxCompute项目开通MaxCompute Lightning服务。

## 6.3 服务定价

**说明：**

- MaxCompute Lightning服务目前处于公测阶段，完全免费，公测结束后将正式商业化收费。
- 正式商业化后，使用Lightning服务时按每个查询所扫描的数据量付费（单价待商业化时发布）。

使用MaxCompute Lightning，您只需为您运行的查询付费，根据执行查询过程中，实际扫描的MaxCompute项目表的数据量计费。当不运行查询时，MaxCompute Lightning不收取任何费用。

由于MaxCompute Lightning的使用依赖于您开通创建的MaxCompute项目，因此您还需要关注MaxCompute在数据存储、计算（按量后付费/按CU预付费）、外网下载等方面产生的费用。

计费详情请参见*MaxCompute*计量计费说明。

# 6.4 快速开始

## 6.4.1 使用说明

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。

## 6.4.2 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*进行开通并创建一个MaxCompute项目。

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*MaxCompute*快速入门。

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。子账号如果没有查看权限，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。

## 6.4.3 准备连接的客户端工具

MaxCompute Lightning兼容PostgreSQL接口，支持PostgreSQL数据库连接的客户端工具都可以用于连接MaxCompute Lightning。

本教程中选择了大家熟悉的BI工具*Tableau Desktop* 进行示例，相关工具请到Tableau官网进行下载。

其他常见客户端工具，如SQL Workbench/J、PSQL、帆软BI、MicroStrategy BI等都可以像连接PostgreSQL数据库一样配置服务连接。

## 6.4.4 连接服务并开展分析

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 EndPoint，如华东2区域的Endpoint为：`lightning.cn-shanghai.maxcompute.aliyun.com` |
| 端口 | 443 |
| 数据库 | MaxCompute项目名 |
| 身份验证 | 用户名和密码 |
| 用户名/密码 | 访问用户的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/模型。

   配置好联系信息并登录后，Tableau会加载所连接的MaxCompute项目下的表，使用者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*Tableau Desktop*。

## 6.5 访问域名

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。

MaxCompute Lightning在公共云的不同Region及网络环境下的服务连接对照表如下。

**表 6-1: 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 | 公测中 | lightning.cn-hangzhou.maxcompute.aliyun.com |
| 华东2 | 公测中 | lightning.cn-shanghai.maxcompute.aliyun.com |
| 华北2 | 公测中 | lightning.cn-beijing.maxcompute.aliyun.com |

| Region | 开服状态 | 外网Endpoint |
| --- | --- | --- |
| 亚太东南1 | 公测中 | lightning.ap-southeast-1.maxcompute.aliyun.com |
| 其他区域 | 暂未开服 | - |

表 6-2: 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 | 公测中 | lightning.cn-hangzhou.maxcompute.aliyun-inc.com |
| 华东2 | 公测中 | lightning.cn-shanghai.maxcompute.aliyun-inc.com |
| 华北2 | 公测中 | lightning.cn-beijing.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 | - |

表 6-3: 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 | 公测中 | lightning.cn-hangzhou.maxcompute.aliyun-inc.com |
| 华东2 | 公测中 | lightning.cn-shanghai.maxcompute.aliyun-inc.com |
| 华北2 | 公测中 | lightning.cn-beijing.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 | - |

## 6.6 通过JDBC连接服务

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。

如果您的MaxCompute项目还没有数据或需要对现有数据进行加工处理，请参见*MaxCompute*文档，通过*MaxCompute*客户端或*DataWorks*连接MaxCompute项目进行数据对象创建和加工处理。

### 6.6.1 JDBC驱动程序

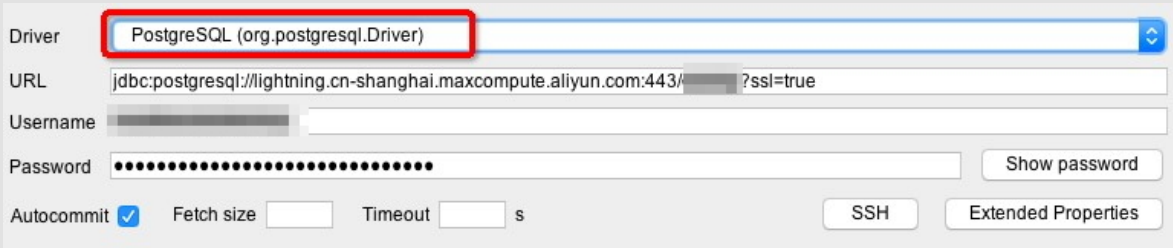
MaxCompute提供完全兼容PostgreSQL消息协议的Java数据库连接（JDBC）接口，使用者可以通过JDBC将SQL客户端工具连接到MaxCompute Lightning服务。

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。

1. 使用PostgeSQL官方提供的*JDBC*驱动程序。

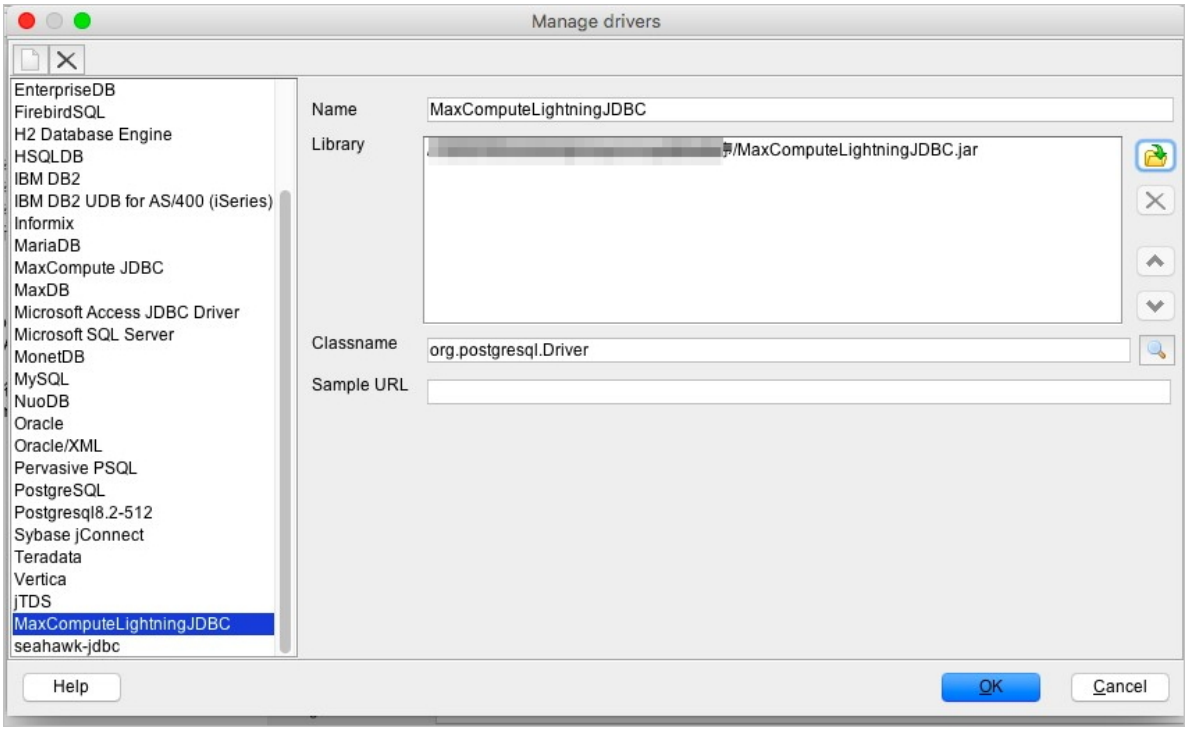
   **说明：**

   很多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。

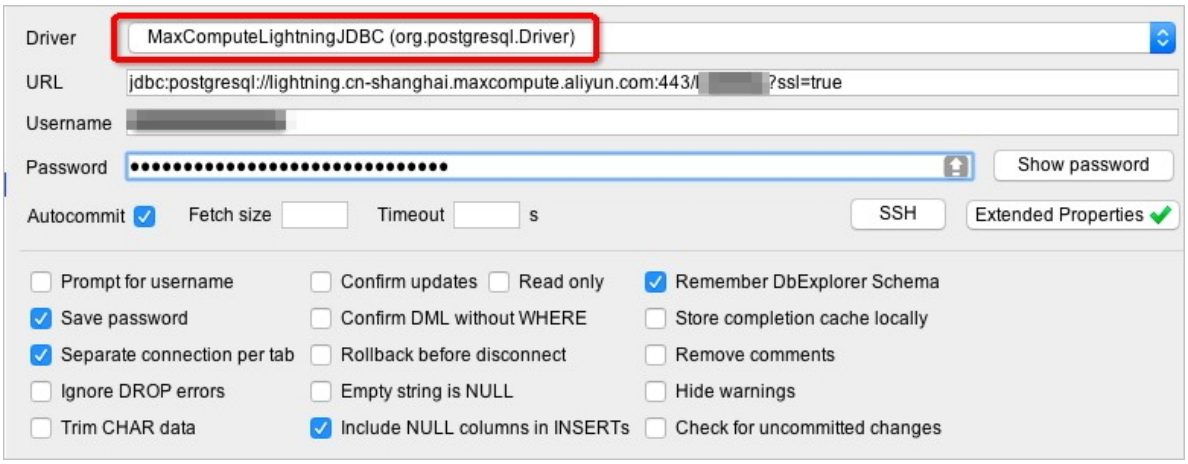
   
   

2. 使用阿里云MaxCompute Lightning优化过的*JDBC驱动*，以获取更好的性能。

   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。

   在创建连接时，从Driver列表中选择刚才添加的MaxCompute Lightning JDBC驱动。




## 6.6.2 配置JDBC连接

您需要获得您的MaxCompute项目JDBC URL，才能将SQL客户端工具连接到该项目。

JDBC URL命名方式如下：

```
jdbc:postgresql://endpoint:port/database
```

连接参数说明如下：

| 参数 | 取值 | 说明 |
| --- | --- | --- |
| endpoint | 所在区域不同网络环境下的Lightning访问域名 | 详情请参见访问域名，例如通过外网访问上海Region的服务使用`lightning.cn-shanghai.maxcompute.aliyun.com` |
| port | 443 | - |
| database | 填写MaxCompute的项目名称 | - |
| user | 访问用户的Access Key ID | - |
| password | 访问用户的Access Key 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。 |
| prepareThreshold | 0 | 可选。需要使用JDBC PrepareStatement功能时，建议设置`prepareThreshold=0`。 |

例如`jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject`

同时，设置user、password、SSL连接参数后进行连接。

您也可以将参数添加到JDBC URL来连接到MaxCompute项目，如下所示：

```
jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/
myproject?ssl=true& prepareThreshold=0&user=xxx&password=yyy
```

说明如下：

- lightning.cn-shanghai.maxcompute.aliyun.com是华东2区域的Endpoint。
- myproject是需要访问的MaxCompute项目名称。
- ssl=true是指定通过SSL方式连接。
- xxx是访问用户的Access Key ID。
- yyy是访问用户的Access Key Secret。

## 6.6.3 常见工具的连接

本文将为您介绍几款常见客户端工具的接入说明，除此之外，原则上支持PostgresSQL的工具都可以通过Lightning来对接访问MaxCompute。

**阿里云Quick BI**

1. 登录Quick BI控制台，单击左侧导航栏中的**数据源**。
2. 单击数据源管理页面右上角的**新建数据源**。
3. 选择云数据库或自建数据源中的PostgeSQL数据库类型添加数据源。
4. 填写对话框中的MaxCompute Lightning的连接信息并测试连接连通状态。

| 参数 | 说明 |
| --- | --- |
| 数据库地址 | MaxCompute Lightning对应区域的Endpoint，可使用公网访问的Endpoint，也可以使用经典网络及VPC网络访问的Endpoint。 |
| 数据库 | 需要访问的MaxCompute项目的名称加`?ssl=true`，如上图中的`lightning?ssl=true`。 |
| Schema | MaxCompute项目名称。 |
| 用户名/密码 | 用户的Access Key ID/Access Key Secret。 |

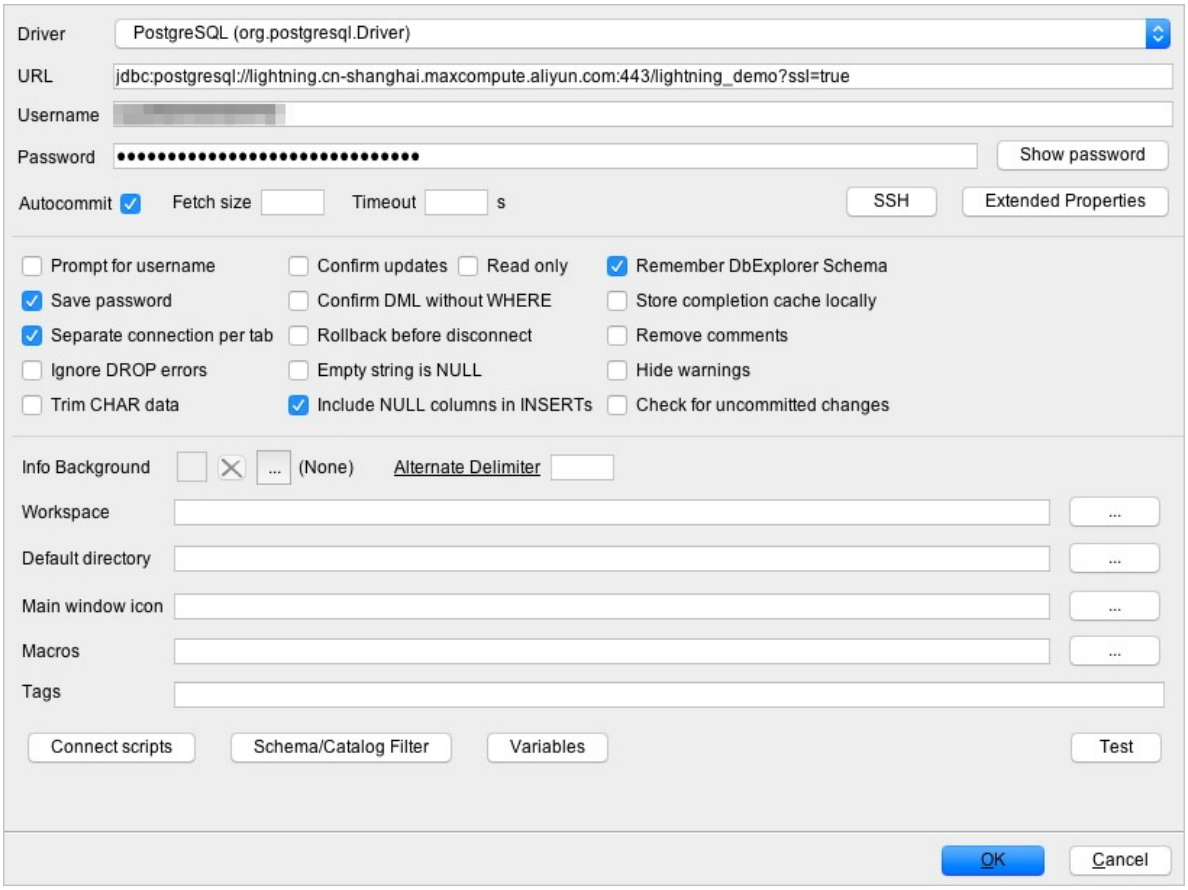
**SQL Workbench/J**

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。

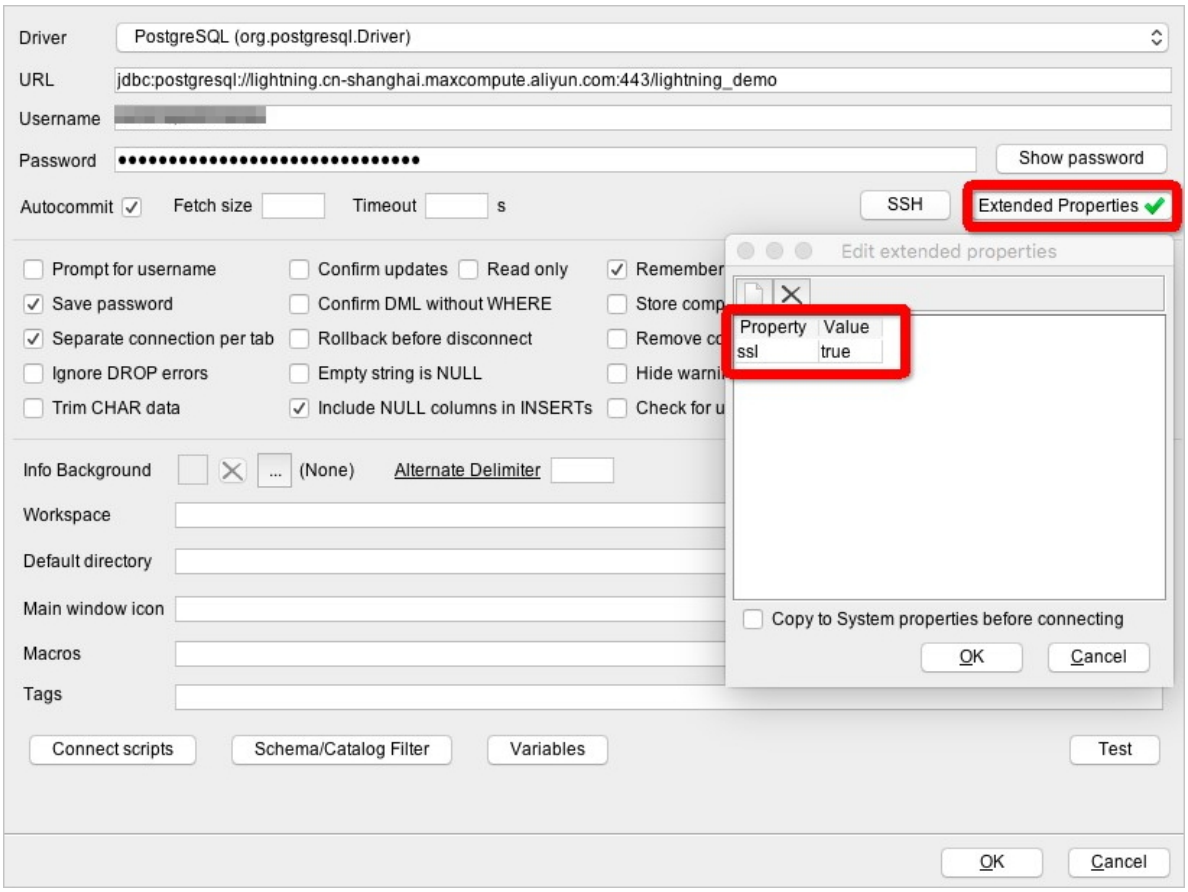
1. 下载并安装SQL Workbench/J 。

2. 启动SQL Workbench/J，创建数据库连接。

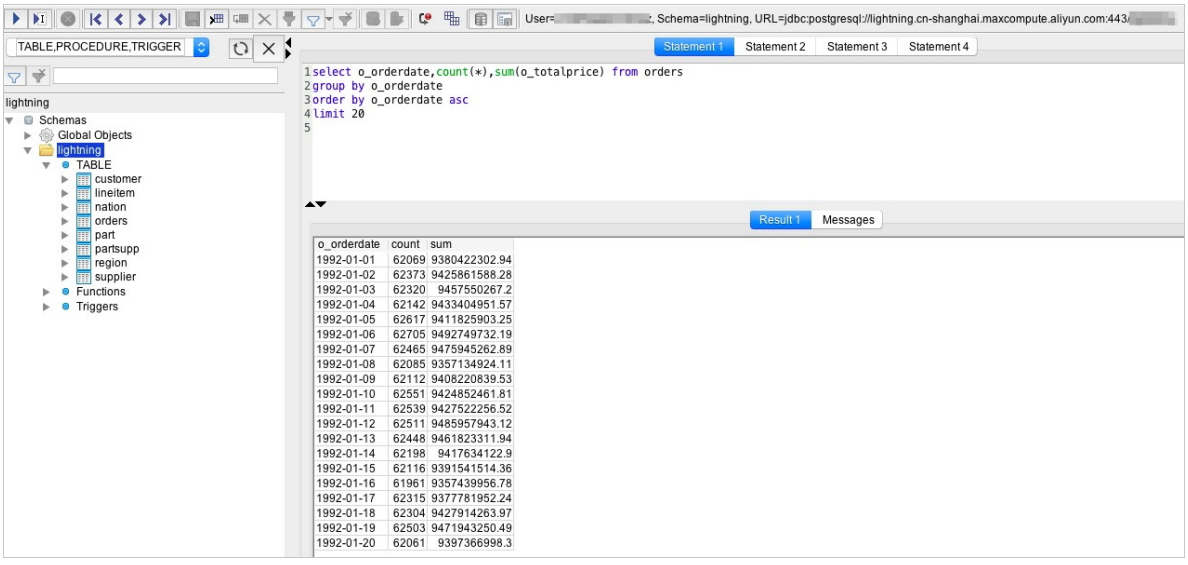
选择PostgreSQL驱动，连接MaxCompute项目所对应的Lightning URL地址，同时输入访问用户的用户名和密码，即Access Key ID和Access Key Secret。




您也可通过扩展属性（Extended Properities）设置ssl取值为true。




3. 连接后，在Workbench工作区查看MaxCompute项目的表数据、查询分析。




**psql**工具连接

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。

通过psql在命令行下可以连接MaxCompute Lightning，语法与连接PostgreSQL数据库一致。

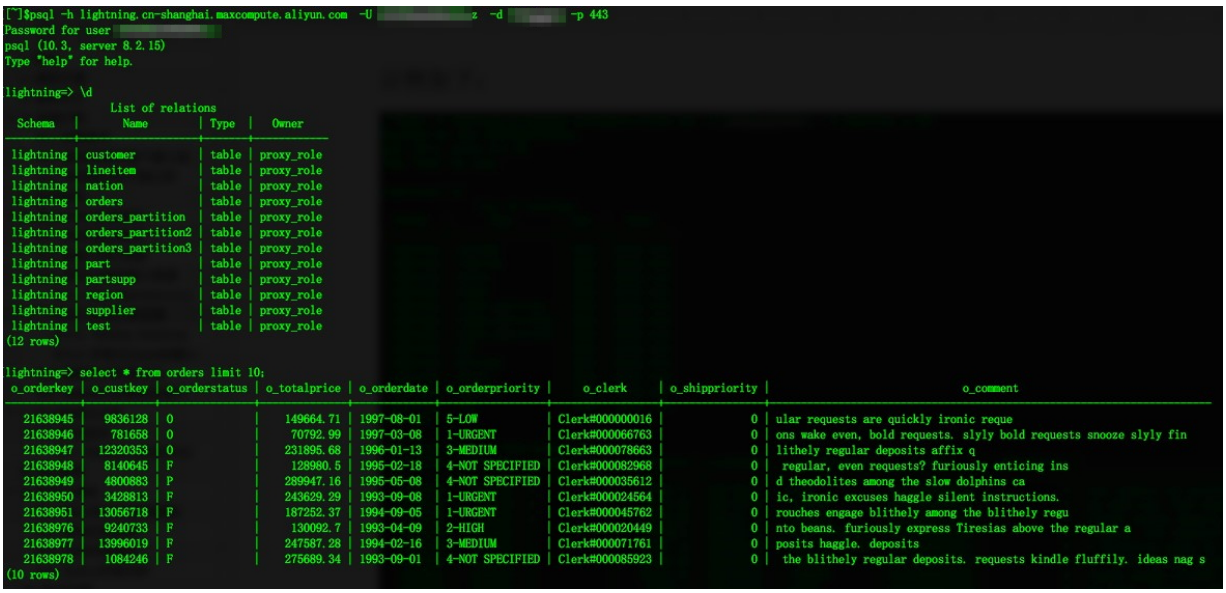
```
psql -h <endpoint> -U <userid> -d <databasename> -p <port>
```

参数说明：

- <endpoint>：MaxCompute Lightning的Endpoint，详情请参见访问域名。
- <userid>：访问用户Access Key ID。
- <databasename>：Maxcompute项目名。
- <port>：443

执行后，在psql密码提示符处，输入<userid>用户的密码，即访问用户的Access Key Secret。

示例如下：




说明：

psql默认优先通过ssl方式连接。

### Tableau Desktop

使用BI工具，选择PostgreSQL数据源，配置连接。

配置连接时，需勾选需要**SSL**。

登录后，通过Tableau创建工作表进行可视化分析。

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。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 Documents\My Tableau Repository\Datasources*目录下。如果是Tableau Server，Windows下请保存在*C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Datasources*，Linux下请保存在*/var/opt/tableau/tableau_server/data/tabsvc/vizqlserver/Datasources/*。
3. 重新打开Tableau，使用PostgreSQL数据源连接MaxCompute Lightning服务。关于tdc文件定制数据源的更多内容，请参见*Tableau*官方帮助文档。

**帆软Report**

1. 打开帆软Report，选择**服务器 > 定义数据库连接**。
2. 添加JDBC连接。

   参数说明如下：

| 参数 | 说明 |
| --- | --- |
| 数据库 | Postgre |
| 驱动器 | 帆软Report自带的org.postgresql.Driver |
| URL | `jdbc:postgresql://<MaxCompute Lightning Endpoint>:443/<Project_Name>?ssl=true&prepareThreshold=0`<br>例如：`jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/lightning_demo?ssl=true&prepareThreshold=0` |
| 用户名/密码 | 访问用户的Access Key ID和Access Key Secret |

# 6.7 SQL参考

**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**

MaxCompute Lightning查询引擎基于PostgreSQL 8.2提供内建函数，请参见 *PostgreSQL官方文档*。

在PostgreSQL官方函数基础上，MaxCompute Lightning补充了以下内建函数。

- **MAX_PT**

  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，按字母排序，且该分区下有对应的数据文件。

  参数说明

  table_full_name：String类型，用于指定表名（必须携带project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，且都包含数据文件：

```
pt ='20120901'
pt ='20120902'
```

则以下语句中分区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');
```

## 6.8 查看作业

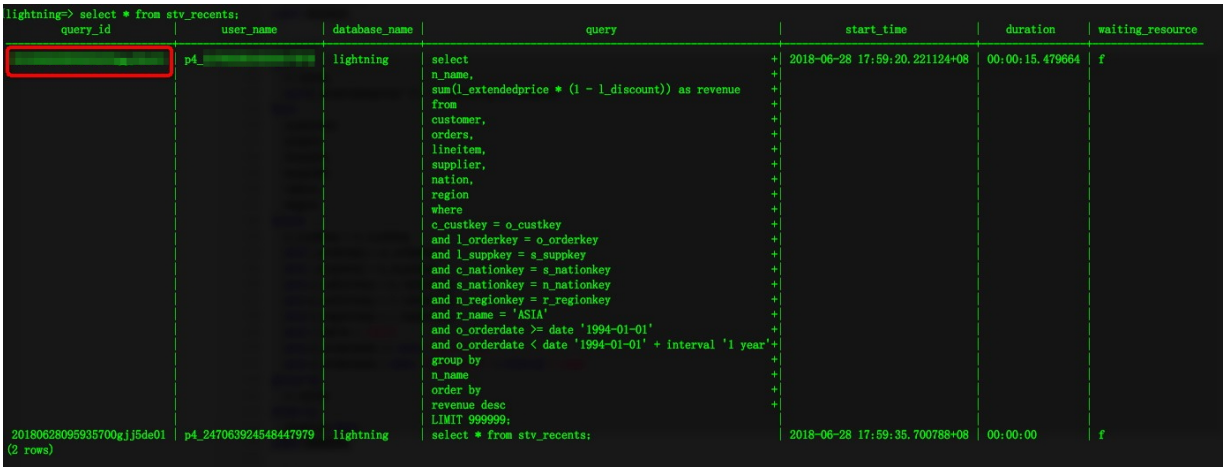
查看运行中的查询

Lightning为用户提供了一个系统视图表stv_recents，通过查询该表能够获取当前用户正在执行中的所有查询作业，查看这些作业的ID、用户、SQL语句、发起开始时间、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：

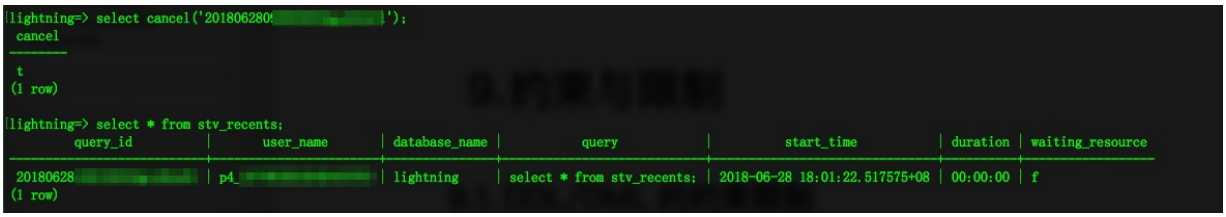



取消正在运行的查询

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：




# 6.9 约束与限制

**DDL/DML的约束限制**

MaxCompute Lightning目前不支持update、create、delete、insert操作，仅支持对MaxCompute表进行select，后续版本将陆续开放相关能力。

查询约束限制

- 查询分区表时，扫描分区数的最大值为1024。
- 目前不支持创建和使用View。
- 目前不支持的数据类型：map、array、tinyint和timestamp（陆续支持中）。
- 每个查询中对单张表最大的数据扫描量为1TB。
- 提交的查询语句的长度不超过100KB。
- 查询超时时间为1小时。

**UDF约束限制**

- 当前不支持在Maxcompute Lightning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）。

**Query并发约束**

单个MaxCompute项目的MaxCompute Lightning查询并发数限制为20。

# 6.10 Lightning常见问题

- Q：还没有建表的情况下，可以用MaxCompute Lightning查什么数据？

  A：您需要先通过DataWorks或odpscmd客户端工具，在MaxCompute项目中创建数据表，加载数据。然后通过MaxCompute Lightning连接到该项目，此时便可查看到项目内的表，并对这些表进行查询。
- Q：MaxCompute Lightning是否限制查询的数据量？查询多大规模的数据性能较好？

A：目前每次查询对单表的扫描数据量限制为1TB，数据量越小查询性能越好。

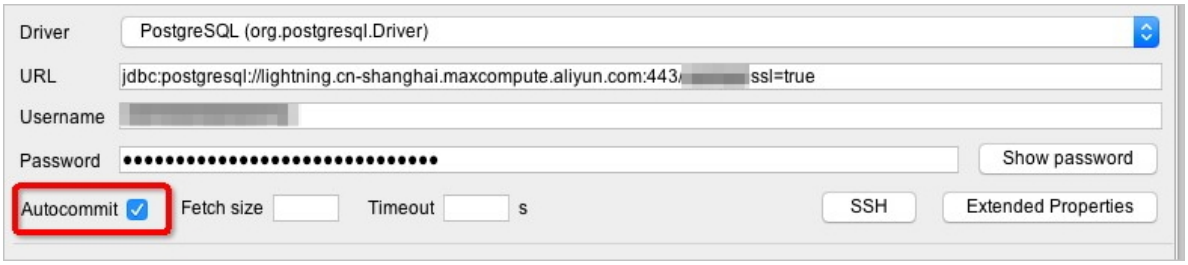
> 说明：
> 建议不要扫描超过100GB的表数据。超过100GB的表数据虽然仍可查询，但查询性能会随数据规模增长逐渐下降。如果需要扫描量超过100GB的查询，建议您根据性能表现考虑使用MaxCompute SQL。

- Q：使用BI工具，通过拖拽方式选择一张分区表进行分析时，提示报错：`ERROR: AXF Exception: specified partitions count in odps table: <project_name.table_name> is: xxx, exceeds the limitation of xxx, please add stricter partition filter`。

  A：MaxCompute Lightning为保障查询性能，对分区表的分区数量进行了限制，一次查询所扫描的单表最大分区数量不能超过1024。由于部分BI工具使用拖拽方式选择表直接进行分析，不能BI前端指定分区条件，导致请求扫描的分区数超限制、触发了Lightning限制而报错提示。建议先对查询的数据表进行加工处理，处理为非分区表或分区小于1024的表再进行分析。
- Q：连接时提示创建数据连接失败: `ERROR: SSL required`。

  A：MaxCompute Lightning要求SSL连接服务，需要客户端指定以SSL方式连接。如果使用客户端工具，可以选择SSL连接选项。如果没有相关选项，可以在JDBC URL连接串中增加SSL参数，需要替换为您项目所在region的endpoint、连接的项目名称，例如`jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject?ssl=true`。
- Q：使用Workbench/J客户端查询时提示`Error:current transaction is aborted, commands ignored until end of transaction block`。

  A：使用的客户端请勾选**Autocommit**选项。



# 7 PyODPS

## 7.1 安装指南

如果能访问外网，推荐使用pip安装，pip安装可以参考 地址 ， 推荐使用 阿里云镜像 加快下载速度。

接着确保 setuptools 和 requests 的版本，对于非 Windows 用户可以安装 Cython 加速 Tunnel 上传下载：

```
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
```

安装有 合适版本 Visual C++ 和 Cython 的 Windows 用户也可使用 Tunnel 加速功能。

接着就可以安装PyODPS：

```
pip install pyodps
```

检查安装完成：

```
python -c "from odps import ODPS"
```

如果使用的python不是系统默认的python版本，安装完pip则可以：

```
/home/tops/bin/python2.7 -m pip install setuptools>=3.0
```

其余类似。

## 7.2 工具平台使用指南

### 7.2.1 工具平台使用指南概述

PyODPS 可在 DataWorks 等数据开发平台中作为节点调用。这些平台提供了 PyODPS 运行环境，不需要手动创建 ODPS 入口对象，免除了手动配置的麻烦，而且还提供了调度执行的能力。对于想从平台迁移到自行部署 PyODPS 环境的用户，下面也提供了迁移注意事项。

- *DataWorks* 用户使用指南
- 从平台到自行部署

## 7.2.2 从平台到自行部署

如果你需要在本地调试 PyODPS，或者平台中没有提供你所需的包，或者平台的资源限制不能满足你的要求，此时你可能需要 从平台迁移到自己部署的 PyODPS 环境。

安装 PyODPS 的步骤可以参考 安装指南。你需要手动创建先前平台为你创建的 ODPS 入口对象。可以在先前的平台 使用下列语句生成创建 ODPS 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))
```

其中，<access-key> 和 <endpoint> 需要手动替换成可用的值。access-key 可在 DataWorks 中点击右上角图标 -> 用户信息， 再点击“点击查看”获得。Endpoint 可在 *MaxCompute开通Region和服务连接对照表* 获得， 或者联系项目管理员。

修改完成后，将上述代码放置到所有代码的最顶端即可。

## 7.2.3 DataWorks 用户使用指南

新建工作流节点

在工作流节点中会包含PYODPS节点。新建即可。

### ODPS入口

DataWorks 的 PyODPS 节点中，将会包含一个全局的变量 `odps` 或者 `o` ，即 ODPS 入口。用户不需要手动定义 ODPS 入口。

```
print(o.exist_table('pyodps_iris'))
```

### 执行SQL

可以参考执行*SQL*文档 。

说明：

Dataworks 上默认没有打开 instance tunnel，即 instance.open_reader 默认走 result 接口（最多一万条）。 打开 instance tunnel，通过 reader.count 能取到记录数，如果要迭代获取全部数据，则需要关闭 limit 限制。

要想全局打开，则：

```
options.tunnel.use_instance_tunnel = True
options.tunnel.limit_instance_tunnel = False  # 关闭 limit 读取全部数据

with instance.open_reader() as reader:
    # 能通过 instance tunnel 读取全部数据
```

或者通过在 open_reader 上添加 `tunnel=True`，来仅对这次 open_reader 打开 instance tunnel；添加 `limit=False`，来关闭 limit 限制从而能下载全部数据。

```
with instance.open_reader(tunnel=True, limit=False) as reader:
    # 这次 open_reader 会走 instance tunnel 接口，且能读取全部数据
```

### DataFrame

- 执行

  在 DataWorks 的环境里，*DataFrame*概述 的执行需要显式调用 立即执行的方法#如*execute#head*等# 。

  ```
  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
```

如果用户想在print的时候调用立即执行，需要打开 `options.interactive` 。

```
from odps import options
from odps.df import DataFrame

options.interactive = True  # 在开始打开开关

iris = DataFrame(o.get_table('pyodps_iris'))
print(iris.sepal_width.sum())  # 这里print的时候会立即执行
```

- 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，为了避免侵入代码，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}` ，则可以通过下面的方式在代码中获取此参数：

```
print('ds=' + args['ds'])
```

```
ds=20161116
```

特别地，如果要获取名为 `ds=${yyyymmdd}`的分区，则可以使用

```
o.get_table('table_name').get_partition('ds=' + args['ds'])
```

受限功能

由于缺少 matplotlib 等包，所以如下功能可能会受限。

- DataFrame的plot函数

DataFrame 自定义函数需要提交到 MaxCompute 执行。由于 Python 沙箱的原因，第三方库只支持所有的纯 Python 库以及 Numpy， 因此不能直接使用 Pandas，可参考:ref:第三方库支持 <third_party_library> 上传和使用所需的库。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。其他带有二进制代码的三方包不被支持。

由于兼容性的原因，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。

使用限制

在 DataWorks 上使用 PyODPS，为了防止对 DataWorks 的 gateway 造成压力，对内存和 CPU 都有限制。这个限制由 DataWorks 统一管理。

如果看到 **Got killed**，即内存使用超限，进程被 kill。因此，尽量避免本地的数据操作。

通过 PyODPS 起的 SQL 和 DataFrame 任务 ( 除 to_pandas) 不受此限制。

# 7.3 基本操作

## 7.3.1 基本操作概述

PyODPS 提供直接针对 ODPS 对象的基本操作接口，可通过符合 Python 习惯的编程方式操作 ODPS。

我们会对这几部分来分别展开说明。

- 项目空间
- 表
- *SQL*
- 任务实例
- 资源
- 函数
- *XFlow*和模型

## 7.3.2 项目空间

项目空间 是ODPS的基本组织单元， 有点类似于Database的概念。

我们通过 ODPS 入口对象的 `get_project` 来取到某个项目空间。

```
project = o.get_project('my_project')  # 取到某个项目
project = o.get_project()              # 取到默认项目
```

如果不提供参数，则取到默认项目空间。

`exist_project` 方法能检验某个项目空间是否存在。

## 7.3.3 表

表是ODPS的数据存储单元。

### 基本操作

我们可以用 ODPS 入口对象的 `list_tables`来列出项目空间下的所有表。

```
for table in o.list_tables():
    # 处理每个表
```

通过调用 `exist_table` 来判断表是否存在。

通过调用 `get_table` 来获取表。

```
>>> t = o.get_table('dual')
>>> 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
'Dual 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'
```

通过提供 `project`参数，来跨project获取表。

```
>>> t = o.get_table('dual', project='other_project')
```

### 创建表的Schema

有两种方法来初始化。第一种方式通过表的列、以及可选的分区来初始化。

```
>>> 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]
```

第二种方法是使用 `Schema.from_lists`，这种方法更容易调用，但显然无法直接设置列和分区的注释了。

```
>>> 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

可以使用表 schema 来创建表，方法如下：

```
>>> table = o.create_table('my_new_table', schema)
>>> table = o.create_table('my_new_table', schema, if_not_exists=True)
  # 只有不存在表时才创建
>>> table = o.create_table('my_new_table', schema, lifecycle=7)  # 设置
生命周期
```

更简单的方式是采用“字段名 字段类型”字符串来创建表，方法如下：

```
>>> 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)
```

在未经设置的情况下，创建表时，只允许使用 bigint、double、decimal、string、datetime、boolean、map 和 array 类型。 如果你使用的是位于公共云上的服务，或者支持 tinyint、struct 等新类型，可以设置

`options.sql.use_odps2_extension = True`

打开这些类型的支持，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

有时候，一个表可能被别的程序做了更新，比如schema有了变化。此时可以调用 `reload` 方法来更新。

```
>>> table.reload()
```

### 行记录Record

Record表示表的一行记录，我们在 Table 对象上调用 new_record 就可以创建一个新的 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()  #  也可以不传入值
>>> r2[0] = 'val0' # 可以通过偏移设置值
>>> r2['field1'] = 'val1'  # 也可以通过字段名设置值
>>> 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
```

### 获取表数据

有若干种方法能够获取表数据。首先，如果只是查看每个表的开始的小于1万条数据，则可以使用head方法。

```
>>> t = o.get_table('dual')
>>> for record in t.head(3):
```

```
>>>     # 处理每个Record对象
```

其次，在table上可以执行 `open_reader` 操作来打一个reader来读取数据。

使用 with 表达式的写法：

```
>>> 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，这里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：

```
>>> 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，这里可以改造成并行操作
>>>     # 处理一条记录
```

更简单的调用方法是使用 ODPS 对象的 `read_table`方法，例如

```
>>> for record in o.read_table('test_table', partition='pt=test'):
>>>     # 处理一条记录
```

**向表写数据**

类似于 `open_reader`，table对象同样能执行 `open_writer` 来打开writer，并写数据。

使用 with 表达式的写法：

```
>>> with t.open_writer(partition='pt=test') 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)  # 这里records可以是可迭代对象
>>>
>>>     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)
>>>
>>> with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer:
 # 这里同是打开两个block
>>>     writer.write(0, gen_records(block=0))
>>>     writer.write(1, gen_records(block=1))  # 这里两个写操作可以多线程
并行，各个block间是独立的
```

```
>>>
```

不使用 with 表达式的写法：

```
>>> writer = t.open_writer(partition='pt=test', blocks=[0, 1])
>>> writer.write(0, gen_records(block=0))
>>> writer.write(1, gen_records(block=1))
>>> writer.close()  # 不要忘记关闭 writer，否则数据可能写入不完全
```

如果分区不存在，可以使用`create_partition`参数指定创建分区，如

```
>>> 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)  # 这里records可以是可迭代对象
```

说明：

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。因此，我们建议在使用 write_table 方法时，一次性写入多组数据，或者传入一个 generator 对象。

删除表

```
>>> o.delete_table('my_table_name', if_exists=True)  #  只有表存在时删除
>>> t.drop()  # Table对象存在的时候可以直接执行drop函数
```

创建**DataFrame**

PyODPS提供了DataFrame框架*DataFrame*框架 ，支持更方便地方式来查询和操作ODPS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

- 基本操作

  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)
2015-11-18 22:22:27
>>> partition.size
0
```

- 创建分区

```
>>> t.create_partition('pt=test', if_not_exists=True)  # 不存在的时候才创建
```

- 删除分区

```
>>> t.delete_partition('pt=test', if_exists=True)  # 存在的时候才删除
>>> partition.drop()  # Partition对象存在的时候直接drop
```

**数据上传下载通道**

ODPS Tunnel是ODPS的数据通道，用户可以通过Tunnel向ODPS中上传或者下载数据。

**说明：**

不推荐直接使用tunnel接口（难用且复杂），推荐直接使用表的写和读接口。 如果安装了

**Cython**，在安装pyodps时会编译C代码，加速Tunnel的上传和下载。

- 上传

```
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

```
from odps.tunnel import TableTunnel

tunnel = TableTunnel(odps)
download_session = tunnel.create_download_session('my_table',
partition_spec='pt=test')

with download_session.open_record_reader(0, download_session.count)
as reader:
    for record in reader:
        # 处理每条记录
```

## 7.3.4 SQL

PyODPS支持ODPS SQL的查询，并可以读取执行的结果。

### SQL

`execute_sql` 或者 `run_sql` 方法的返回值是 运行实例 。

**说明：**

并非所有在 ODPS Console 中可以执行的命令都是 ODPS 可以接受的 SQL 语句。 在调用非 DDL / DML 语句时，请使用其他方法，例如 GRANT / REVOKE 等语句请使用`run_security_query`方法，PAI 命令请使用 `run_xflow` 或 `execute_xflow`方法。

### 执行SQL

```
>>> 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
```

### 设置运行参数

有时，我们在运行时，需要设置运行时参数，我们可以通过设置 `hints`参数，参数类型是dict。

```
>>> o.execute_sql('select * from pyodps_iris', hints={'odps.sql.mapper
.split.size': 16})
```

我们可以对于全局配置设置sql.settings后，每次运行时则都会添加相关的运行时参数。

```
>>> 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

运行 SQL 的 instance 能够直接执行 `open_reader`的操作，一种情况是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
```

另一种情况是 SQL 可能执行的比如 `desc`，这时通过 `reader.raw`属性取到原始的SQL执行结果。

```
>>> with o.execute_sql('desc dual').open_reader() as reader:
>>>     print(reader.raw)
```

如果 `options.tunnel.use_instance_tunnel == True`，在调用 open_reader 时，PyODPS 会默认调用 Instance Tunnel， 否则会调用旧的 Result 接口。如果你使用了版本较低的 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，可根据警告信息判断导致降级的原因。如果 Instance Tunnel 的结果不合预期，请将该选项设为 False。在调用 open_reader 时，也可以使用 `tunnel`参数来指定使用何种结果接口，例如

```
>>> # 使用 Instance Tunnel
>>> with o.execute_sql('select * from dual').open_reader(tunnel=True)
as reader:
>>>     for record in reader:
>>>         # 处理每一个record
>>> # 使用 Results 接口
>>> with o.execute_sql('select * from dual').open_reader(tunnel=False
) as reader:
>>>     for record in reader:
```

```
>>>         # 处理每一个record
```

PyODPS 默认不限制能够从 Instance 读取的数据规模。对于受保护的 Project，通过 Tunnel 下载数据受限。此时， 如果 options.tunnel.limit_instance_tunnel 未设置，会自动打开数据量限制。此时，可下载的数据条数受到 Project 配置限制， 通常该限制为 10000 条。如果你想要手动限制下载数据的规模，可以为 open_reader 方法增加 limit 选项， 或者设置 options.tunnel.limit_instance_tunnel = True 。

如果你所使用的 MaxCompute 只能支持旧 Result 接口，同时你需要读取所有数据，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）。

### 设置alias

有时在运行时，比如某个UDF引用的资源是动态变化的，我们可以alias旧的资源名到新的资源，这样免去了重新删除并重新创建UDF的麻烦。

```
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])
```

```
2
```

```
res2 = o.create_resource('test_alias_res2', 'file', file_obj='2')
# 把内容为1的资源alias成内容为2的资源，我们不需要修改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])
```

```
3
```

#### 在交互式环境执行 SQL

在 ipython 和 jupyter 里支持使用 *SQL* 插件的方式运行 *SQL* ，且支持 参数化查询。

#### 设置 biz_id

在少数情形下，可能在提交 SQL 时，需要同时提交 biz_id，否则执行会报错。此时，你可以设置全局 options 里的 biz_id。

```
from odps import options

options.biz_id = 'my_biz_id'
o.execute_sql('select * from pyodps_iris')
```

## 7.3.5 任务实例

Task如SQLTask是ODPS的基本计算单元。

#### 基本操作

当一个Task在执行时会被实例化， 以 *ODPS实例* 的形式存在。可以调用 `list_instances`来获取项目空间下的所有instance，`exist_instance`能判断是否存在某instance， `get_instance`能获取实例。

```
>>> for instance in o.list_instances():
>>>     print(instance.id)
>>> o.exist_instance('my_instance_id')
```

停止一个instance可以在odps入口使用 `stop_instance`，或者对instance对象调用 `stop`方法。

#### 获取 LogView 地址

对于 SQL 等任务，通过调用 `get_logview_address`方法即可。

```
>>> # 从已有的 instance 对象
>>> instance = o.run_sql('desc pyodps_iris')
```

```
>>> print(instance.get_logview_address())
>>> # 从 instance id
>>> instance = o.get_instance('20160426055209459g9k5pvyi2')
>>> print(instance.get_logview_address())
```

对于 XFlow 任务，需要枚举其子任务，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`，用户可以通过 `status` 属性来获取状态。 `is_terminated` 方法返回当前instance是否已经执行完成，`is_successful` 方法返回当前instance是否正确完成执行， 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'
```

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，不同的是， 如果最终任务执行失败，则会抛出相关异常。

### 子任务操作

一个Instance真正运行时，可能包含一个或者多个子任务，我们称为Task，要注意这个Task不同于ODPS的计算单元。

我们可以通过 `get_task_names` 来获取所有的Task任务，它返回一个所有子任务的名称列表。

```
>>> instance.get_task_names()
['SQLDropTableTask']
```

拿到Task的名称，我们就可以通过 `get_task_result`来获取这个Task的执行结果。

`get_task_results`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')])
```

有时候我们需要在任务实例运行时显示所有子任务的运行进程。使用 `get_task_progress`能获得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
%]
```

## 7.3.6 资源

### 资源

资源 在ODPS上常用在UDF和MapReduce中。

列出所有资源还是可以使用 `list_resources`，判断资源是否存在使用 `exist_resource`。 删除资源时，可以调用 `delete_resource`，或者直接对于Resource对象调用 `drop` 方法。

在PyODPS中，主要支持两种资源类型，一种是文件，另一种是表。

### 文件资源

文件资源包括基础的 `file` 类型、以及 `py`、`jar`、`archive`。

- 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、以及一个file-like的对象（或者是字符串对象）来创建，比如

  ```
  resource = o.create_resource('test_file_resource', 'file', file_obj=
  open('/to/path/file'))  # 使用file-like的对象
  resource = o.create_resource('test_py_resource', 'py', file_obj='
  import this')  # 使用字符串
  ```

- 读取和修改文件资源

对文件资源调用 `open`方法，或者在odps入口调用 `open_resource`都能打开一个资源， 打开后的对象会是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。我们看例子：

```
>>> with resource.open('r') as fp:  # 以读模式打开
>>>     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
>>>     fp.read()
>>>     fp.tell()  # 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，文件内容会被先清空
- `a`，追加模式，只能写入内容到文件末尾
- `r+`，读写模式，能任意读写内容
- `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，打开如说一些压缩文件等等就需要以这种模式， 因此 `rb` 就是指以二进制读模式打开文件，`r+b` 是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)
```

## 7.3.7 函数

**函数**

ODPS用户可以编写自定义函数 用在ODPS SQL中。

**基本操作**

可以调用 ODPS 入口对象的 `list_functions` 来获取项目空间下的所有函数，`exist_function`能判断是否存在函数， `get_function` 获取函数对象。

**创建函数**

可以调用 ODPS 入口对象的 `list_functions`来获取项目空间下的所有函数，`exist_function` 能判断是否存在函数， `get_function`获取函数对象。

**创建函数**

```
>>> resource = o.get_resource('my_udf.py')
>>> function = o.create_function('test_function', class_type='my_udf.
Test', resources=[resource, ])
```

**说明：**

注意，公共云由于安全原因，使用 Python UDF 需要申请。

**删除函数**

```
>>> o.delete_function('test_function')
```

```
>>> function.drop()  # Function对象存在时直接调用drop
```

**更新函数**

只需对函数调用 `update`方法即可。

```
>>> 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
```

## 7.3.8 XFlow和模型

XFlow 是 ODPS 对算法包的封装，使用 PyODPS 可以执行 XFlow。

**XFlow**

对于下面的 PAI 命令：

```
PAI -name AlgoName -project algo_public -Dparam1=param_value1 -Dparam2
=param_value2 ...
```

可以使用如下方法调用：

```
>>> # 异步调用
>>> inst = o.run_xflow('AlgoName', 'algo_public',
                       parameters={'param1': 'param_value1', 'param2':
 'param_value2', ...})
```

或者使用同步调用：

```
>>> # 同步调用
>>> inst = o.execute_xflow('AlgoName', 'algo_public',
                            parameters={'param1': 'param_value1', '
param2': 'param_value2', ...})
```

参数不应包含命令两端的引号（如果有），也不应该包含末尾的分号。

这两个方法都会返回一个 Instance 对象。由于 XFlow 的一个 Instance 包含若干个子 Instance，需要使用下面的方法来获得每个 Instance 的 LogView：

```
>>> for sub_inst_name, sub_inst in o.get_xflow_sub_instances(inst).
items():
>>>     print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address
()))
```

需要注意的是，

`get_xflow_sub_instances`

返回的是 Instance 当前的子 Instance，可能会随时间变化，因而可能需要定时查询。 为简化这一步骤，可以使用

`iter_xflow_sub_instances` 方法

。该方法返回一个迭代器，会阻塞执行直至发现新的子 Instance 或者主 Instance 结束：

```
>>> # 此处建议使用异步调用
>>> inst = o.run_xflow('AlgoName', 'algo_public',
                       parameters={'param1': 'param_value1', 'param2':
 'param_value2', ...})
>>> for sub_inst_name, sub_inst in o.iter_xflow_sub_instances(inst):
 # 此处将等待
>>>     print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address
())))
```

在调用 run_xflow 或者 execute_xflow 时，也可以指定运行参数，指定的方法与 SQL 类似：

```
>>> parameters = {'param1': 'param_value1', 'param2': 'param_value2
', ...}
>>> o.execute_xflow('AlgoName', 'algo_public', parameters=parameters,
hints={'odps.xxx.yyy': 10})
```

使用 options.ml.xflow_settings 可以配置全局设置：

```
>>> from odps import options
>>> options.ml.xflow_settings = {'odps.xxx.yyy': 10}
>>> parameters = {'param1': 'param_value1', 'param2': 'param_value2
', ...}
>>> o.execute_xflow('AlgoName', 'algo_public', parameters=parameters)
```

PAI 命令的文档可以参考 这份文档 。

XFlow模型

在线模型是 ODPS 提供的模型在线部署能力。

- 在线模型

  用户可以通过 Pipeline 部署自己的模型。详细信息请参考“机器学习平台——在线服务”章节。

  需要注意的是，在线模型的服务使用的是独立的 Endpoint，需要配置 Predict Endpoint。通过

  ```
  >>> o = ODPS('your-access-id', 'your-secret-access-key', 'your-
  default-project',
  >>>           endpoint='your-end-point', predict_endpoint='predict_en
  dpoint')
  ```

  即可在 ODPS 对象上添加相关配置。Predict Endpoint 的地址请参考相关说明或咨询管理员。

- 部署离线模型上线

PyODPS 提供了离线模型的部署功能。部署方法为：

```
>>> model = o.create_online_model('online_model_name', 'offline_mo
del_name')
```

- 部署自定义 Pipeline 上线

  含有自定义 Pipeline 的在线模型可自行构造 ModelPredictor 对象，例子如下：

```
>>> from odps.models.ml import ModelPredictor, ModelProcessor,
BuiltinProcessor, PmmlProcessor, PmmlRunMode
>>> predictor = ModelPredictor(target_name='label')
>>> predictor.pipeline.append(BuiltinProcessor(offline_model_name='
sample_offlinemodel',
>>                                             offline_model_project
='online_test'))
>>> predictor.pipeline.append(PmmlProcessor(pmml='data_preprocess.
xml',
>>                                           resources='online_test/
resources/data_preprocess.xml',
>>                                           run_mode=PmmlRunMode.
Converter))
>>> predictor.pipeline.append(CustomProcessor(class_name='SampleProc
essor',
>>                                             lib='libsample_
processor.so',
>>                                             resources='online_test
/resources/sample_processor.tar.gz'))
>>> model = o.create_online_model('online_model_name', predictor)
```

  其中，BuiltinProcessor、PmmlProcessor 和 CustomProcessor 分别指 ODPS OfflineModel 形成的 Pipeline 节点、PMML 模型文件形成的 Pipeline 节点和用户自行开发的 Pipeline 节点。

- 在线模型操作

  与其他 ODPS 对象类似，创建后，可列举、获取和删除在线模型：

```
>>> models = o.list_online_models(prefix='prefix')
>>> model = o.get_online_model('online_model_name')
>>> o.delete_online_model('online_model_name')
```

  可使用模型名和数据进行在线预测，输入数据可以是 Record，也可以是字典或数组和 Schema 的组合：

```
>>> data = [[4, 3, 2, 1], [1, 2, 3, 4]]
>>> result = o.predict_online_model('online_model_name', data,
```

```
>>>                                     schema=['sepal_length', '
sepal_width', 'petal_length', 'petal_width'])
```

也可为模型设置 ABTest。参数中的 modelx 可以是在线模型名，也可以是 get_online_model 获得的模型对象本身，而 percentagex 表示 modelx 在 ABTest 中所占的百分比，范围为 0 至 100：

```
>>> result = o.config_online_model_ab_test('online_model_name',
model1, percentage1, model2, percentage2)
```

修改模型参数可以通过修改 OnlineModel 对象的属性，再调用 update 方法实现，如

```
>>> model = o.get_online_model('online_model_name')
>>> model.cpu = 200
>>> model.update()
```

与其他对象不同的是，在线模型的创建和删除较为耗时。PyODPS 默认 create_online_model 和 delete_online_model 以及 OnlineModel 的 update 方法在整个操作完成后才返回。用户可以通过 async 选项控制是否要在模型创建请求提交后立即返回， 然后自己控制等待。例如，下列语句

```
>>> model = o.create_online_model('online_model_name', 'offline_mo
del_name')
```

等价于

```
>>> model = o.create_online_model('online_model_name', 'offline_mo
del_name', async=True)
>>> model.wait_for_service()
```

而

```
>>> o.delete_online_model('online_model_name')
```

等价于

```
>>> o.delete_online_model('* online_model_name *', async=True)
>>> model.wait_for_deletion()
```

# 7.4 DataFrame

## 7.4.1 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 ODPS 的计算能力； 同时能在本地使用同样的接口，用 pandas 进行计算。

## 7.4.2 快速开始

在本例子中，我们拿 *movielens 100K* 来做例子。现在我们已经有三张表了，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）。

如果你的运行环境没有提供 ODPS 对象，你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
**'))
```

创建一个DataFrame对象十分容易，只需传入Table对象即可。

```
>>> from odps.df import DataFrame
>>> users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

我们可以通过dtypes属性来查看这个DataFrame有哪些字段，分别是什么类型

```
>>> users.dtypes
odps.Schema {
  user_id             int64
  age                 int64
  sex                 string
  occupation          string
```

```
  zip_code            string
}
```

通过head方法，我们能取前N条数据，这让我们能快速预览数据。

```
>>> 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
```

有时候，我们并不需要都看到所有字段，我们可以从中筛选出一部分。

```
>>> users[['user_id', 'age']].head(5)
   user_id  age
0        1   24
1        2   53
2        3   23
3        4   24
4        5   33
```

有时候我们只是排除个别字段。

```
>>> 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
```

又或者，排除掉一些字段的同时，得通过计算得到一些新的列，比如我想将sex为M的置为True，否则为False，并取名叫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
```

现在，让我们看看年龄在20到25岁之间的人有多少个

```
>>> users[users.age.between(20, 25)].count()
```

```
195
```

接下来，我们看看男女用户分别有多少。

```
>>> users.groupby(users.sex).agg(count=users.count())
   sex  count
0    F    273
1    M    670
```

用户按职业划分，从高到底，人数最多的前10职业是哪些呢？

```
>>> 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
```

DataFrame API提供了value_counts这个方法来快速达到同样的目的。

```
>>> 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
```

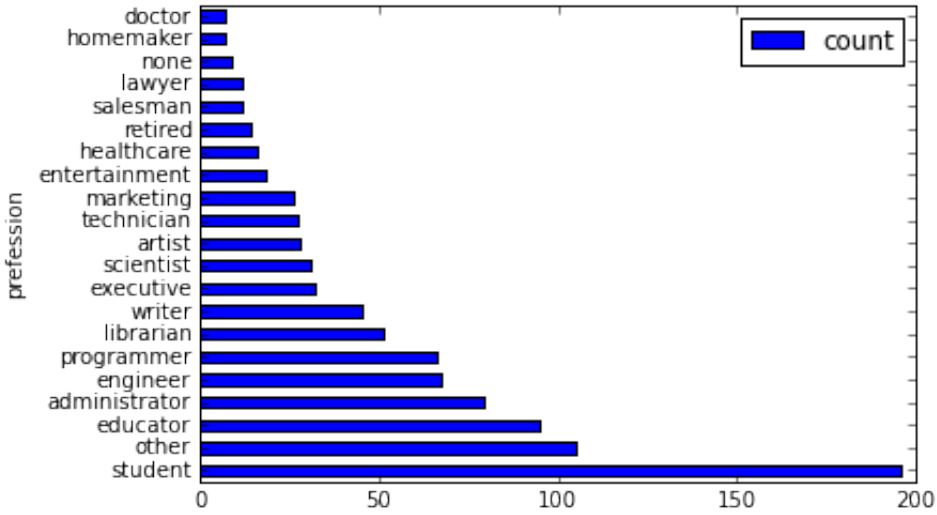
让我们用更直观的图来看这份数据。

```
>>> %matplotlib inline
```

我们可以用个横向的柱状图来可视化

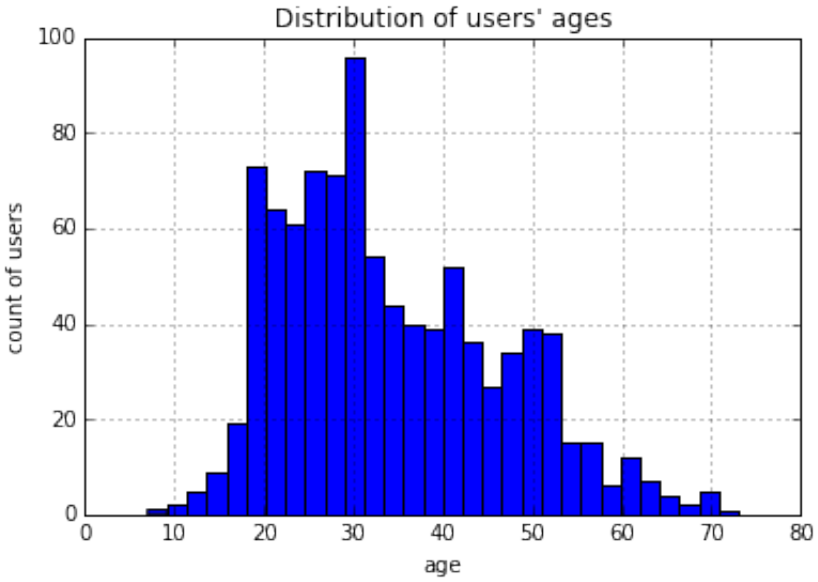
```
>>> users['occupation'].value_counts().plot(kind='barh', x='occupation
', ylabel='prefession')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10653cfd0>
```




我们将年龄分成30组，来看个年龄分布的直方图

```
>>> users.age.hist(bins=30, title="Distribution of users' ages",
xlabel='age', ylabel='count of users')
<matplotlib.axes._subplots.AxesSubplot at 0x10667a510>
```




好了，现在我们把这三张表联合起来，只需要使用join就可以了。join完成后我们把它保存成一张新的表。

```
>>> 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
}
```

现在我们把年龄分成从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')]
```

我们取分组和年龄唯一的前10条看看。

```
>>> 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
```

最后，我们来看看在各个年龄分组下，用户的评分总数和评分均值分别是多少。

```
>>> 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
```

## 7.4.3 创建 DataFrame

在使用 DataFrame 时，你需要了解三个对象上的操作：`Collection(DataFrame)`，`Sequence`，`Scalar`。 这三个对象分别表示表结构（或者二维结构）、列（一维结构）、标量。需要注意的是，这些对象仅在使用 Pandas 数据创建后会包含实际数据， 而在 ODPS 表上创建的对象中并不包含实际的数据，而仅仅包含对这些数据的操作，实质的存储和计算会在 ODPS 中进行。

### 创建 DataFrame

通常情况下，你唯一需要直接创建的 Collection 对象是 *DataFrame Reference*，这一对象用于引用数据源，可能是一个 ODPS 表， ODPS 分区，Pandas DataFrame或sqlalchemy.Table（数据库表）。 使用这几种数据源时，相关的操作相同，这意味着你可以不更改数据处理的代码，仅仅修改输入/输出的指向， 便可以简单地将小数据量上本地测试运行的代码迁移到 ODPS 上，而迁移的正确性由 PyODPS 来保证。

创建 DataFrame 非常简单，只需将 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
>>> iris = DataFrame(o.get_table('pyodps_iris'))
>>> iris2 = o.get_table('pyodps_iris').to_df()  # 使用表的to_df方法
>>>
>>> # 从 ODPS 分区创建
>>> 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
>>> 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
>>> 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)
```

在用 pandas DataFrame 初始化时，对于 numpy object 类型或者 string 类型，PyODPS DataFrame 会尝试推断类型， 如果一整列都为空，则会报错。这时，用户可以指定

unknown_as_string

为True，会将这些列指定为string类型。 用户也可以指定 as_type 参数。若类型为基本类型，会在创建 PyODPS DataFrame 时进行强制类型转换。 如果 Pandas DataFrame 中包含 list 或者 dict 列，该列的类型不会被推断，必须手动使用 as_type 指定。 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
}
```

## 7.4.4 Sequence

*SequenceExpr* 代表了二维数据集中的一列。你不应当手动创建 SequenceExpr，而应当从一个 Collection 中获取。

获取列

你可以使用 collection.column_name 取出一列，例如

```
>>> iris.sepallength.head(5)
   sepallength
0          5.1
1          4.9
2          4.7
3          4.6
4          5.0
```

如果列名存储在一个字符串变量中，除了使用 getattr(df, ‘column_name’) 达到相同的效果外，也可以使用 df[column_name] 的形式，例如

```
>>> iris['sepallength'].head(5)
   sepallength
0          5.1
1          4.9
2          4.7
3          4.6
```

```
4           5.0
```

列类型

DataFrame包括自己的类型系统，在使用Table初始化的时候，ODPS的类型会被进行转换。这样做的好处是，能支持更多的计算后端。 目前，DataFrame的执行后端支持ODPS SQL、pandas以及数据库（MySQL和Postgres）。

PyODPS DataFrame 包括以下类型：

`int8`，`int16`，`int32`，`int64`，`float32`，`float64`，`boolean`，`string`，`decimal`，`datetime`，`list`，`dict`

ODPS的字段和DataFrame的类型映射关系如下：

| **ODPS**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> |

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，未来的版本会支持。

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。例如，

```
>>> 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，一个 Sequence 必须要有列名。在很多情况下，DataFrame 会起一个名字。比如：

```
>>> iris.groupby('name').sepalwidth.max()
   sepalwidth_max
0             4.4
1             3.4
2             3.8
```

可以看到，`sepalwidth`取最大值后被命名为`sepalwidth_max`。还有一些操作，比如一个 Sequence 做加法，加上一个 Scalar，这时，会被命名为这个 Sequence 的名字。其它情况下，需要用户去自己命名。

Sequence 提供 rename 方法对一列进行重命名，用法示例如下：

```
>>> 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

你可以对一个 Sequence 进行运算，返回一个新的 Sequence，正如对简单的 Python 变量进行运算一样。对数值列， Sequence 支持四则运算，而对字符串则支持字符串相加等操作。例如，

```
>>> (iris.sepallength + 5).head(5)
   sepallength
0         10.1
1          9.9
2          9.7
3          9.6
4         10.0
```

而

```
>>> (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
```

注意到两列参与运算，因而 PyODPS 无法确定最终显示的列名，需要手动指定。详细的列变换说明，请参见 列运算。

## 7.4.5 Collection

DataFrame 中所有二维数据集上的操作都属于 *PyODPS DataFrame指南#DataFrame Reference#*，可视为一张 ODPS 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

如果要从一个 CollectionExpr 中选取部分列，产生新的数据集，可以使用 expr[columns] 语法。例如，

```
>>> 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 df[df.sepal_length, ] 或 df[[df.sepal_length]]，否则返回的将是一个 Sequence 对象，而不是 Collection。

如果想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
```

0.7.2 以后的 PyODPS 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
```

如果我们需要在已有 collection 中添加某一列变换的结果，也可以使用 expr[expr, new_sequence] 语法， 新增的列会作为新 collection 的一部分。

下面的例子将 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 df[df, new_sequence] 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。

0.7.2 以后版本的 PyODPS 支持直接在当前数据集中追加，写法为

```
>>> 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
```

我们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
```

对于 0.7.2 以后版本的 PyODPS，如果想在当前数据集上直接覆盖，则可以写

```
>>> 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
```

增删列以创建新 collection 的另一种方法是调用 select 方法，将需要选择的列作为参数输入。如果需要重命名，使用 keyword 参数输入，并将新的列名作为参数名即可。

```
>>> 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
```

此外，我们也可以传入一个 lambda 表达式，它接收一个参数，接收上一步的结果。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
```

此外，在 0.7.2 以后版本的 PyODPS 中，支持对数据进行条件赋值，例如

```
>>> 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，如

```
>>> 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，可以使用*NullScalar* ，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 及以后版本中，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

```
>>> 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，我们试着查询`sepallength`大于5的几条数据。

```
>>> 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
```

多个查询条件：

```
>>> 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：

记住，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
```

我们也可以显式调用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
```

同样对于连续的操作，我们可以使用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，记住对Collection取单个squence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，我们也支持Pandas中的query方法，用查询语句来做数据的筛选，在表达式中直接使用列名如sepallength进行操作， 另外在查询语句中&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支持的语法包括：

| 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。使用 apply 方法也可以输出多行。 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。

并列多行输出的例子如下：

```
>>> 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
```

关于 explode 使用并列输出的具体文档可参考 集合类型相关操作，对于 apply 方法使用并列输出的例子可参考 对一行数据使用自定义函数。

**限制条数**

```
>>> 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
```

值得注意的是，目前切片对于ODPS SQL后端不支持start和step。我们也可以使用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：

另外，切片操作只能作用在collection上，不能作用于sequence。

## 7.4.6 执行

延迟执行

DataFrame上的所有操作并不会立即执行，只有当用户显式调用`execute`方法，或者一些立即执行的方法时（内部调用的就是`execute`），才会真正去执行。

这些立即执行的方法包括：

| 方法 | 说明 | 返回值 |
| --- | --- | --- |
| persist | 将执行结果保存到ODPS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返回PyODPS DataFrame，False（默认）返回pandas DataFrame |
| plot，hist，boxplot | 画图有关 | |

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，这样省去了用户手动去调用execute。

```
>>> 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
```

如果想关闭自动调用执行，则需要手动设置

```
>>> 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
```

此时打印或者repr对象，会显示整棵抽象语法树。

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 **ODPS** 表

对 Collection，我们可以调用`persist`方法，参数为表名。返回一个新的DataFrame对象

```
>>> 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，这样会创建一个表，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
```

如果想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（如ds=******）。 这时要注意，该DataFrame的每个字段都必须在该表存在，且类型相

同。drop_partition和create_partition参数只有在此时有效，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，还可以指定表的生命周期，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，如：

```
>>> # 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**

我们可以使用 `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
```

**立即运行设置运行参数**

对于立即执行的方法，比如 `execute`、`persist`、`to_pandas` 等，可以设置运行时参数（仅对ODPS SQL后端有效 ）。

一种方法是设置全局参数。详细参考 *SQL*设置运行参数 。

也可以在这些立即执行的方法上，使用 `hints` 参数。这样，这些参数只会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**

有时，用户需要查看运行时instance的logview时，需要修改全局配置：

```
>>> 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
```

用户可以指定自己的日志记录函数，比如像这样：

```
>>> 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_iri
s` 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，一些Collection被多处使用，或者用户需要查看中间过程的执行结果，这时用户可以使用 cache标记某个collection需要被优先计算。

说明：

值得注意的是，cache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，即可以将一个操作异步执行，`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
```

可以看到上面的例子里，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` 操作来指定是否异步执行，当异步的时候，也可以指定 `timeout` 参数来指定超时时间。

## 7.4.7 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'))
```

对于一个Sequence来说，对它加上一个常量、或者执行sin函数的这类操作时，是作用于每个元素上的。接下来会详细说明。

### NULL相关（isnull，notnull，fillna）

DataFrame API提供了几个和NULL相关的内置函数，比如isnull来判断是否某字段是NULL，notnull则相反，fillna是将NULL填充为用户指定的值。

```
>>> 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，写法为：

```
>>> 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+，-，*，/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
```

算术运算支持的操作包括：

| 算术操作 | 说明 |
| --- | --- |
| abs | 绝对值 |
| sqrt | 平方根 |
| sin | |
| sinh | |
| cos | |
| cosh | |
| tan | |
| tanh | |
| arccos | |
| arccosh | |
| arcsin | |
| arcsinh | |
| arctan | |

| 算术操作 | 说明 |
| --- | --- |
| arctanh | |
| exp | 指数函数 |
| expm1 | 指数减1 |
| log | 传入参数表示底是几 |
| log2 | |
| log10 | |
| 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 |

对于sequence，也支持其于其他sequence或者scalar的比较。

```
>>> (iris.sepallength < 5).head(5)
   sepallength
0        False
1         True
2         True
3         True
4        False
```

值得主意的是，DataFrame API不支持连续操作，比如`3 <= iris.sepallength <= 5`，但是提供了between这个函数来进行是否在某个区间的判断。

```
>>> (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相关操作包括：

| **string** 操作 | 说明 |
| --- | --- |
| capitalize | |
| 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） |

| string 操作 | 说明 |
| --- | --- |
| 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 |
| 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） |

| string 操作 | 说明 |
| --- | --- |
| strptime | 按格式化读取成时间，时间格式和Python标准库相同，详细参考 *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
```

与时间相关的属性包括：

| 时间相关属性 | 说明 |
| --- | --- |
| year | |
| month | |
| day | |
| hour | |
| minute | |
| second | |
| 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 | 同 weekday |
| strftime | 格式化时间，时间格式和 Python 标准库相同，详细参考 *Python* 时间格式化 |

PyODPS 也支持时间的加减操作，比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
```

支持的时间类型包括：

| 属性 | 说明 |
| --- | --- |
| year | |
| month | |
| day | |
| hour | |
| minute | |
| second | |
| millisecond | |

### 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，另有 len 方法，可获得集合的大小。

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，例子如下：

```
>>> 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） |

其他元素操作（**isin**，**notin**，**cut**）

`isin`给出Sequence里的元素是否在某个集合元素里。`notin`是相反动作。

```
>>> 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
```

## 7.4.8 聚合操作

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

首先，我们可以使用`describe`函数，来查看DataFrame里数字列的数量、最大值、最小值、平均值以及标准差是多少。

```
>>> print(iris.describe())
    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
```

我们可以使用单列来执行聚合操作：

```
>>> iris.sepallength.max()
7.9
```

如果要在消除重复后的列上进行聚合，可以先调用 `unique`方法，再调用相应的聚合函数：

```
>>> iris.name.unique().cat(sep=',')
u'Iris-setosa,Iris-versicolor,Iris-virginica'
```

如果所有列支持同一种聚合操作，也可以直接在整个 DataFrame 上执行聚合操作：

```
>>> iris.exclude('category').mean()
   sepal_length  sepal_width  petal_length  petal_width
1      5.843333     3.054000      3.758667     1.198667
```

需要注意的是，在 DataFrame 上执行 count 获取的是 DataFrame 的总行数：

```
>>> iris.count()
150
```

PyODPS 支持的聚合操作包括：

| 聚合操作 | 说明 |
| --- | --- |
| count（或size） | 数量 |
| nunique | 不重复值数量 |
| min | 最小值 |
| max | 最大值 |

| 聚合操作 | 说明 |
| --- | --- |
| 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 |

需要注意的是，与 Pandas 不同，对于列上的聚合操作，不论是在 ODPS 还是 Pandas 后端下，PyODPS DataFrame 都会忽略空值。这一逻辑与 SQL 类似。

### 分组聚合

DataFrame API提供了groupby来执行分组操作，分组后的一个主要操作就是通过调用agg或者aggregate方法，来执行聚合操作。

```
>>> iris.groupby('name').agg(iris.sepallength.max(), smin=iris.
sepallength.min())
              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
```

最终的结果列中会包含分组的列，以及聚合的列。

DataFrame 提供了一个`value_counts`操作，能返回按某列分组后，每个组的个数从大到小排列的操作。

我们使用 groupby 表达式可以写成：

```
>>> iris.groupby('name').agg(count=iris.name.count()).sort('count',
ascending=False).head(5)
              name  count
0   Iris-virginica     50
1  Iris-versicolor     50
```

```
2      Iris-setosa     50
```

使用value_counts就很简单了：

```
>>> iris['name'].value_counts().head(5)
              name  count
0   Iris-virginica     50
1  Iris-versicolor     50
2      Iris-setosa     50
```

对于聚合后的单列操作，我们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()
   petallength_sum
0             73.2
1            213.0
2            277.6
```

```
>>> iris.groupby('name').agg(iris.petallength.notnull().sum())
              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()
   petallength_sum
0            563.8
```

### 编写自定义聚合

对字段调用agg或者aggregate方法来调用自定义聚合。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（比如 list、dict），buffer大小不应随数据而递增。
- __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。

让我们看一个计算平均值的例子。

```
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)
3.0540000000000007
```

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)
   petallength_aggregation
0                    3.418
1                    2.770
2                    2.974
```

当对多列调用自定义聚合，可以使用agg方法。

```
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]r[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)
              name       val
0      Iris-setosa  0.682781
1  Iris-versicolor  0.466644
```

```
2   Iris-virginica  0.451427
```

要调用 ODPS 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，受限于 Python UDF，自定义聚合无法支持将 list / 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`，这个接口是个近似的估计接口， 当数据量很大时，能较快的对数据的唯一个数进行估计。

这个接口在对比如海量用户UV进行计算时，能很快得出估计值。

```
>>> df = DataFrame(pd.DataFrame({'a': np.random.randint(100000, size=100000)}))
>>> df.a.hll_count()
63270
>>> df.a.nunique()
63250
```

提供 `splitter`参数会对每个字段进行分隔，再计算唯一数。

## 7.4.9 排序、去重、采样、数据变换

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

**排序**

排序操作只能作用于Collection。我们只需要调用sort或者sort_values方法。

```
>>> 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
```

如果想要降序排列，则可以使用参数ascending，并设为False。

```
>>> 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
```

也可以这样调用，来进行降序排列：

```
>>> 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
```

多字段排序也很简单：

```
>>> 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
```

下面效果是一样的：

```
>>> 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， 如果要排序尽量多的数据，可以把这个值设到较大的值。不过注意，此时可能会导致 OOM。

### 去重

去重在Collection上，用户可以调用distinct方法。

```
>>> 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
```

在Sequence上，用户可以调用unique，但是记住，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
```

下面的代码是错误的用法。

```
>>> iris[iris.name, iris.name.unique()]  # 错误的
```

### 采样

要对一个 collection 的数据采样，可以调用 `sample` 方法。PyODPS 支持四种采样方式。

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，用户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，用户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。例如，对数据

```
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，如果我们需要使输出值在 (-1, 1) 范围内，可使用

```
>>> 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。例如，

```
>>> 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。例如，

```
>>> 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
```

可见结果中，name1 和 name2 两组均按组中的最值进行了缩放。

std_scale 可依照标准正态分布对数据进行调整。例如，

```
>>> 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
```

std_scale 同样支持 preserve 参数保留原始列以及使用 group 进行分组，具体请参考 min_max_scale，此处不再赘述。

### 空值处理

DataFrame 支持筛去空值以及填充空值的功能。例如，对数据

```
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 可删除 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
```

你也可以使用 thresh 参数来指定行中至少要有多少个非空值。例如：

```
>>> 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
```

使用 fillna 可使用常数或已有的列填充未知值。下面给出了使用常数填充的例子：

```
>>> 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
```

你也可以使用一个已有的列来填充未知值。例如：

```
>>> 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
```

特别地，DataFrame 提供了向前 / 向后填充的功能。通过指定 method 参数为下列值可以达到目的：

| 取值 | 含义 |
| --- | --- |
| bfill / backfill | 向前填充 |
| ffill / pad | 向后填充 |

例如：

```
>>> 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
```

你也可以使用 ffill / bfill 函数来简化代码。ffill 等价于 fillna(method='ffill')， bfill 等价于 fillna(method='bfill')

## 7.4.10 对所有行/列调用自定义函数

要对一行数据使用自定义函数，可以使用 apply 方法，axis 参数必须为 1，表示在行上操作。

对一行数据使用自定义函数

apply 的自定义函数接收一个参数，为上一步 Collection 的一行数据，用户可以通过属性、或者偏移取得一个字段的数据。

```
>>> 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
```

`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。 如果类型不指定，将会默认为string类型。

在 apply 的自定义函数中，reduce 为 False 时，也可以使用 `yield`关键字来返回多行结果。

```
>>> 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
```

我们也可以在函数上来注释返回的字段和类型，这样就不需要在函数调用时再指定。

```
>>> 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
```

也可以使用 map-only 的 map_reduce，和 axis=1 的apply操作是等价的。

```
>>> iris.map_reduce(mapper=handle).count()
300
```

如果想调用 ODPS 上已经存在的 UDTF，则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，可以使用 使用自定义函数 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，当我们不指定axis，或者axis为0的时候，我们可以通过传入一个自定义聚合类来对所有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  petallengt
h_aggregation  petalwidth_aggregation
```

```
0                 5.843333                      3.054                   3.
758667                1.198667
```

说明：

目前，受限于 Python UDF，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

类似于对*map* 方法的resources参数，每个resource可以是ODPS上的资源（表资源或文件资源），或者引用一个collection作为资源。

对于axis为1，也就是在行上操作，我们需要写一个函数闭包或者callable的类。 而对于列上的聚合操作，我们只需在 __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
```

可以看到这里的stop_words是存放于本地，但在真正执行时会被上传到ODPS作为资源引用。

使用第三方**Python**库

使用方法类似*map*中使用第三方*Python*库 。

可以在全局指定使用的库：

```
>>> from odps import options
```

```
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

或者在立即执行的方法中，局部指定：

```
>>> df.apply(my_func, axis=1).to_pandas(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。因而建议在 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。

## 7.4.11 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。

DataFrame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，受限于 Python UDF，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，需要注意的是，函数外闭包变量值的变化会引起函数内该变量值的变化。

例如，

```
>>> 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，如

```
>>> dfs = []
>>> def get_mapper(i):
>>>     return lambda x: x + i
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(get_mapper(i)))
```

或

```
>>> 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 。

map传入Python函数的实现使用了ODPS Python UDF，因此，如果用户所在的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，第三方库（包含C）只能使用`numpy`，第三方库使用参考 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，如果用户所在Project未开通Python UDF，则这些函数也就无法使用（注：阿里云公共服务暂不提供Python UDF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 ODPS Worker 上执行时会发生错误。因而建议在 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。

### 引用资源

自定义函数也能读取ODPS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。

```
>>> 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。

使用第三方**Python**库

现在我们有个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
```

全局配置使用到的三方库：

```
>>> 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，通过立即运行方法的 `libraries`参数指定：

```
>>> def get_year(t):
>>>     from dateutil.parser import parse
>>>     return parse(t).strftime('%Y')
>>>
>>> df.datestr.map(get_year).execute(libraries=['six.whl', 'python_dat
eutil.whl'])
   datestr
0     2016
1     2015
```

PyODPS 默认支持执行纯 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 格式上传，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。下面的例子展示了如何上传并使用 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
```

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。

**调用ODPS内建或者已定义函数**

要想调用 ODPS 上的内建或者已定义函数，来生成列，我们可以使用 `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')]
>>> # 从其他 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：

注意：在使用 Pandas 后端时，不支持执行带有 `func`的表达式。

## 7.4.12 MapReduce API

PyODPS DataFrame也支持MapReduce API，用户可以分别编写map和reduce函数（map_reduce可以只有mapper或者reducer过程）。

首先我们来看个简单的wordcount的例子。

```
>>> def mapper(row):
>>>     for word in row[0].split():
>>>         yield word.lower(), 1
>>>
>>> def reducer(keys):
>>>     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
>>>
>>> words_df.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   hello    2
1       i    1
2      is    1
3    life    1
```

```
4  python    2
5   short    1
6     use    1
7   world    1
```

group参数用来指定reduce按哪些字段做分组，如果不指定，会按全部字段做分组。

其中对于reducer来说，会稍微有些不同。它需要接收聚合的keys初始化，并能继续处理按这些keys聚合的每行数据。 第2个参数表示这些keys相关的所有行是不是都迭代完成。

这里写成函数闭包的方式，主要为了方便，当然我们也能写成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来注释会让代码更简单些。

```
>>> 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):
>>>     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
>>>
>>> words_df.map_reduce(mapper, reducer, group='word')
     word  cnt
0   hello    2
1       i    1
2      is    1
3    life    1
4  python    2
5   short    1
6     use    1
7   world    1
```

有时候我们在迭代的时候需要按某些列排序，则可以使用 sort参数，来指定按哪些列排序，升序降序则通过 ascending参数指定。 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，我们就可以使用reducer作为combiner来先在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，我们能分别指定mapper和reducer所要引用的资源。

如下面的例子，我们对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_re
sources=[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 *map中使用第三方Python库* 。

可以在全局指定使用的库：

```
>>> from odps import options
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

或者在立即执行的方法中，局部指定：

```
>>> 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。因而建议在 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

有时候我们的数据在集群上分布可能是不均匀的，我们需要对数据重排。调用 `reshuffle`接口即可。

```
>>> df1 = df.reshuffle()
```

默认会按随机数做哈希来分布。也可以指定按那些列做分布，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` 接口来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，我们能对另外一个sequence2进行布隆过滤，sequence1不在sequence2中的一定会过滤， 但可能不能完全过滤掉不存在于sequence2中的数据，这也是一种近似的方法。

这样的好处是能快速对collection进行快速过滤一些无用数据。

这在大规模join的时候，一边数据量远大过另一边数据，而大部分并不会join上的场景很有用。 比如，我们在join用户的浏览数据和交易数据时，用户的浏览大部分不会带来交易，我们可以利用交易数据先对浏览数据进行布隆过滤， 然后再join能很好提升性能。

```
>>> 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
```

这里由于数据量很小，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，可能会有一定的数据不能被过滤。

如之前提的join场景中，少量不能过滤并不能并不会影响正确性，但能较大提升join的性能。

我们可以传入 `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提供透视表的功能。我们通过几个例子来看使用。

```
>>> 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
```

最简单的透视表必须提供一个rows 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，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
```

我们可以指定 values来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，用户可以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
```

我们也可以把原始数据的某一列的值，作为新的collection的列。 这也是透视表最强大的地方。

```
>>> 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
```

我们可以提供 fill_value 来填充空值。

```
>>> 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。

我们的数据为

```
>>> 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
```

可以通过 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。例如，相同的 df，

```
>>> 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
```

DataFrame 也支持将多列数据转换为一个 Key-Value 列。例如，

```
>>> 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
```

可通过 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
```

DataFrame创建及其对象操作请参见创建 *DataFrame*。

## 7.4.13 数据合并

```
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
  imdb_url                              string
}
```

```
ratings.dtypes
```

```
odps.Schema {
  user_id                    int64
  movie_id                   int64
  rating                     int64
  unix_timestamp             int64
}
```

**Join 操作**

DataFrame 也支持对两个 Collection 执行 join 的操作，如果不指定 join 的条件，那么 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
                                      imdb_url  user_id  rating
unix_timestamp
0         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
888068877
1         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
881444887
2         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
874833936
```

我们也可以显式指定join的条件。有以下几种方式：

```
>>> movies.join(ratings, on='movie_id').head(3)
```

```
   movie_id              title  release_date  video_release_date
                                      imdb_url  user_id  rating
unix_timestamp
0         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
888068877
1         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
881444887
2         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
874833936
```

在join时，on条件两边的字段名称相同时，只会选择一个，其他类型的join则会被重命名。

```
>>> movies.left_join(ratings, on='movie_id').head(3)
   movie_id_x              title  release_date  video_release_date
                                        imdb_url  user_id  movie_id_y
rating  unix_timestamp
0           3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...       49           3
       3       888068877
1           3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      621           3
       5       881444887
2           3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      291           3
       3       874833936
```

可以看到，`movie_id`被重命名为movie_id_x，以及movie_id_y，这和`suffixes`参数有关（默认是`('_x', '_y')`），当遇到重名的列时，就会被重命名为指定的后缀。

```
>>> 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
                                      imdb_url  user_id  rating
unix_timestamp  movie_id2
0         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
888068877          3
1         3  Four Rooms (1995)   01-Jan-1995                      http
://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
881444887          3
```

```
2         3  Four Rooms (1995)   01-Jan-1995                         http
://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
874833936          3
```

也可以直接写等于表达式。

```
>>> movies.join(ratings2, on=[movies.movie_id == ratings2.movie_id2]).
head(3)
   movie_id              title  release_date  video_release_date
                                       imdb_url  user_id  rating
unix_timestamp  movie_id2
0         3  Four Rooms (1995)   01-Jan-1995                         http
://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
888068877          3
1         3  Four Rooms (1995)   01-Jan-1995                         http
://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
881444887          3
2         3  Four Rooms (1995)   01-Jan-1995                         http
://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
874833936          3
```

self-join的时候，可以调用`view`方法，这样就可以分别取字段。

```
>>> 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

                                        imdb_url_x  movie_id2
0  http://us.imdb.com/M/title-exact?GoldenEye%20(...          2
1  http://us.imdb.com/M/title-exact?Four%20Rooms%...          3
2  http://us.imdb.com/M/title-exact?Get%20Shorty%...          4
```

除了`join`以外，DataFrame还支持`left_join`，`right_join`，和`outer_join`。在执行上述外连接操作时， 默认会将重名列加上 _x 和 _y 后缀，可通过在 suffixes 参数中传入一个二元 tuple 来自定义后缀。

如果需要在外连接中避免对谓词中相等的列取重复列，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)
```

要使用 **mapjoin**也很简单，只需将mapjoin设为True，执行时会对右表做mapjoin操作。

用户也能join分别来自ODPS和pandas的Collection，或者join分别来自ODPS和数据库的Collection，此时计算会在ODPS上执行。

**Union操作**

现在有两张表，字段和类型都一致（可以顺序不同），我们可以使用union或者concat来把它们合并成一张表。

```
>>> 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)
```

用户也能union分别来自ODPS和pandas的Collection，或者union分别来自ODPS和数据库的Collection，此时计算会在ODPS上执行。

## 7.4.14 窗口函数

DataFrame API也支持使用窗口函数。

```
>>> 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
```

窗口函数可以使用在列选择中：

```
>>> 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，和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支持的窗口函数包括：

| 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 |

其中，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 |
| ascending | 排序时，是否依照升序排序，默认 True |

lag 和 lead 除 rank 的参数外，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 |

而 cumsum、cummax、cummin、cummean、cummedian、cumcount 和 cumstd 除 rank 的上述参数外，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 |

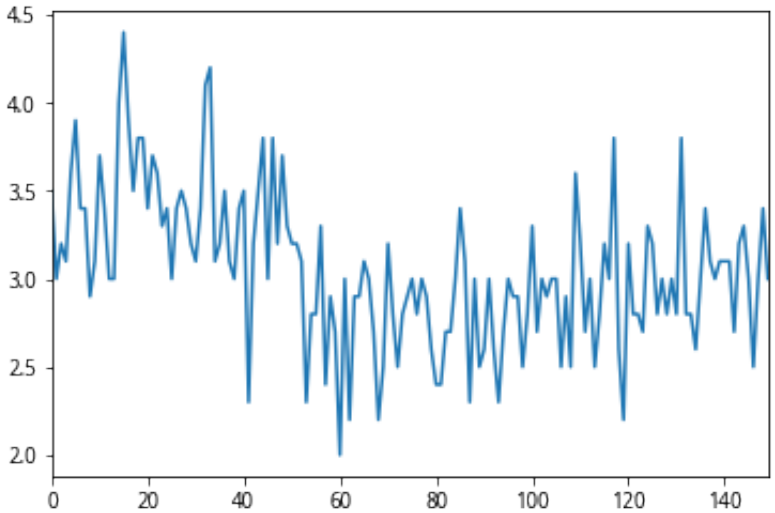
## 7.4.15 绘图

PyODPS DataFrame提供了绘图的方法。如果要使用绘图，需要 pandas 和 matplotlib 的安装。

绘图

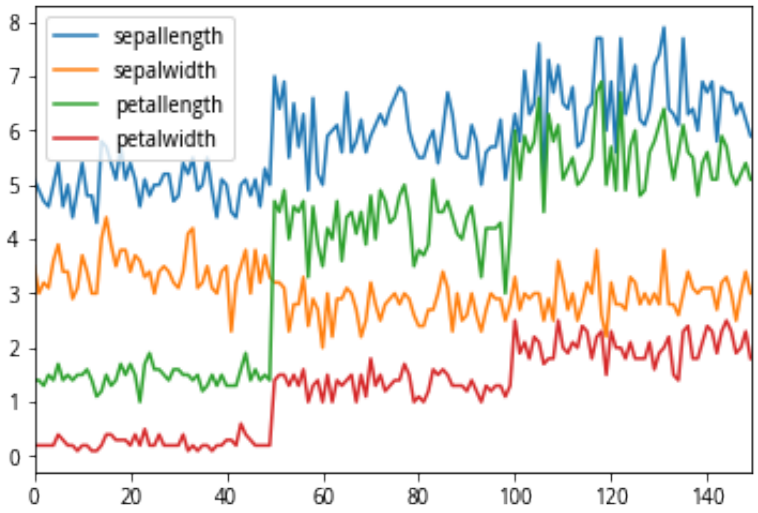
接下来的例子都是在jupyter中运行。

```
>>> from odps.df import DataFrame
>>> iris = DataFrame(o.get_table('pyodps_iris'))
>>> %matplotlib inline
>>> iris.sepalwidth.plot()
<matplotlib.axes._subplots.AxesSubplot at 0x10c2b3510>
```

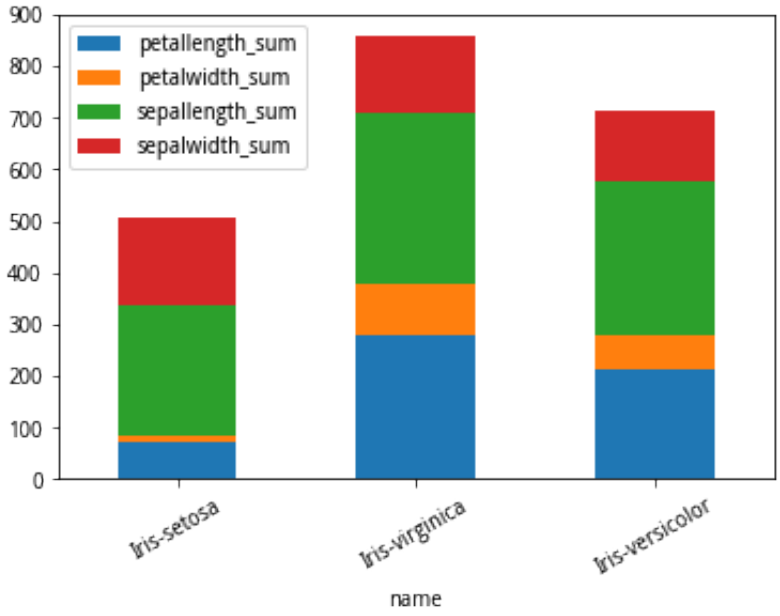



```
>>> iris.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10db7e690>
```

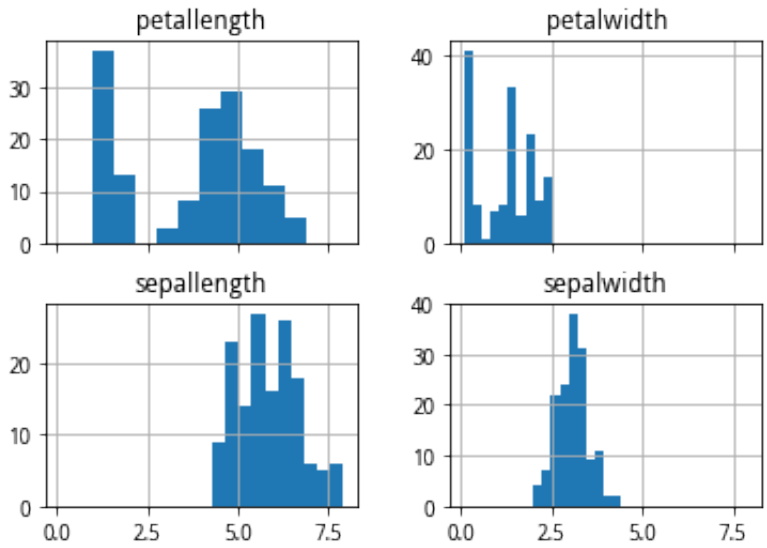



```
>>> iris.groupby('name').sum().plot(kind='bar', x='name', stacked=True
, rot=30)
<matplotlib.axes._subplots.AxesSubplot at 0x10c5f2090>
```




```
>>> 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表示了绘图的类型，支持的包括：

| kind | 说明 |
| --- | --- |
| line | 线图 |
| bar | 竖向柱状图 |
| barh | 横向柱状图 |
| hist | 直方图 |
| box | boxplot |
| kde | 核密度估计 |
| density | 和kde相同 |
| area | |
| pie | 饼图 |
| scatter | 散点图 |
| hexbin | |

详细参数可以参考Pandas文档：*pandas.DataFrame.plot*

除此之外，plot函数还增加了几个参数，方便进行绘图。

| 参数 | 说明 |
| --- | --- |
| xlabel | x轴名 |

| 参数 | 说明 |
| --- | --- |
| ylabel | y轴名 |
| xlabelsize | x轴名大小 |
| 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 |

## 7.4.16 调试指南

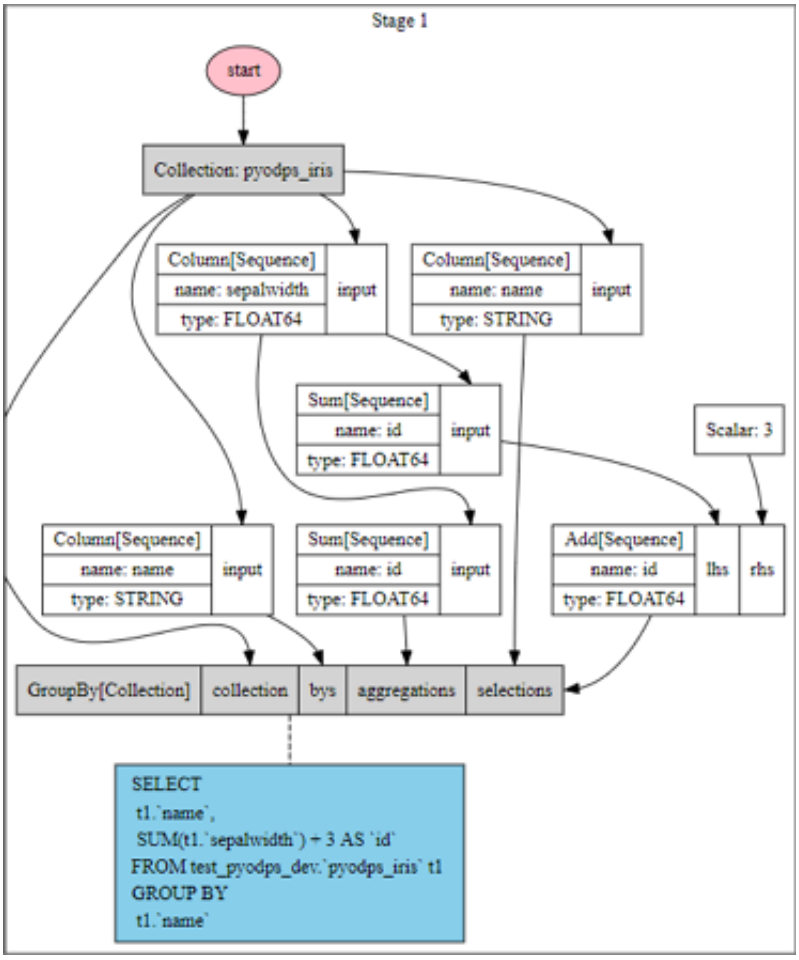
### 可视化DataFrame

由于PyODPS DataFrame本身会对整个操作执行优化，为了能直观地反应整个过程， 我们可以使用可视化的方式显示整个表达式的计算过程。

值得注意的是，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]
```

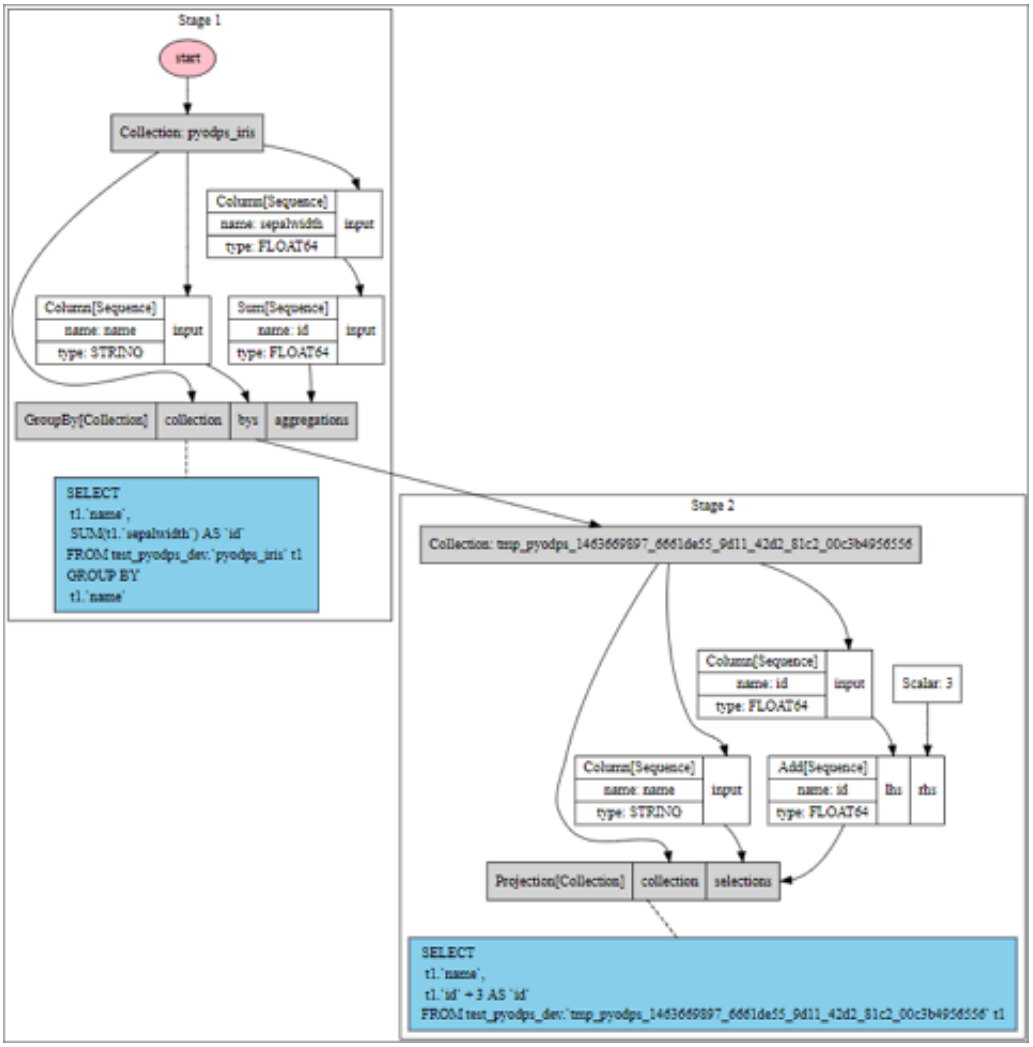
```
>>> df.visualize()
```




可以看到，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，由于用户执行了cache操作，这时整个执行计划将会分成两步来执行。

### ODPS SQL后端查看编译结果

我们可以直接调用 `compile`方法来查看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，一些操作不会compile到ODPS SQL执行，而是会使用Tunnel下载， 这个过程是很快的，且无需等待ODPS SQL任务的调度。 利用这个特性，我们能快速下载小部分ODPS数据到本地，使用pandas计算后端来进行代码编写和调试。

这些操作包括：

- 对非分区表进行选取整个或者有限条数据、或者列筛选的操作（不包括列的各种计算），以及计算其数量
- 对分区表不选取分区或筛选前几个分区字段，对其进行选取全部或者有限条数据、或者列筛选的操作，以及计算其数量

如我们的iris这个DataFrame的来源ODPS表是非分区表，以下操作会使用tunnel进行下载。

```
>>> iris.count()
>>> iris['name', 'sepalwidth'][:10]
```

对于分区表，如有个DataFrame来源于分区表（有三个分区字段，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)]
```

因此我们可以使用 `to_pandas` 方法来将部分数据下载到本地来进行调试，我们可以写出如下代码：

```
>>> DEBUG = True
```

```
>>> if DEBUG:
>>>     df = iris[:100].to_pandas(wrap=True)
>>> else:
>>>     df = iris
```

这样，当我们全部编写完成时，再把 `DEBUG`设置为False就可以在ODPS上执行完整的计算了。

 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。

## 7.5 交互体验增强

# 7.5.1 Jupyter Notebook 增强

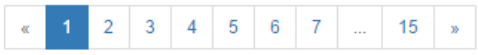
PyODPS 针对 Jupyter Notebook下的探索性数据分析进行了增强， 包括结果探索功能以及进度展示功能。

### 结果探索

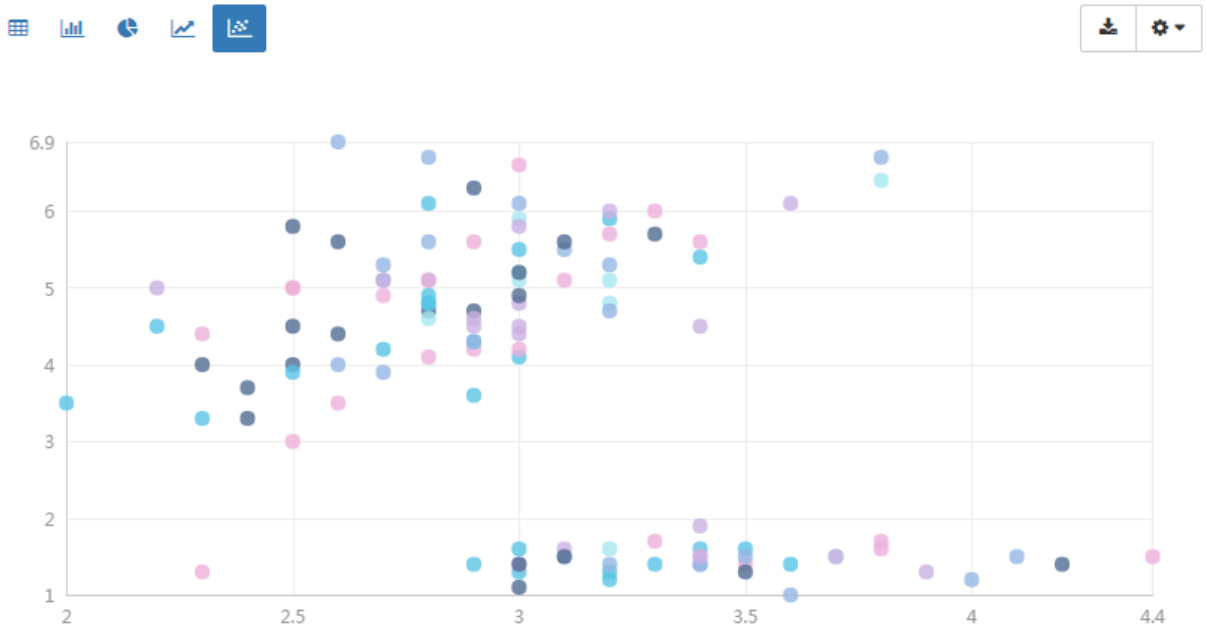
PyODPS 在 Jupyter Notebook 中为 SQL Cell 和 DataFrame 提供了数据探索功能。对于已拉到本地的数据，可使用交互式的数据探索工具 浏览数据，交互式地绘制图形。

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。单击页号或前进 / 后退按钮可在数据中导航， 如下图。

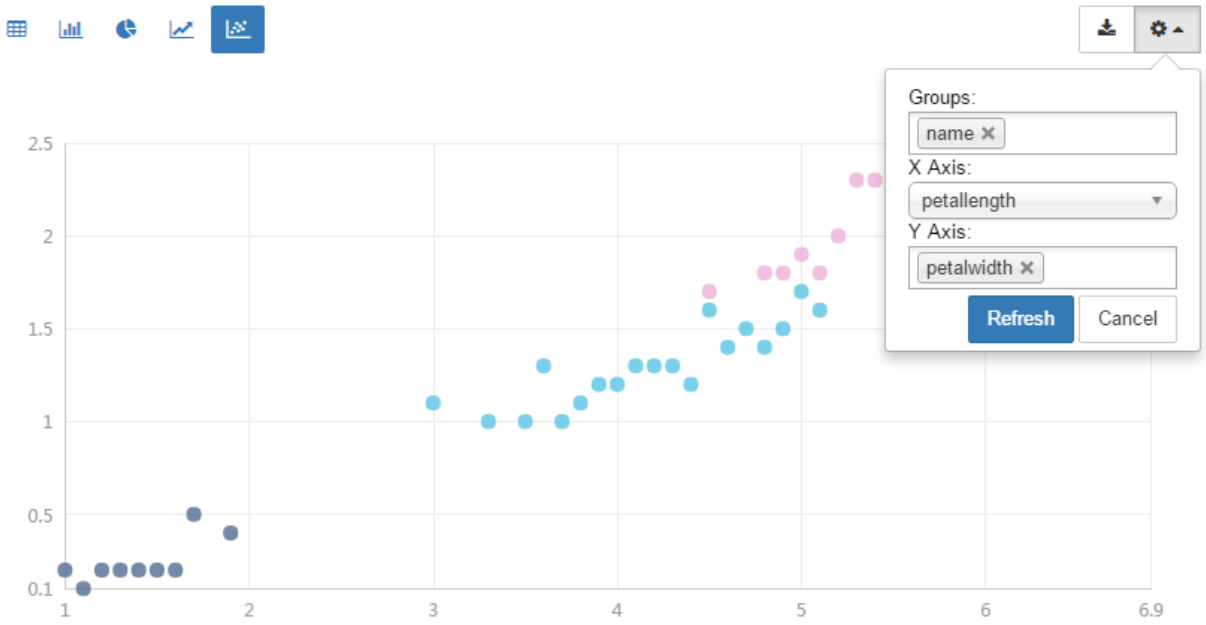
| sepallength | sepalwidth | petallength | petalwidth | name |
|---|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 4.9 | 3 | 1.4 | 0.2 | Iris-setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 5 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| 5.4 | 3.9 | 1.7 | 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 |



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。下图为使用默认字段选择（即前三个字段）绘制的散点图。

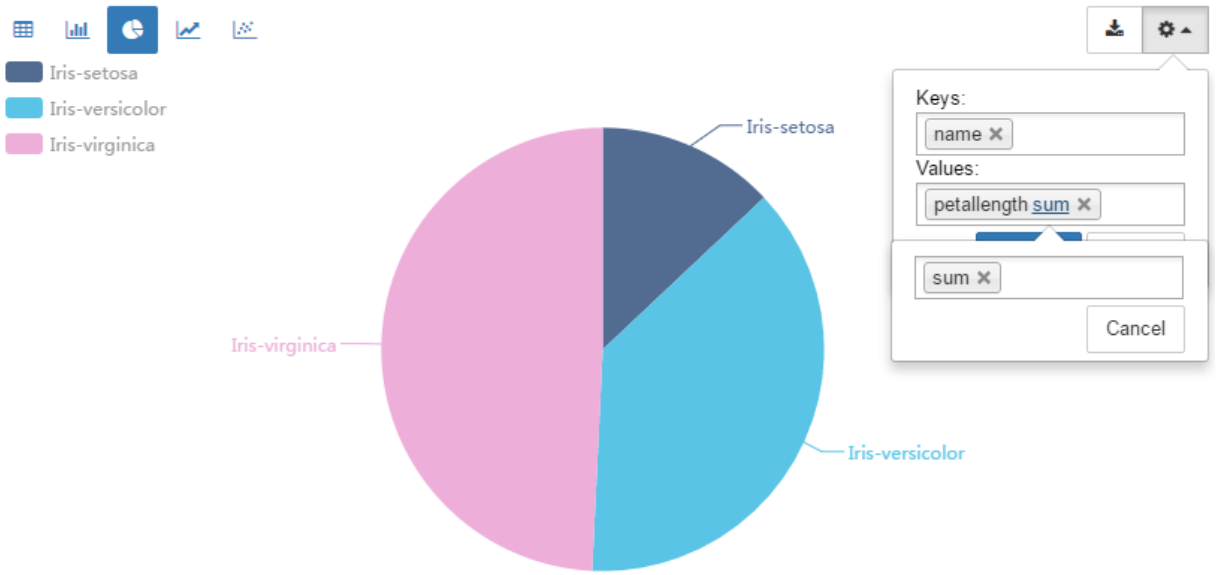



在绘图模式下，单击右上角的配置按钮可以修改图表设置。如下图中，将 name 设置为分组列，X 轴选择为 petallength，Y 轴选择为 petalwidth，则图表变为下图。可见在 petallength - petalwidth 维度下，数据对 name 有较好的区分度。




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， 可在值字段名称后的聚合函数名上单击，此后选择所需的聚合函数即可。

对于折线图，需要避免 X 轴包含空值，否则图像可能不符合预期。



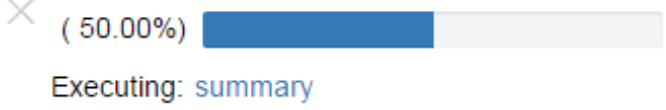

完成绘图后，可单击“下载”保存绘制的图表。

> **说明：**
> 注意：使用此功能需要安装 Pandas ，并保证 ipywidgets 被正确安装。

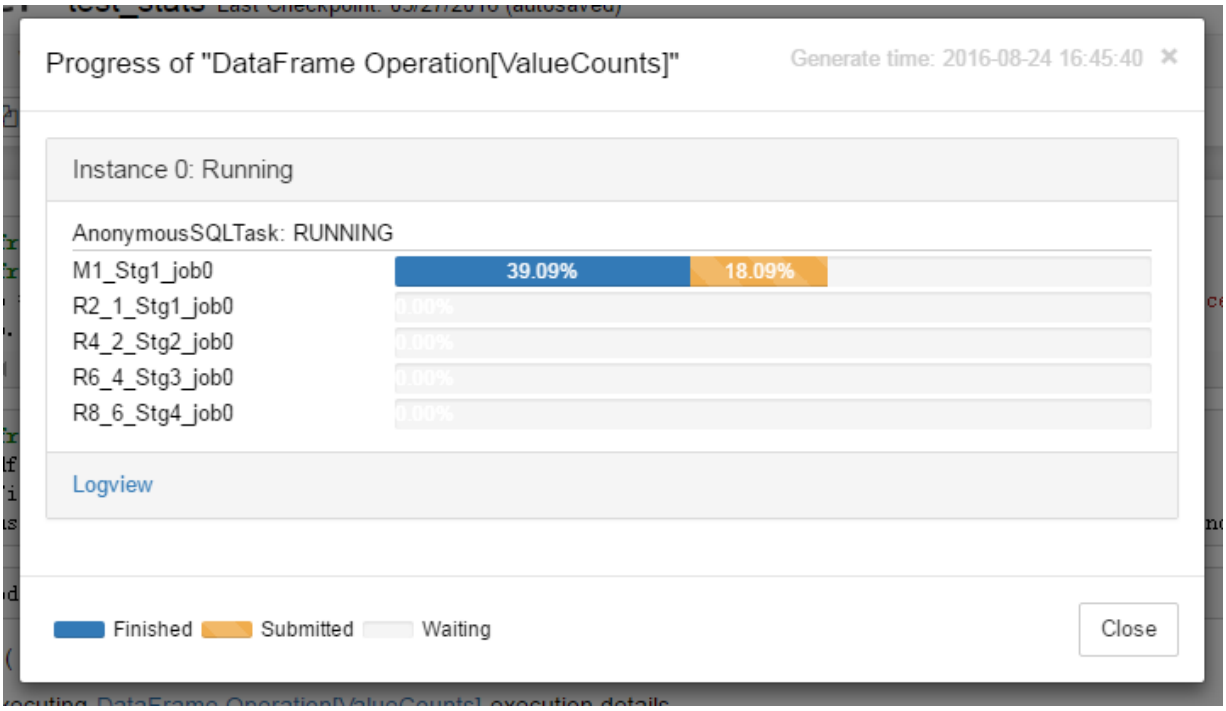
### 进度展示

大型作业执行通常需要较长的时间，因而 PyODPS 提供了进度展示功能。当 DataFrame、机器学习作业或通过 %sql 编写的 SQL 语句在 Jupyter Notebook 中执行作业时，会显示当前正在执行的作业列表及总体进度，如下图：

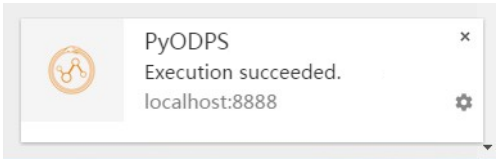
```
In [*]: from odps import ODPS
from odps.df.examples import create_ionosphere
o = ODPS(access_id, secret_access_key, project=project, endpoint=endpoint)
df = create_ionosphere(o)['a01', 'a02', 'a03', 'a04', 'class']
df.calc_summary()
```




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，如图：




当作业运行成功后，浏览器将给出提醒信息，告知作业是否成功：




## 7.5.2 IPython增强

PyODPS 还提供了 IPython 的插件，来更方便得操作 ODPS。

### IPython增强

首先，针对命令行增强，也有相应的命令。让我们先加载插件：

```
%load_ext odps
```

```
%enter
```

```
<odps.inter.Room at 0x11341df10>
```

此时全局会包含o和odps变量，即ODPS入口。

```
o.get_table('dual')
odps.get_table('dual')
```

```
odps.Table
  name: odps_test_sqltask_finance.`dual`
  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
```

```
%stores
```

| default | desc |
| --- | --- |
| **name** | |
| 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 `o.get_xxx` 这样的语句时，自动补全对象名。

例如，在 IPython 中输入下列语句（<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）：

```
o.get_table(<tab>
```

如果已知需要补全对象的前缀，也可以使用

```
o.get_table('tabl<tab>
```

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。下列用法都被支持：

```
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， 默认为 10。

### SQL命令

PyODPS 还提供了 SQL 插件，来执行 ODPS SQL。下面是单行 SQL：

```
%sql select * from pyodps_iris limit 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 |

多行SQL可以使用`%%sql`的命令

```
%%sql

select * from pyodps_iris
where sepallength < 5
limit 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 |

如果想执行参数化SQL查询，则需要替换的参数可以使用`:参数`的方式。

```
In [1]: %load_ext odps

In [2]: mytable = 'dual'

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运行时参数，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。

```
In [17]: %%sql
         set odps.sql.mapper.split.size = 16;
         select * from pyodps_iris;
```

这个会局部设置，不会影响全局的配置。

```
In [18]: %set odps.sql.mapper.split.size = 16
```

这样设置后，后续运行的SQL都会使用这个设置。

**持久化 pandas DataFrame 到 ODPS 表**

PyODPS 还提供把 pandas DataFrame 上传到 ODPS 表的命令:

```
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
```

这里的第0个参数df是前面的变量名，pyodps_pandas_df是ODPS表名。

## 7.5.3 命令行增强

PyODPS 提供了命令行下的增强工具。

首先，用户可以在任何地方配置了帐号以后，下次就无需再次输入帐号信息。

```
from odps.inter import setup, enter, teardown
```

接着就可以配置帐号

```
setup('**your-access_id**', '**your-access-key**', '**your-project
**', endpoint='**your-endpoint**')
```

在不指定room这个参数时，会被配置到叫做default的room里。

以后，在任何命令行打开的地方，都可以直接调用：

```
room = enter()
```

我们可以拿到ODPS的入口：

```
o = room.odps
```

```
o.get_table('dual')
```

```
odps.Table
  name: odps_test_sqltask_finance.`dual`
  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() 以获得新的 Room 对象。

我们可以把常用的ODPS表或者资源都可以存放在room里。

```
room.store('存储表示例', o.get_table('dual'), desc='简单的表存储示例')
```

我们可以调用`display`方法，来把已经存储的对象以表格的形式打印出来：

```
room.display()
```

| default | desc |
| --- | --- |
| **name** | |
| 存储表示例 | 简单的表存储示例 |
| iris | 安德森鸢尾花卉数据集 |

我们通过`room['存储表示例']`，或者像`room.iris`，就可以取出来存储的对象了。

```
room['存储表示例']
```

```
odps.Table
```

```
  name: odps_test_sqltask_finance.`dual`
  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
```

删除也很容易，只需要调用drop方法

```
room.drop('存储表示例')
```

```
room.display()
```

| default | desc |
| --- | --- |
| **name** | |
| iris | 安德森鸢尾花卉数据集 |

要删除某个room，只需要调用teardown就可以了，不传参数时删除默认room。

```
teardown()
```

## 7.6 配置选项

PyODPS 提供了一系列的配置选项，可通过 `odps.options` 获得，如下面的例子：

```
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
# PyODPS DataFrame 用 ODPS 执行时，参照下面 dataframe 相关配置，sort 时设置
limit 到一个比较大的值
options.df.odps.sort.limit = 100000000
```

下面列出了可配的 ODPS 选项。

**通用配置**

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| end_point | ODPS Endpoint | None |
| default_project | 默认 Project | None |

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| 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 | 1 |
| 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 |
| 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 Endpoint | None |
| tunnel.use_instance_tunnel | 使用 Instance Tunnel 获取执行结果 | True |

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| 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 |

| 选项 | 说明 | 默认值 |
| --- | --- | --- |
| ml.model_volume | 在使用 ModelTransfer 时使用的 Volume 名称 | pyodps_volume |

# 7.7 API Reference

## 7.7.1 API概述

我们将为您提供自动生成的PyODPS API文档。

- *ODPS详解#Definitions#*
- *PyODPS DataFrame指南#DataFrame Reference#*

# 8 MaxCompute管家

当开通MaxCompute预付费后，会遇到账号购买了150CU，但是经常看到使用预付费的项目很多任务依然要排队很长时间，管理员或运维人员希望能看到具体哪些任务抢占了资源，从而对任务进行合理的管控，如按任务对应的业务优先级进行调度时间调整，重要和次要任务错开调度的问题。

**MaxCompute** 管家就是解决这个预付费计算资源监控管理的问题。目前MaxCompute 管家主要提供三个功能，系统状态、资源组分配、任务监控。具体使用说明请参考DataWorks文档*MaxCompute*预付费资源监控工具-*CU*管家。

**说明：**

**MaxCompute**管家使用须知：

- 您需要购买了MaxCompute预付费CU资源，且购买数量为60CU或以上。CU过小无法发挥计算资源及管家的优势。