PowerBI星球

Data
Analysis
Expressions

PowerBI数据分析

# DAX函数卡片

Version 1.2

采悟 | 出品

「PowerBI星球」创始人、《PowerBI商业数据分析》作者

# 目录

PowerBI星球

DAX

Data 001
Analysis
Expressions

# CALCULATE

**注释：** 在由过滤器修改的上下文中计算表达式

**语法：**

CALCULATE ( 表达式，过滤1，过滤2，……)

**参数：**

第一个参数(必须)：返回计算结果，一般为聚合表达式

第二个参数/第三个参数……(可选)：可以没有过滤器，也可以有1个、2个……

**返回：**

值，根据第二个及之后的过滤器筛选后的上下文，返回第一个表达式计算的值。

**示例：**

1，计算硬盘的销售数量

```
CALCULATE(SUM('订单'[数量]),'订单'[产品名称]="硬盘")
```

2，计算所有产品的销售数量

```
CALCULATE(SUM('订单'[数量]),ALL('产品'))
```

3，计算2017年硬盘的销售数量

```
CALCULATE(
    SUM('订单'[数量]),
    FILTER(
        ALL('订单'),
        '订单'[产品名称]="硬盘"
        &&YEAR('订单'[订单日期])=2017
    )
)
```

**说明：**

CALCULATE函数是DAX中最重要也是最常用的函数，没有之一，
更多介绍可以参考文章：CALCULATE函数

# FILTER

**注释：** 返回经过表达式过滤后的表

**语法：**

```
FILTER ( 表，表达式)
```

**参数：**

```
第一个参数：需要过滤的表,可以是返回表的表达式
第二个参数：过滤条件
```

**返回：**

表，仅包含符合过滤条件行的表。

**示例：** 计算2017年硬盘的销售数量，两种方式：

```
方式1=
CALCULATE(
    SUM('订单'[数量]),
    FILTER(
        ALL('订单'),
        '订单'[产品名称]="硬盘"
        &&YEAR('订单'[订单日期])=2017
    )
)
```

```
方式2=
CALCULATE(
    SUM('订单'[数量]),
    FILTER(
        FILTER(
            ALL('订单’),
            '订单'[产品名称]="硬盘"
        ),
        YEAR('订单'[订单日期])=2017
    )
)
```

**说明：** 更多介绍可以参考文章：FILTER函数

# ALL

**注释：** 返回表中的所有行或列中的所有值

**语法：** ALL( 表）或者ALL(列,列，……）

**参数：** ALL的参数可以是表；也可以是表的一列或者表的多列

**返回：** 表

**示例：**

1，返回整个产品表(相当于复制表)

```
ALL('产品表')
```

2，返回订单表中不重复的产品名称（相当于VALUE）

```
ALL('订单'[产品名称])
```

3，返回订单表中产品名称和客户城市的所有组合

```
ALL('订单'[产品名称],'订单'[客户城市])
```

4，作为CALCULATE的参数，忽略上下文，返回累计销售量

```
CALCULATE(SUM('订单'[数量]),ALL('产品'))
```

# VALUES

**注释：** 参数为列时，返回该列不重复值的列表

**语法：**

VALUES( 表[列])

**参数：**

表中的列作为参数。

注：VALUES的参数也可以为表，但不常用，暂不介绍

**返回：**

表，只有一列的表；如果该表也只有一行，可以作为单个值使用。

**应用：**

1、从事实表中提取某一列，生成维度表

```
VALUES ( ‘订单表’[客户名称])
```

2、用于度量值，返回符合条件的文本型数据

```
IF ( HASONEVALUE ( ‘产品表’[产品名称]),
      VALUES ( ‘产品表’[产品名称] )
 )
```

3、将列转换为表，以便作为表函数的参数

```
ADDCOLUMNS(VALUES('产品'[产品名称]),"指标","产品")
```

**说明：**

VALUES看起来很简单，但使用起来十分灵活，通常情况下返回的是一个表，当该表只有一行一列时，它就成了一个值，因此经常用于返回文本值；
注意和DISTINCT函数区分，通常情况下这两个函数是一致的，当列中有空白值时，DISTINCT函数返回的结果不包含空白值，而VALUES函数可以返回空白值。

# SUMX

**注释：** 按表达式计算表中所有行的和

**语法：**

SUMX ( 表，表达式)

**参数：**

第一个参数为表，或者可以返回表的表达式；

第二个参数是求和的表达式，可以是一个列，也可以是列的表达式。

**应用：** ‘订单’表数据如下：

| 订单日期 | 订单号 | 产品名称 | 客户姓名 | 客户城市 | 发货日期 | 数量 | 单价 |
|---|---|---|---|---|---|---|---|
| 1/1/2016 | xs000001 | 充电宝 | 齐石 | 武汉市 | 1/2/2016 | 1 | 119 |
| 1/2/2016 | xs000002 | 数据线 | 周歌 | 上海市 | 1/2/2016 | 1 | 19 |
| 1/5/2016 | xs000003 | U盘 | 孙娜 | 北京市 | 1/6/2016 | 2 | 49 |
| 1/7/2016 | xs000004 | 键盘 | 许娜 | 上海市 | 1/8/2016 | 2 | 85 |
| 1/9/2016 | xs000005 | 硬盘 | 赵东 | 北京市 | 1/11/2016 | 1 | 349 |
| 1/10/2016 | xs000006 | 鼠标垫 | 王永 | 杭州市 | 1/11/2016 | 2 | 69 |

**1、计算‘订单’表中，所有产品的销售数量合计**

```
SUMX('订单','订单'[数量])
```

**2、计算‘订单’表中，“硬盘”的销售数量**

```
SUMX(
        FILTER('订单','订单'[产品名称]="硬盘"),
        '订单'[数量]
)
```

**3、计算‘订单’表中，“硬盘”的销售金额**

```
SUMX(
        FILTER('订单','订单'[产品名称]="硬盘"),
        ‘订单’[数量]*订单’[单价]
)
```

# AVERAGEX

**注释：** 在表中计算表达式的平均值

**语法：**

AVERAGEX ( 表，表达式)

**参数：**

第一个参数为表，或者可以返回表的表达式；

第二个参数是需要求平均值的表达式，可以是一个列，也可以是列的表达式。

**应用：** ‘订单’表如下，以及标准的日期表

| 订单日期 | 订单号 | 产品名称 | 客户姓名 | 客户城市 | 发货日期 | 数量 | 单价 | 销售额 |
|---|---|---|---|---|---|---|---|---|
| 1/1/2016 | xs000001 | 充电宝 | 齐石 | 武汉市 | 1/2/2016 | 1 | 119 | 119 |
| 1/2/2016 | xs000002 | 数据线 | 周歌 | 上海市 | 1/2/2016 | 1 | 19 | 19 |
| 1/5/2016 | xs000003 | U盘 | 孙娜 | 北京市 | 1/6/2016 | 2 | 49 | 98 |
| 1/7/2016 | xs000004 | 键盘 | 许娜 | 上海市 | 1/8/2016 | 2 | 85 | 170 |

[销售额]=SUM(‘订单’[销售额])

**1、计算每个订单的平均销售额**

```
AVERAGEX(‘订单‘,[销售额])
```

**2、计算每天的平均销售额**

```
AVERAGEX2 = AVERAGEX('日期表',[销售额])
```

**3、计算7天平均销售额**

```
AVERAGEX(
        DATESINPERIOD(
                '日期表'[日期],
                 MIN('日期表'[日期]),
                 -7,
                 DAY
         ),
        [销售额]
)
```

# COUNTROWS

**注释：** 计算表的行数

**语法：** COUNTROWS(表)

**参数：** 参数：表或者返回表的表达式

**返回：** 值，表的行数

**示例：**

```
COUNTROWS(订单表)//计算订单数量

COUNTROWS(
    FILTER(
      订单表,
      YEAR(订单表[订单日期])=2018
      &&订单表[产品名称]=“智能手表”
    )
)//计算2018年智能手表的订单数量
```

# RANKX

**注释：** 返回表中每行的表达式在表中的排名

**语法：**

RANKX(表，表达式，[value],[order],[ties])

**参数：**

第一个参数：表，可以是表表达式，排名对象；
第二个参数：为表中的每一行计算的表达式；
第三个参数：可选，当前行的排名依据，如省略，使用第二个参数的表达式；
第四个参数：可选，排序方式，忽略时默认降序，
0 / FALSE / DESC - 下降，　1 / TRUE / ASC - 升序
第五个参数：可选，值相同时排名确定方式：
Skip：跳跃排名，如果排名第3的有3个相等值，下一个排名是6，忽略时默认按skip确定排名；
Dense：连续排名，如果排名第3的有3个相等值，下一个排名是4。

**返回：**

值

**示例：** 在矩阵中对各产品按销售额进行排名

```
IF(
    HASONEVALUE('产品'[产品名称]),
    RANKX(ALL('产品'[产品名称]),[销售额])
)
```

**说明：** 更多介绍可以参考文章：RANKX函数

# MINX

**注释：** 按照表达式计算表的每一行，返回计算的最小值

**语法：**

MINX(表,表达式)

**参数：**

第一个参数：需要执行表达式计算的表
第二个参数：为表的每一行计算的表达式

**返回：**

值

**示例：**

1,返回订单表最早的订单日期

```
MINX(ALL('订单'),'订单'[订单日期])
```

2,计算订单表中硬盘的最低销售价格

```
MINX(
    FILTER(
        '订单',
        '订单'[产品名称]="硬盘“
    ),
    '订单'[单价]
)
```

**说明：**

MINX函数只计算数字和日期。如果表达式不产生数字，则MINX返回0。
计算时忽略空单元格，逻辑值和文本值。表示为文本的数字被视为文本。

# MAXX

**注释：** 按照表达式计算表的每一行，返回计算的最大值

**语法：**

```
MAXX(表,表达式)
```

**参数：**

第一个参数：需要执行表达式计算的表
第二个参数：为表的每一行计算的表达式

**返回：**

值

**示例：**

1,返回订单表中最后一个订单日期

```
MAXX(ALL('订单'),'订单'[订单日期])
```

2,计算订单表中硬盘的单笔销售额最大订单

```
MAXX(
    FILTER(
        '订单',
        '订单'[产品名称]="硬盘“
    ),
    ‘订单’[单价]* ‘订单’[数量]
)
```

**说明：**

MAXX函数只计算数字和日期。如果表达式不产生数字，则MAXX返回0。
计算时忽略空单元格，逻辑值和文本值。表示为文本的数字被视为文本。

# HASONEVALUE

**注释：** 当某一列被过滤为只有一个值时，返回TRUE.

**语法：**

```
HASONEVALUE ( ‘表’[列])
```

**参数：**

仅有一个参数，参数应为表中的一列，不能是表达式。

**返回：**

布尔值，TRUE/FALSE

**示例：** 一般作为IF函数的第一个参数使用,判断交叉筛选后是否为一个值

```
IF ( HASONEVALUE ( ‘产品表’[产品名称]),
      VALUES ( ‘产品表’[产品名称] )
 )
```

上述表达式的意思是，当产品表中的产品名称被当前的上下文交叉过滤后有且仅有一个值时，利用VALUES函数显示出这个值，否者，返回空值。

HASONEVALUE函数确保在当前上下文中有且仅有一个值，只有这样，度量值中的VALUES函数才能正确运算。

**说明：**

1、HASONEVALUE的内涵相当于下面这个表达式：
*COUNTROWS ( VALUES ( ‘表’[列]) ) = 1*

2、一般用于度量值，结合外部筛选灵活计算；

3、注意和HASONEFILTER函数区分。

# ISINSCOPE

**注释：** 当指定的列是级别层次结构中的级别时，返回true

**语法：** ISINSCOPE(列)

**参数：** 参数：必须是现有列的名称，不能是表达式

**返回：** 值，True/False

**示例：** 按级别计算百分比，以下是度量值和实现的效果

```
SWITCH (TRUE (),
    ISINSCOPE ('产品'[产品名称]),
        DIVIDE( [销售额],
            CALCULATE([销售额],
                ALLSELECTED('产品'[产品名称]))),
    ISINSCOPE ('产品'[产品类别]),
        DIVIDE([销售额],
            CALCULATE([销售额],ALLSELECTED('产品'))))
```

| 产品名称 | 产品类别 | 占比 ISINSCOPE |
|---|---|---|
| ■ U盘 | | |
| ■ VR眼睛 | 电脑外设 | **25.83%** |
| ■ 充电宝 | U盘 | 22.99% |
| ■ 耳机 | 键盘 | 40.51% |
| ■ 键盘 | 鼠标 | 36.50% |
| □ 手机壳 | 手机配件 | **27.58%** |
| ■ 鼠标 | 充电宝 | 52.48% |
| □ 鼠标垫 | 耳机 | 47.52% |
| □ 数据线 | 智能设备 | **46.59%** |
| □ 贴膜 | VR眼睛 | 67.98% |
| □ 硬盘 | 智能手环 | 32.02% |
| □ 智能手表 | | |
| ■ 智能手环 | | |

无论如何筛选：产品类别占比之和总是100%；

每个产品类别下级的产品明细占比之和也总是100%

**说明：** 更多介绍可以参考文章：按层级计算占比

# DISTINCT

**注释：** 返回不重复值的列表

**语法：**

```
DISTINCT( 表[列])
```

**参数：**

表中的列作为参数。

**返回：**

表，只有一列的表；如果该表也只有一行，可以作为单个值使用。

**示例：**

1、从事实表中提取某一列，生成维度表

```
DISTINCT ( '订单表'[客户名称])
```

2、用于度量值，返回符合条件的文本型数据

```
IF ( HASONEVALUE ( '产品表'[产品名称]),
      DISTINCT ( '产品表'[产品名称] )
 )
```

3、计算不重复的客户数量

```
COUNTROWS(DISTINCT('订单表'[客户名称]))
```

**说明：**

注意和VALUES函数区分，通常情况下这两个函数是一致的，当列中有空白值时，DISTINCT函数返回的结果不包含空白值，而VALUES函数可以返回空白值。

# DISTINCTCOUNT

**注释：** 计算列中不重复值的数量

**语法：**

```
DISTINCTCOUNT(表[列])
```

**参数：**

表的列

**返回：**

值

**示例：**

1，计算每天实现销售的产品种类的数量

```
CALCULATE(
    DISTINCTCOUNT('产品'[产品名称]),
    '订单’
)
```

2，计算某产品有销售的天数

```
CALCULATE(
    DISTINCTCOUNT('订单'[订单日期]),
    '订单’
)
```

3，计算不重复的客户数量

```
CALCULATE(
    DISTINCTCOUNT('客户'[客户代码])
    '订单’
)
```

**说明：**

该函数也可以用这个表达式替代
COUNTROWS(DISTINCT( ) )

# DATEADD

**注释：** 返回移动一定间隔后的日期区间

**语法：**

```
DATEADD (日期列,移动数量,移动粒度)
```

**参数：**

第一个参数：日期列，或者可以返回日期的表达式
第二个参数：整数，正数向前移动，负数向后回溯
第三个参数：粒度，year/quarter/month/day

**返回：** 单列日期值的表

**示例：**

**1,计算去年同期的销售额（同 SAMEPERIODLASTYEAR）**

```
CALCULATE(
    [销售额],
    DATEADD('日期表'[日期],-1,year)
)
```

**2,无外部上下文的情况下计算昨日销售额（比如在卡片图中动态显示昨天的销售）**

```
昨日销售额 =
CALCULATE(
    [销售额],
    DATEADD(
        TREATAS({TODAY()},日期表[日期]),
        -1,
        day
    )
)
```

# DATESINPERIOD

**注释：** 返回从指定日期移动一定间隔的时间段

返回类型：表(只有一列的表)

**语法：**

```
DATESINPERIOD ( 日期列，开始日期，移动间隔，移动粒度)
```

**参数：**

第一个参数:日期列
第二个参数:日期的表达式，作为开始日期
第三个参数:移动间隔，正数向未来移动，负数向过去移动
第四个参数:间隔粒度4个可选:DAY/MONTH/QUARTER/YEAR

**应用：**

1、返回过去7天的日期

```
DATESINPERIOD(
        '日期表'[日期],
         MIN('日期表'[日期]),
         -7,
          DAY
)
```

2、计算每月最后5天的销售额

```
CALCULATE(
        [销售额],
        DATESINPERIOD(
                '日期表'[日期],
                 ENDOFMONTH('日期表'[日期]),
                -5,
                DAY
         )
)
```

# DATESBETWEEN

**注释：** 返回从开始日期到结束日期的时间段

**语法：**

```
DATESBETWEEN (日期列,开始日期,结束日期)
```

**参数：**

第一个参数：日期列，或者可以返回日期的表达式
第二个参数：开始日期，可以是表达式
第三个参数：结束日期，可以是表达式

**返回：** 单列日期值的表

**示例：**

**1,计算2018年国庆放假期间的销售额**

```
CALCULATE(
    [销售额],
    DATESBETWEEN(
        '日期表'[日期],
        DATE(2018,10,1),
        DATE(2018,10,7)
    )
)
```

**2,计算年初至今的销售额**

```
昨日销售额 =
CALCULATE(
    [销售额],
    DATESBETWEEN(
        日期表[日期],
        STARTOFYEAR(日期表[日期]),
        LASTDATE(日期表[日期])
    )
)
```

# SELECTEDVALUE

**注释：** 上下文过滤为一个值时，返回该值

**语法：**

SELECTEDVALUE(表[列], 备用值(可选参数))

**参数：**

第一个参数必须：表的列，不能是表达式

第二个参数是可选，当上下文过滤后为空，或者多个值时，返回该备用值，不填时默认为BLANK().

**返回：**

值

**示例：**

**通常用于度量值，获取外部上下文筛选器，比如动态指标分析时，使用它获取外部切片器的信息：**

**当切片器选中“销售额”时，计算度量值[销售额]**

**当切片器选中“利润”时，计算度量值[利润]**

```
指标数据 =
SWITCH(TRUE(),
    SELECTEDVALUE('分析指标'[分析指标])="销售额",[销售额],
    SELECTEDVALUE('分析指标'[分析指标])="利润",[利润],
    [销售额]
)
```

**说明：** SELECTEDVALUE相当于下面这个函数组合：

*IF(HASONEVALUE(表[(列]), VALUES (表[(列]), 备用值)*

# SUMMARIZE

**注释：** 返回汇总表

**语法：**

SUMMARIZE (表,[分组列],汇总列名,汇总表达式，…)

**参数：**

第一个参数：需要汇总的表

第二个参数：可选，提取该列的非重复值列表，可以使用多个列，用逗号分隔，返回多个列的有效组合…

第三个参数：汇总列的列名;

第四个参数：汇总的表达式;

可以有更多的参数，功能和类型与第三和第四的参数类似，用于同时返回多个汇总值

**返回：**

表，汇总表

**示例：**

返回每个产品的销售额汇总表

```
SUMMARIZE(
    ‘订单表’,
     ‘订单表’[产品名称] ,
     “销售额合计”, SUM(‘订单表’[销售额])
)
```

**说明：**

该函数还有高级功能，使用ROLLUP参数和ROLLUPGROUP参数，来分别返回总计和小计

虽然它很方便，不过不建议使用，返回汇总表时，可以使用替代函数SUMMARIZECOLUMNS。

关于SUMMARIZE的更多介绍请参考公众号文章:
《SUMMARIZE：值得你花30分钟来了解的函数！》

**说明：**

更多介绍可以参考文章：SUMMARIZE函数

# CALCULATETABLE

**注释：** 在由过滤器修改的上下文中计算表达式

**语法：**

CALCULATETABLE ( 表达式，过滤1，过滤2，……)

**参数：**

第一个参数(必须)：表或者返回表的表达式

第二个参数/第三个参数……(可选)：可以没有过滤器，也可以有1个、2个……

**返回：**

表，根据第二个及之后的过滤器筛选后的上下文，返回第一个表达式计算的表。

**示例：**

1，返回所有硬盘的订单信息表

```
CALCULATETABLE('订单','订单'[产品名称]="硬盘")
```

2，返回销售额大于500的订单表

```
CALCULATETABLE(‘订单’, ‘订单’[销售额]>500)
```

3，计算2017年硬盘的销售数量

```
SUMX(
    CALCULATETABLE(
        '订单’,
        YEAR(‘订单’[订单日期])=2017,
        订单‘[产品名称]=“硬盘”
    ),
 ‘订单’[销售数量]
)
```

**说明：**

CALCULATETABLE与CALCULATE功能和运算逻辑相同，只是CALCULATE返回的是值，而CALCULATETABLE返回表。

# ALLSELECTED

**注释：** 保留除行和列过滤器之外的显式过滤器和上下文。可用于获取查询中的可视总计。

**语法：**

```
ALLSELECTED(表或者列)
```

**参数：**

参数可选，可以不用参数，或者用一个参数，参数可以是表(但不能是表的表达式)，也可以是列(不能是表达式)

**返回：**

上下文，除行和列过滤器之外的其他显式过滤器和上下文

**示例：**

1,计算某产品销售额占被筛选的产品合计销售额的比重

```
DIVIDE(
    [销售额],
    CALCULATE([销售额],ALLSELECTED('产品’))
)
```

2,计算某产品销售额占被筛选的所属分类销售额的比重

```
DIVIDE(
    [销售额],
    CALCULATE([销售额],ALLSELECTED('产品'[产品名称]))
)
```

**说明：**

该函数较为抽象，主要用来灵活控制上下文环境，最好根据示例来进行理解，参考公众号文章：利用ALL和ALLSELECTED灵活计算占比

# ALLEXCEPT

**注释：** 除参数列的过滤器外，忽略表中所有其他上下文的筛选

**语法：**

```
ALLEXCEPT (表,表[列],……)
```

**参数：**

第一个参数：忽略上下文的表

第二个参数：需要保留上下文的列

可以有第三个、第四个参数...，类型和功能同第二个参数

**返回：** 表

**示例：** ABC分析时，按产品类别统计ABC分类，因此需要按类别计算累计销售额，新建列的DAX代码：

```
按类别 累计=
CALCULATE(
    SUM('产品表'[销售额]),
    ALLEXCEPT('产品表','产品表'[产品类别]),
    '产品表'[销售额]>=EARLIER('产品表'[销售额])
)
```

结果如下：

| 产品类别 | 产品名称 | 销售额 | 按类别 累计 | 按类别 累计占比 |
|---|---|---|---|---|
| 智能设备 | 智能手环 | 234,558 | 732,604 | 82.17% |
| 智能设备 | VR眼睛 | 498,046 | 498,046 | 55.86% |
| 智能设备 | 智能手表 | 159,001 | 891,605 | 100.00% |
| 手机配件 | 充电宝 | 227,647 | 227,647 | 40.46% |
| 手机配件 | 手机壳 | 50,856 | 538,127 | 95.63% |
| 手机配件 | 耳机 | 206,119 | 433,766 | 77.08% |
| 手机配件 | 数据线 | 24,586 | 562,713 | 100.00% |
| 手机配件 | 贴膜 | 53,505 | 487,271 | 86.59% |

**说明：** 更多介绍可以参考文章：ALLEXCEPT函数

# SWITCH

**注释：** 用于条件判断,可替代IF函数,代码比IF更简洁

**语法：**

```
SWITCH ( <Expression>,
        <Value1>, <Result1> ,
        <Value2>, <Result2>,
         …,
        <Else>
)
```

**示例1：**

```
SWITCH (‘日期表’ [季度],
        1, “春季” ,
        2, “夏季” ,
        3, “秋季” ,
        4, “冬季” ,
        BLANK()
)
```

**示例2：**

```
SWITCH ( TRUE(),
        ‘数学’[分数]<60, “不及格” ,
        ‘数学’[分数]<75, “中等” ,
        ‘数学’[分数]<90, “良好” ,
        “优秀”
)
```

**判断条件是否相等的情况，推荐使用第一种写法；判断条件是大于或者小于的情况，使用第二种写法。**

# ADDCOLUMNS

**注释：** 为指定的表添加列

**语法：**

```
ADDCOLUMNS ( 表, 列名, 表达式 ,……)
```

**参数：**

第一个参数为表或者返回表的表达式，在该表中添加列；

第二个参数和第三个参数确定添加的第1列的列名和列值；

第四个参数和第五个参数确定添加的第2列的列名和列值；

以此类推……

**返回：** 表，包含原始表和新添加列的新表

**示例：** 生成日期表

```
ADDCOLUMNS (
    CALENDAR (DATE(2017,1,1), DATE(2018,12,31)),
    "年度", YEAR ( [Date] ),
    "季度", "Q" & FORMAT ( [Date], "Q" ),
    "月份", FORMAT ( [Date], "MM" ),
    "日",FORMAT ( [Date], "DD" ),
    "年度月份", FORMAT ( [Date], "YYYY/MM" ),
    "星期几", WEEKDAY ( [Date],2 )
)
```

# SELECTCOLUMNS

**注释：** 返回包含选定列的表

**语法：**

```
SELECTCOLUMNS ( 表, 列名, 表达式 ,……)
```

**参数：**

第一个参数为表或者可以返回表的表达式，从该表中选择列；

第二个参数和第三个参数确定选择的第1列的列名和列值；

第四个参数和第五个参数确定选择的第2列的列名和列值；

以此类推……

**返回：** 表，包含选择列的新表

**示例：** 从日期表中提取年度月份列（假设日期表中没有现成的年度月份）

```
SELECTCOLUMNS (
        ‘日期表’,
        “年度月份”, [年度]&[月份]
)
```

**说明：**

SELECTCOLUMNS函数 与ADDCOLUMNS 函数有很多相似之处，它们的参数完全一样，返回的也都是一个表。
这两个函数经常搭配一起使用。

# USERELATIONSHIP

**注释：** 激活模型中的候选关系

**语法：**

```
USERELATIONSHIP(列1，列2)
```

**参数：** 两个参数必须是现有表中的列，不能是表达式

**返回：** 不返回任何值，仅仅是激活关系

**示例：**

订单表中有订单日期和发货日期，其中订单日期与日期表表的日期建立了活动关系；

发货日期与日期表的日期列已建立了虚线关系（尚未激活），如果按照发货日期来计算销售额，可以这样写：

```
CALCULATE（
    [销售金额],
    USERELATIONSHIP(订单表[发货日期],日期表[日期])
）
```

**说明：**

参数列如果没有建立任何关系，或者属于不同的关系，将会返回错误；

USERELATIONSHIP只能用于以过滤器作为参数的函数，例如：CALCULATE，CALCULATETABLE，CLOSINGBALANCEMONTH，CLOSINGBALANCEQUARTER，CLOSINGBALANCEYEAR，OPENINGBALANCEMONTH，OPENINGBALANCEQUARTER，OPENINGBALANCEYEAR，TOTALMTD，TOTALQTD和TOTALYTD等函数

更多介绍可以参考文章：认识Power BI中的非活动关系

# TOPN

**注释：** 返回表的前N行

**语法：**

```
TOPN (N，表，排序依据，[排序类型])
```

**参数：**

第一个参数：要返回的行数;
第二个参数：需要提取的表，可是是返回表的表达式
第三个参数：对表排序的依据;
第四个参数：可选，默认为0，按降序排列;为1时，按升序排列。

**返回：** 表，前N行的表，如果N小于等于0，则返回空表

**示例：** 计算前10大订单的销售金额合计

```
CALCULATE(
    [销售金额],
    TOPN(
        10,
        ‘订单表’,
         ‘订单表’[订单金额]
    )
)
```

**说明：** 如果存在相等值的情况，则返回相等值的所有行，所以返回的行数有可能大于N行。

# SUMMARIZECOLUMNS

**注释：** 返回一个汇总表（可替代SUMMARIZE）

**语法：**

```
SUMMARIZECOLUMNS (
    [分组列]……,
    [表],
    汇总列名,汇总表达式，…)
```

**参数：**

第一组参数：提取该列的非重复值列表，可以使用多个列，用逗号分隔，返回多个列的有效组合…

第二组参数：可选，需要汇总的表

第三组参数：汇总列的列名和表达式，可以成对出现

**返回：** 表，汇总表

**示例：** 返回每个产品每月的销售额汇总表

```
SUMMARIZECOLUMNS(
    日期表[年度月份],
    '订单表'[产品名称] ,
    "销售额合计", SUM('订单表'[销售额])
)
```

**说明：**

该函数还有高级功能，使用ROLLUP参数和ROLLUPGROUP参数，来分别返回总计和小计；

返回汇总表时应熟练使用此函数，尽量避免使用SUMMARIZE

# LOOKUPVALUE

**注释：** 查找符合条件的单元格值

**语法：**

```
LOOKUPVALUE(结果列,查找列,查找值,[查找列,查找值])
```

**参数：**

第一个参数：返回结果的列名，不能是表达式
第二个参数：查找的列，不能是表达式
第三个参数：查找内容
第四个参数，第五个参数...,可选，同第二个和第三个参数，用于多条件查找，必须成对出现。

**返回：**

值，查找结果行与结果列交叉的单元格的值

**示例：** 查找产品类别为“智能设备”且价格为199元的产品名称

```
LOOKUPVALUE(
    '产品'[产品名称],
    '产品'[产品类别],"智能设备",
    '产品'[零售价],199
)
```

**说明：**

如果没有满足所有条件的值，则返回BLANK()
如果满足所有条件的有多行，且多行在结果列的值相等，则返回该值；如果不相等，则返回错误。

# FORMAT

**注释：** 根据指定的格式将值转换为文本

**语法：** FORMAT (值，文本格式)

**参数：**
第一个参数可以是数值或者返回数值的表达式;
第二个参数为带有格式化模板的字符串

**返回：** 值，文本型

**示例：** 常见的数字格式

**数字格式： 以1234.56为例**

| 格式参数 | 输出 |
|---|---|
| 0 | 1235 |
| 0.0 | 1234.6 |
| 0% | 123456% |
| 0.00% | 123456.00% |
| 0.00E+00 | 1.23E+03 |
| #,## | 1,235 |
| # , ## 0.00 | 1,234.56 |
| ¥ #,## 0.00 | ¥ 1,234.56 |
| General Number | 1234.56 |
| Currency | ¥ 1,234.56 |
| Standard | 1,234.56 |
| Percent | 123456.00% |
| Scientific | 1.23E+03 |



**说明：** 关于该函数的介绍和更多的格式类型，请参考公众号文章：
利用FORMAT函数自定义数据格式

# SEARCH

**注释：** 返回一个文本在另一个文本中的起始位置

**语法：**

SEARCH (查找内容，被查找内容，[开始位置],
[找不到时返回的值])

**参数：**

第一个参数：要查找的字符;
第二个参数：被查找的内容;
第三个参数：可选，开始搜索的位置，默认为1;
第四个参数：可选，查找不到时返回的值，通常为0，-1或者BLANK()

**返回：**

值

**示例：** 查找产品代码中”N”的位置

```
1  N的位置 = SEARCH("N",[产品代码],,0)
```

| 产品类别 | 产品名称 | 产品代码 | 零售价 | 采购价 | 单位利润 | N的位置 |
|---|---|---|---|---|---|---|
| 智能设备 | VR眼睛 | zn013 | 899 | 460 | 439 | 2 |
| 智能设备 | 智能手环 | zn012 | 249 | 188 | 61 | 2 |
| 智能设备 | 智能手表 | zn011 | 199 | 148 | 51 | 2 |
| 手机配件 | 数据线 | sj005 | 19 | 12 | 7 | 0 |
| 手机配件 | 贴膜 | sj004 | 29 | 10 | 19 | 0 |
| 手机配件 | 充电宝 | sj003 | 119 | 66 | 53 | 0 |
| 手机配件 | 耳机 | sj002 | 109 | 58 | 51 | 0 |
| 手机配件 | 手机壳 | sj001 | 26 | 12 | 14 | 0 |

**说明：**

SEARCH函数不区分大小写, 如果要严格区分大小写请使用FIND
SEARCH函数支持通配符， FIND函数不支持

更多介绍可以参考文章：SEARCH函数应用

# TREATAS

**注释：** 将一个表的列视为其他表的列来使用，一般用于创建虚拟关系

**语法：**

```
TREATAS(表表达式，列)
```

**参数：**

第一个参数必须是返回表的表达式

第二组参数是现有表的一列或者多列,列数应与第一个表达式返回的列相对应

**返回：**

表，第一个参数表达式返回的表

**示例：**

假设有两个日期表：日期表1和日期表2，日期表1的日期列与订单表中的订单日期相关联，日期表2与订单表没有关系。

如果想使用日期表2来筛选订单表，可以按如下的方式来写,即将日期表2的日期视同日期表1的日期来使用。

```
CALCULATE(
    [销售额],
    TREATAS(
        VALUES(日期表2[日期]),
        日期表1[日期])
    )
)
```

# UNION

**注释：** 把多个表合并成一个表

**语法：**

```
UNION ( 表1,表2,…)
```

**参数：**

参数为表，或者可以生成表的表达式，至少是2个表，也可以是2个以上的表。

**返回：**

表，各参数表中所有行合并的新表。

**应用：**

1、合并2个月的订单表，得到一个汇总表

```
UNION ( ‘1月订单表’,’2月订单表’)
```

2、合并1月、2月订单表中的客户列表

```
UNION(
        VALUES(‘1月订单表’[客户名称]),
        VALUES(‘2月订单表’[客户名称])
)
```

**说明：**

UNION函数的功能相当于纵向追加数据，和PQ的追加查询类似，需要说明的几点：

- 需要合并的每个表必须具有相同的列数；
- 返回表中的列名与第一个参数表中的列名称相同；
- 返回的表会保留重复的行。

# CROSSFILTER

**注释：** 指定两个表的关系方向

**语法：**

```
CROSSFILTER(左表[列]，右表[列]，方向类型)
```

**参数：**

第一个参数：左表的关系列，不能是表达式
第一个参数：右表的关系列，不能是表达式
第三个参数：方向类型：
0或者None:两个表没有关系
1或者Oneway：1对多的单向关系
2或者Both：双向关系

**返回：** 不返回任何值

**示例：**

数据模型中日期表和产品表都与订单表建立了一对多的关系，如何获得每个时间段在售的产品表中产品数量？外部上下文为日期。

由于订单表不能直接过滤产品表，要想计算出结果，可以去修改模型中的关系方向，如果不打算修改模型，可以使用DAX来指定关系

```
在售的产品数 =
 CALCULATE(
    COUNTROWS(产品表),
    CROSSFILTER('订单'[产品名称],'产品'[产品名称],2)
)
```

# RELATED

**注释：** 从关联的表中返回一个当前行的相关值

**语法：**

```
RELATED(表[列])
```

**参数：**

表的列

**返回：**

值

**示例：**

**1，订单表和客户表，已经通过客户名称建立关系，需要从客户表中匹配订单表中客户所在的城市**

```
客户城市= RELATED(‘客户‘[城市])
```

**2，订单表和客户表，已经通过客户建立了关系，计算山东的订单数量是多少**

```
山东客户数量=
CALCULATE(
    COUNTROWS(
        FILTER(‘订单表’,
            RELATED(‘客户‘[省份])=“山东”
        )
    )
)
```

**说明：**

RELATED函数根据当前的行上下文返回另一表中对应列的数据，适合把维度表的数据，匹配到事实表中，也就是沿着关系的多端找一端的值。

更多介绍可以参考文章：RELATED函数

# RELATEDTABLE

**注释：** 从关联的表中返回一个当前上下文的相关表

**语法：**

```
RELATEDTABLE(表)
```

**参数：**

表

**返回：**

表

**示例：**

1，订单表和客户表，已经通过客户名称建立关系，计算每个客户的订单数量

```
订单数量= COUNTROWS(RELATEDTABLE(‘订单表’))
```

2，订单表和客户表，已经通过客户建立了关系，计算每个客户的购买金额

```
客户购买金额=
SUMX(
  RELATEDTABLE (‘订单表’),
  ‘订单表’[数量]*’订单表’[单价]
 )
```

**说明：**

RELATEDTABLE的功能与RELATED类似，不过它返回的是一个表，它的参数也需要一个表，可以沿着关系的一端找多端的值。适合把事实表中的数据，匹配到维度表中，也就是沿着关系的一端找多端的表。

# LASTNONBLANK

**注释：** 返回非空值的列中的最后一个值

**语法：**

LASTNONBLANK(列，表达式)

**参数：**

第一个参数：表的列或者单列的表
第二个参数：表达式，计算非空的依据

**返回：**

表，一行一列的表，可以作为值使用

**示例：**

计算当月的库存余额。

假设库存表中有每个工作日的库存余额，由于月底最后一天可能不是工作日，所以库存表中可能不存在最后一天的数据，解决的办法如下

```
CALCULATE (
        SUM (库存表[库存余额] ),
        LASTNONBLANK (
            'Date'[Date],
            CALCULATE (SUM (库存表[库存余额] ))
        )
 )
```

# FIRSTNONBLANK

**注释：** 返回非空值的列中的第一个值

**语法：**

FIRSTNONBLANK(列，表达式)

**参数：**

第一个参数：表的列或者单列的表
第二个参数：表达式，计算非空的依据

**返回：**

表，一行一列的表，可以作为值使用

**示例：**

创建度量值返回销量最高的产品名称

使用TOPN，如果有多个产品的销售额相等且都是最高值，那么度量值将返回错误，为了避免这种错误,可以利用**FIRSTNONBLANK**来返回第一个产品名称

```
FIRSTNONBLANK (
    TOPN (
        1,
        VALUES ( 产品表[产品名称] ),
        [销售额]
    ),
    1
)
```

# IN

**注释：** 判断表中是否含有一行值，如果是则返回True,与CONTAINSROW等效

**语法：**

被查找的字段 IN 查找值的列表

**参数：**

严格来说，IN 是一个运算符

**返回：**

值，True/False

**示例：**

1，查找产品表中耳机和硬盘的产品信息

```
FILTER(
    '产品',
    [产品名称] IN {"耳机","硬盘"})
)
```

2，查找产品表中采购价为58的耳机的产品信息

```
FILTER(
    '产品',
    ([产品名称],[采购价]) IN {("耳机",58)}
)
```

3，查找产品表中除了耳机和硬盘的其他产品信息

```
FILTER(
    '产品',
    NOT [产品名称] IN {"耳机","硬盘"})
)
```

# CROSSJOIN

**注释：** 返回参数表的笛卡尔积

**语法：**

```
CROSSJOIN (表1，表2，……)
```

**参数：**

参数：任意的表或者返回表的表达式，可以有多个表

**返回：**

表，所有参数表行的笛卡尔积

**示例：**

将由一列姓组成的表‘姓’，和一列名组成的表‘名’，生成所有的姓名组合。

```
SELECTCOLUMNS(
    ADDCOLUMNS(
        CROSSJOIN('姓','名’),
        "姓名",[姓]&[名]
    ),
    "姓名",[姓名]
)
```

**说明：** CROSSJOIN参数各个表的列名不能相同，否则返回错误。

# DATEDIFF

**注释：** 计算两个日期之间的间隔数

**语法：**

```
DATEDIFF (
        开始日期,
        结束日期,
        间隔粒度
)
```

**通过第三个参数，可以计算各种粒度的时间间隔：**

SECOND：返回两个日期间隔的秒数
MINUTE：返回两个日期间隔的分钟
HOUR：返回两个日期间隔的小时
DAY：返回两个日期间隔的天数
WEEK：返回两个日期间隔的周数
MONTH：返回两个日期间隔的月数
QUARTER：返回两个日期间隔的季度
YEAR：返回两个日期间隔的年数

**应用：**

**假设日期表的日期有两个数据：**

| 日期 |
| --- |
| 2018-12-31 23:59:59 |
| 2019-01-01 00:00:00 |

**以下的表达式计算结果都为1**

```
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]),,SECOND )
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]), MINUTE )
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]), HOUR)
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]), DAY )
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]), WEEK )
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]), MONTH )
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]), QUARTER )
DATEDIFF(MIN(日期表[日期]), MAX(日期表[日期]), YEAR )
```

# CONCATENATE

**注释：** 将两个文本字符串连接成一个文本字符串

**语法：** CONCATENATE(文本1，文本2)

**参数：** 参数可以是文本、数字或者是列引用

**返回：** 值，连接的字符串

**示例：**

```
CONCATENATE (“Hello”,”PowerBI”) //返回 Hello PowerBI

CONCATENATE (产品表[产品编号],产品表[产品名称])
//将产品编号和产品名称合并在一起
```

# CONCATENATEX

**注释：** 将表中的每一行按表达式连接在一起

**语法：**

CONCATENATEX(表，表达式，分隔符)

**参数：**

第一个参数：计算表达式的表
第二个参数：为表的每一行计算的表达式
第三个参数：分隔符

**返回：**

值，连接的字符串

**示例：**

```
CONCATENATEX(订单表,[客户],“,”)

//将订单表中客户连接在一起，以逗号分隔
```

# INTERSECT

**注释：** 返回两个表的行交集，保留重复项

**语法：** INTERSECT(表1，表2)

**参数：**
第一个参数：左表
第二个参数：右表

**返回：** 表

**示例：** 以下表达式返回购买产品A并且产品B的客户

```
VAR customerA=
    CALCULATETABLE(
        VALUES('订单'[客户]),
        FILTER('订单','订单'[产品名称]="A")
    )
VAR customerB=
    CALCULATETABLE(
        VALUES('订单'[客户]),
        FILTER('订单','订单'[产品名称]="B")
    )
RETURN
INTERSECT(customerA,customerB)
```

**说明：**
INTERSECT函数的两个参数表有左右顺序区别，它返回左表中的所有交集行。
注意与NATURALINNERJOIN的区别：
INTERSECT函数要求两个参数表的列数相同；
NATURALINNERJOIN的两个参数表的列可以不同。

# NATURALINNERJOIN

**注释：** 内部联结，返回两个表公共列的交集及其他列

**语法：**

```
NATURALINNERJOIN (左表，右表)
```

**参数：**

第一个参数：表或者表表达式，左表
第二个参数：表或者表表达式，右表

**返回：**

表，包含公共列的交集和所有的其他列

**示例：** 以下表达式，返回产品A和产品B的共同客户数量

```
VAR customerA=
    SUMMARIZE(
        FILTER( '订单', ‘订单’[产品名称]=“产品A" ),
        '订单'[客户姓名]
    )
VAR customerB=
    SUMMARIZE(
        FILTER( ‘订单’, ‘订单’[产品名称]=“产品B” ),
        '订单'[客户姓名]
    )
RETURN
    COUNTROWS(
        NATURALINNERJOIN(customerA, customerB))
```

# SAMEPERIODLASTYEAR

**注释：** 去年同期

**语法：**

```
SAMEPERIODLASTYEAR (日期列)
```

**参数：** 参数：日期列，或者可以返回日期的表达式

**返回：** 表，包含当前上下文去年同期的日期的表

**示例：**

1,计算去年同期的销售额

```
CALCULATE(
    [销售额],
    SAMEPERIODLASTYEAR('日期表'[日期])
)
```

2,计算上年的年初至今销售额

```
CALCULATE(
    [销售额],
    SAMEPERIODLASTYEAR(DATESYTD('日期表'[日期]))
)
```

**说明：** 该函数看起来很长，实际上非常简单，并且从函数名就可以看出它的功能，很容易记。

它实际上与以下表达式功能相同：
DATEADD (‘日期表’[日期],-1,year）

# TOTALYTD

**注释：** 返回年初至今的表达式的值

**语法：**

```
TOTALYTD（表达式,日期列[,筛选条件],年度结束日期)
```

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件
第四个参数：可选，默认为12-31，主要用于财年计算

**返回：** 值

**示例：** 计算季初至今的销售额

```
TOTALYTD(
    [销售额],
    '日期表'[日期],
)
```

**说明：** 该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    [销售额],
    DATESYTD( '日期表'[日期])
)
```

# UNICHAR

**注释：** 返回数值引用的Unicode字符

**语法：**

```
UNICHAR (数值)
```

**参数：**

参数：整数，表示字符的Unicode编号

**返回：**

值，一个字符

**示例：**

```
UNICHAR(65)   //返回字符"A"
UNICHAR(9733)   //返回字符"★"
```

常用字符的Unicode编号

8593：↑
8595：↓
8599：↗
8600：↘
9650：▲
9660：▼
9679：●
9733：★
9734：☆
10004：✓
10006：✕

**说明：** 更多介绍可以参考文章：PowerBI实现红绿灯效果

# DATE

**注释：** 返回日期格式

**语法：**

```
DATE (年，月，日)
```

**参数：**

第一个参数：年份数
第二个参数：月份
第三个参数：日期（不能为0，否则报错）

**返回：**

值，日期型的值

**示例：**

**1，将数值型的年月日字段联结为日期型的数据**

```
DATE([年度],[月份],[日])
```

**2，如果第一个参数为两位数的年度，结果将把这个数加上1900作为年度。下面的表达式将返回1919年2月15日**

```
DATE( 19, 2, 15)
```

**3，如果第二个参数小于1或者大于12，结果将会把差异调整到年度上。下面的表达式将返回2020年2月15日**

```
DATE( 2019, 14, 15)
```

**4，如果第三个参数不是1-31的数字，结果将会把差异调整到月份上。下面的表达式将返回2019年1月28日**

```
DATE( 2019, 2, -3)
```

# GENERATE

**注释：** 在第一个参数表的每一行计算第二个表的表达式，返回交叉连接表

**语法：**

GENERATE(表1，表2)

**参数：**

第一个参数：基表，作为迭代器
第二个参数：在基表中的每一行计算的表表达式

**返回：**

表

**示例：** 以下表达式生成每一年每个产品的销售额汇总表

```
GENERATE(
    VALUES('日期表'[年度]),
    ADDCOLUMNS(
        VALUES('订单'[产品名称]),
        "销售额合计",CALCULATE(SUM('订单'[销售额]))
    )
)
```

# CALENDAR

**注释：** 返回一个表，其中包含一组连续的日期，列名为”Date”

**语法：**

```
CALENDAR(开始日期, 结束日期)
```

**参数：**

第一个参数是开始日期，第二个参数结束日期;
参数可以是返回日期的表达式。

**返回：**

表，只有一列连续日期的表

**示例：**

**1,返回2019年全年日期的表**

```
CALENDAR (DATE (2019, 1, 1), DATE (2019, 12, 31))
```

**2,返回包含订单表全部日期的连续日期列**

```
CALENDAR (
    MINX (订单表, 订单表[订单日期]),
    MAXX (订单表, 订单表[订单日期])
)
```

**3,生成2019-2020年的简易日期表**

```
ADDCOLUMNS (
        CALENDAR (DATE(2019,1,1), DATE(2020,12,31)),
        "年度", YEAR ( [Date] ),
        "季度", "Q" & FORMAT ( [Date], "Q" ),
        "月份", FORMAT ( [Date], "MM" ),
        "日",FORMAT ( [Date], "DD" ),
        "年度月份", FORMAT ( [Date], "YYYY/MM" ),
)
```

**说明：** 如果第二个日期参数小于第一个日期参数，则返回错误

# HASONEFILTER

**注释：** 当外部上下文仅有一个筛选器时，返回TRUE.

**语法：**

```
HASONEFILTER ( ‘表’[列])
```

**参数：**

仅有一个参数，参数应为表中的一列，不能是表达式。

**返回：**

布尔值，TRUE/FALSE

**示例：**

**一般作为IF函数的第一个参数使用，判断外部上下文是否有且仅有一个过滤器，**

```
IF ( HASONEFILTER ( 日期表[月份]),
      [销售额],
      ERROR(“请单选切片器”)
 )
```

上述表达式的意思是，当外部的月份切片器，被单选时，正常计算度量值[销售额]；如果月份切片器被多选、或者不选，均返回错误提示。

**说明：**

1、HASONEFILTER的内涵相当于下面这个表达式：
*COUNTROWS ( FILTERSS ( ‘表’[列]) ) = 1*

2、一般用于度量值，结合外部切片器灵活计算；

3、注意和HASONEVALUE函数区分,区别在于HASONEVALUE基于交叉过滤器工作，而HASONEFILTER通过直接过滤器工作

# CALENDARAUTO

**注释：** 返回一个表，根据模型自动生成整年的日期范围，列名为”Date”

**语法：** CALENDARAUTO()或者 CALENDARAUTO(财年结束月份)

**参数：**
可以不输入参数;
当生成财年日期时，可以输入1-12的数字作为参数
(输入3，3月就是财年的最后一个月，默认是12)

**返回：** 表，只有一列连续日期的表

**示例：**

**假设数据模型中，最小日期是2015.7.1，最大日期是2019.1.12**
**下面的表达式，将生成2015-1-1到2019-12-31的日期表。**

```
ADDCOLUMNS (
    CALENDARAUTO (),
    "年度", YEAR ( [Date] ),
    "季度", "Q" & FORMAT ( [Date], "Q" ),
    "月份", FORMAT ( [Date], "MM" ),
    "日",FORMAT ( [Date], "DD" ),
    "年度月份", FORMAT ( [Date], "YYYY/MM" ),
)
```

**生成财年的日期表，模型 日期同上**
**下面的表达式，将生成2014-10-1到2019-9-30的日期表。**

```
ADDCOLUMNS (
    CALENDARAUTO (9),
    "年度", YEAR ( [Date] ),
    "季度", "Q" & FORMAT ( [Date], "Q" ),
    "月份", FORMAT ( [Date], "MM" ),
    "日",FORMAT ( [Date], "DD" ),
    "年度月份", FORMAT ( [Date], "YYYY/MM" ),
)
```

# DATESMTD

**注释：** 月初至今

**语法：**

```
DATESMTD(日期列)
```

**参数：** 参数是日期列，也可以是返回日期列的表达式

**返回：** 表，包含从当前上下文的月初至今的日期列的表

**示例：**

1,计算月初至今的销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESMTD('日期表'[日期])
)
```

2,计算上年同期的月初至今销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESMTD(SAMEPERIODLASTYEAR(‘日期表’[日期]))
)
```

# DATESQTD

**注释：** 季初至今

**语法：** DATESQTD(日期列)

**参数：** 参数是日期列，也可以是返回日期列的表达式

**返回：** 表，包含从当前上下文的季初至今的日期列的表

**示例：** 计算季度初至今的销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESQTD('日期表'[日期])
)
```

2,计算上年同期的季初至今销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESQTD(SAMEPERIODLASTYEAR(‘日期表’[日期]))
)
```

# DATESYTD

**注释：** 年初至今

**语法：**

```
DATESYTD(日期列,[年度结束日期])
```

**参数：**

第一个参数是日期列，也可以是返回日期列的表达式；
第二个参数可选，不填时默认为12-31；可用于财年日历计算。

**返回：**

表，包含从当前上下文的年初至今的日期列的表

**示例：**

**1,计算季年初至今的销量**

```
CALCULATE(
    SUM('订单表'[数量]),
    DATESYTD('日期表'[日期])
)
```

**2,计算上年同期的年初至今销量**

```
CALCULATE(
    SUM('订单表'[数量]),
    DATESYTD(SAMEPERIODLASTYEAR('日期表'[日期]))
)
```

**3,按财年日历计算年初至今销量，财年结束日期9月30日**

```
CALCULATE(
    SUM('订单表'[数量]),
    DATESYTD('日期表'[日期]),"9-30")
)
```

# FIRSTDATE

**注释：** 返回当前上下文的第一个日期

**语法：**

```
FIRSTDATE(日期列)
```

**参数：** 参数是日期列，也可以是返回日期列的表达式

**返回：** 表，一行一列的表，也可以用作值

**示例：**

1,计算当前上下文第一天的的销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    FIRSTDATE('日期表'[日期])
)
```

2,返回产品第一次销售的日期

```
FIRSTDATE(‘订单表’[订单日期])
```

**说明：** FIRSTDATE函数和LASTDATE正好相反，不过如果当前上下文为单个日期时，二者返回结果相同。（与MAX和MIN的原理类似）

# LASTDATE

**注释：** 返回当前上下文的最后一个日期

**语法：**

```
LASTDATE(日期列)
```

**参数：** 参数是日期列，也可以是返回日期列的表达式

**返回：** 表，一行一列的表，也可以用作值

**示例：**

**1,计算当前上下文最后一天的的销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    LASTDATE('日期表'[日期])
)
```

**2,返回产品最后一次销售的日期**

```
LASTDATE(‘订单表’[订单日期])
```

**说明：** LASTDATE函数和FIRSTDATE正好相反，不过如果当前上下文为单个日期时，二者返回结果相同。（与MAX和MIN的原理类似）

# PREVIOUSDAY

**注释：** 返回当前上下文的第一个日期之前的日期

**语法：**

```
PREVIOUSDAY(日期列)
```

**参数：** 参数是日期列，也可以是返回日期列的表达式

**返回：** 表，一行一列的表，也可以用作值

**示例：**

1,计算当前上下文的上一天的销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    PREVIOUSDAY ('日期表'[日期])
)
```

2,当前上下文是月份，和LASTDATE结合可以返回本月倒数第二天的销量。

```
CALCULATE(
    SUM(‘订单表’[数量]),
    PREVIOUSDAY(LASTDATE('日期表'[日期]))
)
```

**说明：** PREVIOUSDAY函数与NEXTDAY相对应，NEXTDAY返回下一天；

# PREVIOUSMONTH

**注释：** 基于当前上下文的第一个日期，返回上个月所有日期的表

**语法：**

```
PREVIOUSMONTH(日期列)
```

**参数：**

参数是日期列，也可以是返回日期列的表达式

**返回：**

表，包含整个月的日期的表

**示例：**

**1,计算当前上下文的上个月的销量,假如当前上下文是1月19号，该函数将返回上年12月整月的数据**

```
CALCULATE(
    SUM('订单表'[数量]),
    PREVIOUSMONTH ('日期表'[日期])
)
```

**2,可以通过嵌套，返回上上个月的数据**

```
CALCULATE(
    SUM('订单表'[数量]),
    PREVIOUSMONTH(
        PREVIOUSMONTH('日期表'[日期])
    )
)
```

**说明：** 无论外部上下文是什么时间粒度，总是返回一整月的数据

# PREVIOUSQUARTER

**注释：** 基于当前上下文的第一个日期，返回上个季度所有日期的表

**语法：**

```
PREVIOUSQUARTER(日期列)
```

**参数：** 参数是日期列，也可以是返回日期列的表达式

**返回：** 表，包含整个季度的日期的表

**示例：**

**1,计算当前上下文的上个季度的销量,假如当前上下文是1月20号，下述表达式将返回上年第4季度的数据**

```
CALCULATE(
    SUM('订单表'[数量]),
    PREVIOUSQUARTER ('日期表'[日期])
)
```

**2,计算季度环比(外部上下文为季度)**

```
VAR  cur_quarter = CALCULATE([数量],
                        DATESQTD ('日期表'[日期]))

VAR  pre_quarter = CALCULATE([数量],
                        PREVIOUSQUARTER('日期表'[日期]))

RETURN DIVIDE(cur_quarter, pre_quarter)-1
```

**说明：** 无论外部上下文是什么时间粒度，总是返回一整季度的数据

# PREVIOUSYEAR

**注释：** 基于当前上下文的最后一个日期，返回上年所有日期的表

**语法：**

```
PREVIOUSYEAR(日期列, ,[年度结束日期])
```

**参数：**

第一个参数是日期列，也可以是返回日期列的表达式

第二个参数可选，不填时默认为12-31；可用于财年日历计算。

**返回：**

表，包含整年的日期的表

**示例：**

**1,计算当前上下文的上年的销量,假如当前上下文是1月21号，下述表达式将返回上年全年的数据**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    PREVIOUSYEAR('日期表'[日期])
)
```

**2,按财年日历计算上年的销量，假如当前上下文是2019年1月21号，下述表达式将返回2017年10月1日至2018年9-30日的数据。**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    PREVIOUSYEAR(‘日期表’[日期],”9-30”)
)
```

**说明：** 无论外部上下文是什么时间粒度，总是返回一整年的数据

# NEXTDAY

**注释：** 返回次日，基于当前上下文第一个日期

**语法：**

```
NEXTDAY(日期列)
```

**参数：**

日期列，或者可以返回日期的表达式

**返回：**

表，只有一个日期的表

**示例：**

**1,计算次日销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    NEXTDAY('日期表'[日期])
)
```

**2,计算次日到月底的销量(加上DATESMTD正好是整月数据）**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESBETWEEN (
        ‘日期表’[日期],
        NEXTDAY('日期表'[日期]),
        ENDOFMONTH('日期表'[日期])
    )
)
```

**说明：** 与PREVIOUSDAY 函数相对应

# NEXTMONTH

**注释：** 返回次月所有日期的表，基于当前上下文第一个日期

**语法：**

```
NEXTMONTH(日期列)
```

**参数：**

日期列，或者可以返回日期的表达式

**返回：**

表，包含整月日期的表

**示例：**

1,计算次月销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    NEXTMONTH('日期表'[日期])
)
```

2,计算次月到年底的销量(加上DATESYTD正好是整年数据）

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESBETWEEN (
        ‘日期表’[日期],
        NEXTMONTH('日期表'[日期]),
        ENDOFYEAR('日期表'[日期])
    )
)
```

**说明：** 与PREVIOUSMONTH函数相对应

# NEXTQUARTER

**注释：** 返回下一个季度所有日期的表，基于当前上下文第一个日期

**语法：**

```
NEXTQUARTER(日期列)
```

**参数：**

日期列，或者可以返回日期的表达式

**返回：**

表，包含整季度日期的表

**示例：**

1,计算次季度销量

```
CALCULATE(
    SUM(‘订单表’[数量]),
    NEXTQUARTER('日期表'[日期])
)
```

2,计算次季度到年底的销量(加上DATESYTD正好是整年数据）

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESBETWEEN (
        ‘日期表’[日期],
        NEXTMQUARTER('日期表'[日期]),
        ENDOFYEAR('日期表'[日期])
    )
)
```

**说明：** 与PREVIOUSQUARTER函数相对应

# NEXTYEAR

**注释：** 返回下一年所有日期的表，基于当前上下文第一个日期

**语法：**

```
NEXTYEAR(日期列,[年度结束日期])
```

**参数：**

第一个参数：日期列，或者可以返回日期的表达式
第二个参数：可选，默认为12-31，主要用于财年计算

**返回：** 表，包含整年日期的表

**示例：**

1,计算下一年销量

```
CALCULATE(
    SUM('订单表'[数量]),
    NEXTYEAR('日期表'[日期])
)
```

2,计算下一财年的销量，假如当前上下文为2017年1月25日，财年结束日期为9月30，下述表达式将返回2017年10月1日至2018年9月30日的数据

```
CALCULATE(
    SUM('订单表'[数量]),
    NEXTYEAR('日期表'[日期],"9-30")
)
```

**说明：** 与PREVIOUSYEAR函数相对应

# ENDOFMONTH

**注释：** 返回当前上下文月份的最后一天

**语法：**

```
ENDOFMONTH(日期列)
```

**参数：**

第一个参数：日期列，或者可以返回日期的表达式

**返回：**

表，只有一个日期的表

**示例：**

**1,计算每月最后一天的销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    ENDOFMONTH('日期表'[日期])
)
```

**2,计算每月最后5天的销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESINPERIOD(
        '日期表'[日期],
        ENDOFMONTH('日期表'[日期]),
        -5,
        DAY
    )
)
```

**说明：**

*如果当前上下文是季度或者年，则返回该粒度的最后一个月的最后一天。*

# ENDOFQUARTER

**注释：** 返回当前上下文季度的最后一天

**语法：**

```
ENDOFQUARTER(日期列)
```

**参数：**

第一个参数：日期列，或者可以返回日期的表达式

**返回：**

表，只有一个日期的表

**示例：**

**1,计算每季度最后一天的销量**

```
CALCULATE(
    SUM('订单表'[数量]),
    ENDOFQUARTER('日期表'[日期])
)
```

**2,计算每季度最后5天的销量**

```
CALCULATE(
    SUM('订单表'[数量]),
    DATESINPERIOD(
        '日期表'[日期],
        ENDOFQUARTER('日期表'[日期]),
        -5,
        DAY
    )
)
```

**说明：**

*如果当前上下文是年份，则返回该年度的最后一个季度的最后一天。*

# ENDOFYEAR

**注释：** 返回当前上下文年度的最后一天

**语法：**

ENDOFYEAR(日期列,[年度结束日期])

**参数：**

第一个参数：日期列，或者可以返回日期的表达式
第二个参数：可选，默认为12-31，主要用于财年计算

**返回：**

表，只有一个日期的表

**示例：**

**1,计算每年12月31日的销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    ENDOFYEAR('日期表'[日期])
)
```

**2,计算财年最后5天的销量，财年结束日期为9月30日**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESINPERIOD(
        '日期表'[日期],
        ENDOFYEAR('日期表'[日期],”9-30”),
        -5,
        DAY
    )
)
```

# STARTOFMONTH

**注释：** 返回当前上下文月份的第一天

**语法：**

```
STARTOFMONTH(日期列)
```

**参数：**

参数：日期列，或者可以返回日期的表达式

**返回：**

表，只有一个日期的表

**示例：**

**1,计算每月1号的销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    STARTOFMONTH('日期表'[日期])
)
```

**2,计算每月前5日的退货金额**

```
CALCULATE(
    [退货金额],
    DATESINPERIOD(
        '日期表'[日期],
        STARTOFMONTH('日期表'[日期]),
        5,
        DAY
    )
)
```

**说明：** *如果当前上下文是季度或者年，则返回该粒度的第一个月的1号*

# STARTOFQUARTER

**注释：** 返回当前上下文所在季度的第一天

**语法：**

```
STARTOFQUARTER(日期列)
```

**参数：**

参数：日期列，或者可以返回日期的表达式

**返回：**

表，只有一个日期的表

**示例：**

**1,计算每季度第一天的销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    STARTOFQUARTER('日期表'[日期])
)
```

**2,计算每季度前10日的退货金额**

```
CALCULATE(
    [退货金额],
    DATESINPERIOD(
        '日期表'[日期],
        STARTOFQUARTER('日期表'[日期]),
        10,
        DAY
    )
)
```

**说明：** *如果当前上下文是年，则返回该年度一季度的第一天*

# STARTOFYEAR

**注释：** 返回当前上下文年度的第一天

**语法：**

```
STARTOFYEAR(日期列,[年度结束日期])
```

**参数：**

第一个参数：日期列，或者可以返回日期的表达式
第二个参数：可选，默认为12-31，主要用于财年计算

**返回：**

表，只有一个日期的表

**示例：**

**1,计算每年1月1日的销量**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    STARTOFYEAR('日期表'[日期])
)
```

**2,计算财年的年度开始日前14天的退货金额，财年结束日期为9月30日**

```
CALCULATE(
    SUM(‘订单表’[数量]),
    DATESINPERIOD(
        '日期表'[日期],
        STARTOFYEAR('日期表'[日期],”9-30”),
        14,
        DAY
    )
)
```

# PARALLELPERIOD

**注释：** 返回移动指定间隔的完整粒度的时间段

**语法：**

```
PARALLELPERIOD（日期列,移动数量,移动粒度)
```

**参数：**

第一个参数：日期列，或者可以返回日期的表达式
第二个参数：整数，正数向前移动，负数向后回溯
第三个参数：粒度，year/quarter/month/day

**返回：** 单列日期值的表

**示例：** 计算去年全年的销售额

```
CALCULATE(
    [销售额],
    PARALLELPERIOD (
        '日期表'[日期],
        -1,
        year
    )
)
```

**说明：** 该函数与DATEADD含义类似，参数完全一样；
不同的是，DATEADD返回与当前上下文相同的时间段，而PARALLELPERIOD函数返回完整粒度的时间段。

# TOTALMTD

**注释：** 返回月初至今的表达式的值

**语法：**

```
TOTALMTD（表达式,日期列[,筛选条件])
```

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件

**返回：** 值

**示例：** 计算月初至今的销售额

```
TOTALMTD(
    [销售额],
    '日期表'[日期],
)
```

**说明：** 该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    [销售额],
    DATESMTD( '日期表'[日期])
)
```

# TOTALQTD

**注释：** 返回季初至今的表达式的值

**语法：**

```
TOTALQTD（表达式,日期列[,筛选条件])
```

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件

**返回：** 值

**示例：** 计算季初至今的销售额

```
TOTALQTD(
    [销售额],
    '日期表'[日期],
)
```

**说明：** 该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    [销售额],
    DATESQTD( '日期表'[日期])
)
```

# CLOSINGBALANCEMONTH

**注释：** 返回当前上下文月份最后一天的表达式的值

**语法：**

```
CLOSINGBALANCEMONTH（表达式,日期列[,筛选条件])
```

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件

**返回：** 值

**示例：** 计算月度最后一天的库存余额

```
CLOSINGBALANCEMONTH(
    SUMX(库存表,库存表[单位成本]*库存表[期末数量]),
    '日期表'[日期],
)
```

**说明：**

该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    SUMX(库存表,库存表[单位成本]*库存表[期末数量]),
    ENDOFMONTH( '日期表'[日期])
)
```

# CLOSINGBALANCEQUARTER

**注释：** 返回当前上下文季度最后一天的表达式的值

**语法：**

```
CLOSINGBALANCEQUARTER（表达式,日期列[,筛选条件])
```

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件

**返回：** 值

**示例：** 计算季度最后一天的库存余额

```
CLOSINGBALANCEQUARTER(
    SUMX(库存表,库存表[单位成本]*库存表[期末数量]),
    '日期表'[日期],
)
```

**说明：**

该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    SUMX(库存表,库存表[单位成本]*库存表[期末数量]),
    ENDOFQUARTER( '日期表'[日期])
)
```

# CLOSINGBALANCEYEAR

**注释：** 返回当前上下文年度最后一天的表达式的值

**语法：**

CLOSINGBALANCEYEAR（表达式,日期列[,筛选条件],年度结束日期) )

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件
第四个参数：可选，默认为12-31，主要用于财年计算

**返回：**

值

**示例：** 计算财年最后一天的库存余额，年度结束日期9月30日

```
CLOSINGBALANCEYEAR(
    SUMX(库存表,库存表[单位成本]*库存表[期末数量]),
    '日期表'[日期],,
    “9-30”
)
```

**说明：** 该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    SUMX(库存表,库存表[单位成本]*库存表[期末数量]),
    ENDOFYEAR( '日期表'[日期],”9-30”)
)
```

# OPENINGBALANCEMONTH

**注释：** 返回当前上下文月份第一天的表达式的值

**语法：**

OPENSINGBALANCEMONTH（表达式,日期列[,筛选条件])

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件

**返回：**

值

**示例：** 计算月度期初库存余额

```
OPENINGBALANCEMONTH(
    SUMX(库存表,库存表[单位成本]*库存表[期初数量]),
    '日期表'[日期],
)
```

**说明：**

该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    SUMX(库存表,库存表[单位成本]*库存表[数量]),
    STARTOFMONTH( '日期表'[日期])
)
```

# OPENINGBALANCEQUARTER

**注释：** 返回当前上下文季度第一天的表达式的值

**语法：**

OPENSINGBALANCEQUARTER（表达式,日期列[,筛选条件])

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件

**返回：**

值

**示例：** 计算季度期初库存余额

```
OPENINGBALANCEQUARTER(
    SUMX(库存表,库存表[单位成本]*库存表[期初数量]),
    '日期表'[日期],
)
```

**说明：**

该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    SUMX(库存表,库存表[单位成本]*库存表[数量]),
    STARTOFQUARTER( '日期表'[日期])
)
```

# OPENINGBALANCEYEAR

**注释：** 返回当前上下文年度第一天的表达式的值

**语法：**

OPENINGBALANCEYEAR（表达式,日期列[,筛选条件],年度结束日期) )

**参数：**

第一个参数：聚合表达式
第二个参数：日期列，或者可以返回日期的表达式
第三个参数：可选，筛选条件
第四个参数：可选，默认为12-31，主要用于财年计算

**返回：**

值

**示例：** 计算财年的期初库存金额，年度结束日期9月30日

```
OPENINGBALANCEYEAR(
    SUMX(库存表,库存表[单位成本]*库存表[期初数量]),
    '日期表'[日期],,
    “9-30”
)
```

**说明：** 该函数可以直接返回值，实际上内置了CALCULATE，上述示例相当于以下表达式：

```
CALCULATE(
    SUMX(库存表,库存表[单位成本]*库存表[期初数量]),
    STARTOFYEAR( '日期表'[日期],”9-30”)
)
```

# YEARFRAC

**注释：** 返回两个日期之间的天数占全年天数的比例

**语法：**

YEARFRAC (开始日期，结束日期，计算标准)

**参数：**

第一个参数：开始日期
第二个参数：结束日期
第三个参数：计算标准，(忽略时默认为0)
0-美国30/360
1-实际/实际
2-实际/360
3-实际/365
4-欧洲30/360

**返回：**

值

**示例：**

精确计算每个人的年龄?

```
YEARFRAC(
    [出生日期],
    TODAY()
)
```

**说明：**

该函数可以返回任意位数的小数，适用用精确计算。
注意与DATEDIFF的差异

# DATEVALUE

**注释：** 将文本格式的日期转换为日期格式的日期

**语法：**

DATEVALUE (文本日期)

**参数：**

只有一个参数：以文本格式保存的日期

**返回：**

值，日期格式的日期

**示例：**

**1，下述表达式将返回日期型的2019年2月16日**

```
DATEVALUE(“2019-2-16”)
```

**2，如果月/日/年、日/月/年格式不便于区分，返回结果跟随系统，如果当前系统设置为月/日/年，下述表达式返回2019年2月4日**

```
DATEVALUE(“2/4/2019”)
```

**如果当前系统设置为日/月/年，上述表达式返回2019年4月2日**

**3，如果年份缺省，自动返回当前系统所在的年份，下述表达式返回2019年2月16日**

```
DATEVALUE(“2/16”)
```

# DAY

**注释：** 返回日期的某一天，1-31之间的一个数字

**语法：**

```
DAY (日期)
```

**参数：**

只有一个参数：日期，日期格式或者文本格式都可以

**返回：**

值，1-31之间的一个数字

**示例：**

1，返回订单日期所属的日

```
DAY(订单表[订单日期])
```

2，假设每月18日为会员日，需要将订单表中为18日的订单都标记为“会员促销“，新建列

```
IF(DAY(订单表[订单日期])=18,”会员促销”,BLANK())
```

3，若参数为文本类型的日期，且文本不能区分哪个是月份时，会自动按照系统的日期格式返回结果。

假设系统设置是月/日/年，下述表达式返回8

```
DAY(“2/8”)
```

# EDATE

**注释：** 返回开始日期移动一定月数之后的日期

**语法：**

```
EDATE (开始日期，月数)
```

**参数：**

第一个参数：日期，日期格式或者文本格式都可以
第二个参数：移动月数，正数向未来移动，负数相反

**返回：** 值，单个日期

**示例：**

1，返回上个月的今天

```
EDATE(TODAY(),-1)
```

2，假设返回的日期超出该月份的最后一天，则返回最后一天，以下表达式返回2019年2月28日

```
EDATE(DATE(2019,1,31),1)
```

3，若参数为文本类型的日期，且文本不能区分哪个是月份时，会自动按照系统的日期格式返回结果。

假设系统设置是月/日/年，下述表达式返回2019年3月8日

```
EDATE(“2/8/2019”,1)
```

4，假设第二个参数不是整数，则四舍五入到最接近的整数，以下表达式返回2019年4月18日

```
EDATE(DATE(2019,2,18),1.5)
```

# EOMONTH

**注释：** 返回开始日期移动一定月数之后的该月最后一天的日期

**语法：**

EOMONTH (开始日期，月数)

**参数：**

第一个参数：日期，日期格式或者文本格式都可以
第二个参数：移动月数，正数向未来移动，负数相反

**返回：**

值，单个日期

**示例：**

1，返回上个月的最后一天

```
EOMONTH(TODAY(),-1)
```

2，假设返回的日期超出该月份的最后一天，则返回该月的最后一天，以下表达式返回2019年2月28日

```
EOMONTH(DATE(2019,1,31),1)
```

3，若参数为文本类型的日期，且文本不能区分哪个是月份时，会自动按照系统的日期格式返回结果。

假设系统设置是月/日/年，下述表达式返回2019年3月31日

```
EOMONTH(“2/8/2019”,1)
```

4，假设第二个参数不是整数，则四舍五入到最接近的整数，以下表达式返回2019年4月30日

```
EOMONTH(DATE(2019,2,19),1.5)
```

# WEEKDAY

**注释：** 返回星期几的数字形式

**语法：**

WEEKDAY (日期，类型)

**参数：**

第一个参数：日期，日期格式或者文本格式都可以

第二个参数：类型：

**1：星期日（1）-到星期六（7）**

**2：星期一（1）-到星期日（7）**

**3：星期日（0）-到星期六（6）**

**返回：**

值，1到7、或者0到6的整数

**示例：**

**1，返回今天是星期几**

```
WEEKDAY(TODAY(),2)
```

**2，为日期表添加周几的字段**

```
WEEKDAY ( [Date],2 )
```

# WEEKNUM

**注释：** 返回日期在一年中所属的周数

**语法：**

WEEKNUM (日期，类型)

**参数：**

第一个参数：日期，日期格式或者文本格式都可以
第二个参数：类型：

**1：星期日为一周的第一天**
**2：星期一为一周的第一天**

**返回：**

值，整数

**示例：**

**1，计算今天是本年的第几周**

```
WEEKNUM(TODAY(),2)
```

**2，为日期表添加周数的字段**

```
WEEKNUM ( [Date],2 )
```

# EARLIER

**注释：** 返回当前行中参数列的值

**语法：**

EARLIER (列)

**参数：**

参数：当前表的列

**返回：**

值，当前行和参数列列交叉的单元格的值

**示例：** 计算当前订单产品的目前累计销量

```
SUMX(
    FILTER(
        '订单表’,
        '订单表'[序号]<=EARLIER('订单表'[序号])
        &&‘订单表’[产品名称]=EARLIER(‘订单表’[产品名称])
    ),
    ‘订单表’[销售数量]
)
```

**说明：**

EARLIER函数主要用于计算列，获取当前行上下文。
因为它需要扫描每一行，计算性能相对较慢

# ERROR

**注释：** 当计算发生错误时，弹出错误提示窗口

**语法：** ERROR (文本信息)

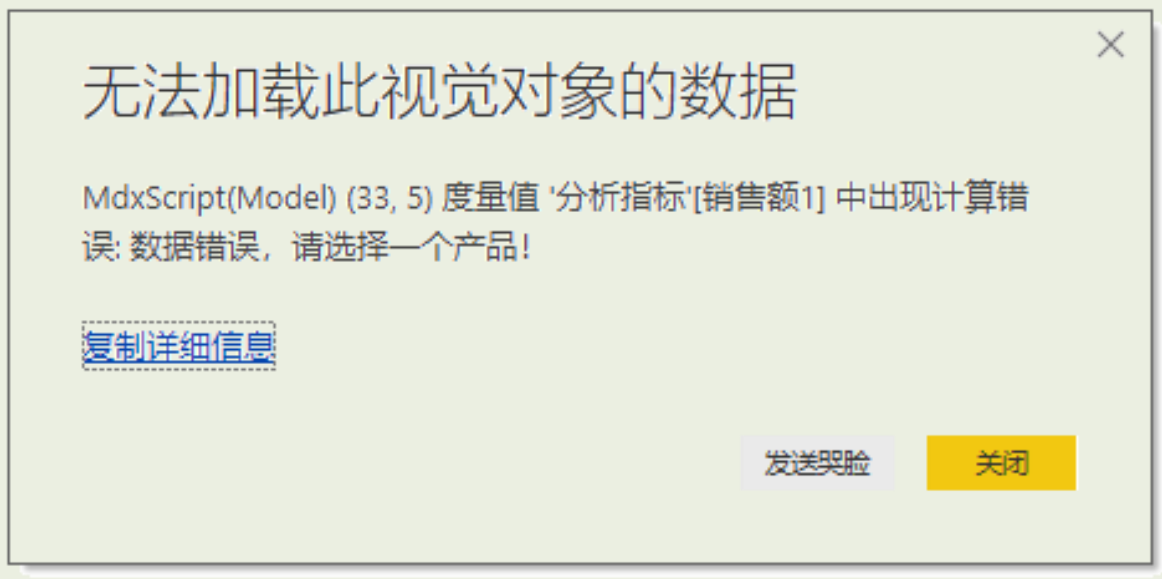
**参数：** 参数：提示错误的文本字符串

**返回：** 无

**示例：** 用于主动拦截错误，并弹出提示信息

```
IF(HASONEFILTER('产品'[产品名称]),
    [销售额],
    ERROR("数据错误，请选择一个产品！")
)
```

上述表达式的意思是，如果上下文选择了一个产品，则计算这个产品的销售额，如果没有选择产品或者选择了多个产品，将出现错误提示，提示信息为：数据错误，请选择一个产品!



# CONTAINS

**注释：** 如果在列中包含引用的值，返回true，否则，返回false

**语法：**

CONTAINS (表,表[列],值,……)

**参数：**

第一个参数：表
第二个参数：表中的列，查找范围
第三个参数：查找的值
可以有第四个参数、第五个参数...，类型和功能同第二和第三参数，必须成对出现

**返回：**

布尔值，true/false

**示例：** 以下表达式，告诉你2019年2月1日，客户张三是否购买过

```
CONTAINS(
    订单表,
    订单表[订单日期], DATE(2019,2,1),
    订单表[客户名称],"张三"
)
```

# NATURALLEFTOUTERJOIN

**注释：** 左外联结，返回左表中的所有行及右表的匹配行

**语法：**

NATURALLEFTOUTERJOIN (左表，右表)

**参数：**

第一个参数：表或者表表达式，左表
第二个参数：表或者表表达式，右表

**返回：**

表，包含左表中的所有行和右表中的匹配行

**示例：**

以下表达式，返回产品A所有的客户列表以及产品B的匹配客户,只购买产品A没有购买产品B的，在产品B客户列中显示为空值。

```
VAR customerA=
    SUMMARIZE(
        FILTER( '订单', ‘订单’[产品名称]=“产品A" ),
        '订单'[客户姓名]
    )
VAR customerB=
    SUMMARIZE(
        FILTER( ‘订单’, ‘订单’[产品名称]=“产品B” ),
        '订单'[客户姓名]
    )
RETURN
    NATURALLEFTOUTERJOIN(customerA, customerB))
```

# COUNT

**注释：** 统计包含数值/文本的列中的单元格数

**语法：** COUNT (列)

**参数：** 只有一个参数：需要统计的列

**返回：** 值

**示例：** 统计订单数量

```
COUNT('订单表'[订单编号])
```

统计订单数量也可以直接使用COUNTROWS(订单表)

COUNT的参数是列， COUNTROWS的参数是表

**说明：** 统计该列所有的单元格，包括空值;

参数列必须包含有数据、日期或者字符串数据，否者返回错误，
如：某一列只有布尔值，则返回错误

# FIND

**注释：** 返回一个文本在另一个文本中的起始位置

**语法：**

```
FIND (查找内容，被查找内容，[开始位置],
        [找不到时返回的值])
```

**参数：**

第一个参数：要查找的字符;
第二个参数：被查找的内容;
第三个参数：可选，开始搜索的位置，默认为1;
第四个参数：可选，查找不到时返回的值，通常为0，-1或者BLANK()

**返回：**

值

**示例：** 查找产品名称中是否包含”智能”

```
1 是否智能 = IF(FIND("智能",[产品名称],,0)>0,"Y","N")
```

| 产品类别 | 产品名称 | 产品代码 | 零售价 | 采购价 | 单位利润 | 是否智能 |
|---|---|---|---|---|---|---|
| 智能设备 | VR眼睛 | zn013 | 899 | 460 | 439 | N |
| 智能设备 | 智能手环 | zn012 | 249 | 188 | 61 | Y |
| 智能设备 | 智能手表 | zn011 | 199 | 148 | 51 | Y |
| 手机配件 | 数据线 | sj005 | 19 | 12 | 7 | N |
| 手机配件 | 贴膜 | sj004 | 29 | 10 | 19 | N |
| 手机配件 | 充电宝 | sj003 | 119 | 66 | 53 | N |
| 手机配件 | 耳机 | sj002 | 109 | 58 | 51 | N |
| 手机配件 | 手机壳 | sj001 | 26 | 12 | 14 | N |

**说明：**

FIND函数区分大小写，而SEARCH函数不区分大小写
FIND函数不支持通配符，SEARCH函数支持

# EXCEPT

**注释：** 返回一个表中未出现在另一个表中的行。

**语法：**

EXCEPT(表1，表2)

**参数：**

第一个参数：表，可以是返回表的表达式
第二个参数：表，可以是返回表的表达式
从第一个参数表中，找出未出现在第二个参数表中的行

**返回：**

表

**示例：** 计算购买A未购买B的客户数量

```
VAR customerA=
    SUMMARIZE(
        FILTER( ‘订单’, ‘订单’[产品名称]=产品A" ),
        '订单'[客户姓名]
    )
VAR customerB=
    SUMMARIZE(
        FILTER( ‘订单’, ‘订单’[产品名称]=“产品B" ),
        '订单'[客户姓名]
    )
RETURN
    COUNTROWS(EXCEPT(customerA,customerB))
```

**说明：**

两个表必须具有相同的列数
两个参数表是有先后顺序的，表的顺序不同，返回的结果也不同

# GENERATESERIES

**注释：** 生成一系列值的列表

**语法：**

GENERATESERIES(开始值，结束值，[增量值])

**参数：**

第一个参数：生成序列的初始值
第二个参数：生成序列的结束值
第三个参数：可选，递增值，忽略时默认为1

**返回：**

表，单列值的表，列名为Value

**示例：** 生成0到100的偶数序列

```
GENERATESERIES(0,100,2)
```

**说明：** 该函数常用于创建参数，进行动态分析

当第一个参数大于第二个参数时，返回空表；
增量值必须是正数；
序列的最后一个值小于或者等于结束值；

# DATATABLE

**注释：** 创建表并声明每个字段的类型

**语法：**

```
DATATABLE(列名1，类型，列名2，类型，……
                {{第一行数据}，{第二行数据}})
```

**参数：**

数据类型包括：INTEGER，DOUBLE，STRING，BOOLEAN，CURRENCY，DATETIME；

每一行的数据以逗号分隔；

缺失值视同BLANK().

**返回：** 表

**示例：** 以下表达式生成一个表

```
客户表 =
DATATABLE(
    "姓名",STRING,"年龄",INTEGER,"生日",DATETIME,
    {
        {"张三",25,"1994-02-13"},
        {"李四",35,"1984-08-25"}
    }
)
```

返回：

| 姓名 | 年龄 | 生日 |
|---|---|---|
| 张三 | 25 | 2/13/1994 |
| 李四 | 35 | 8/25/1984 |

# ISBLANK

**注释：** 检查值是否为空，并返回TRUE或FALSE

**语法：** ISBLANK(value)

**参数：** 需要检查的值或者表达式

**返回：** 值，TRUE或FALSE

**示例：** 计算本年累计销售额的同比增长率，检查上年销售额是否为空，如果为空，就不再计算增长率。

(实际上可以直接使用DIVIDE来相除,主要是理解思路)

```
IF(
    ISBLANK(
        CALCULATE(
            [销售额],
            SAMEPERIODLASTYEAR(DATESYTD(日期表[日期]))
        )
    ),
    BLANK(),
    CALCULATE([销售额],DATESYTD(日期表[日期]))
     / CALCULATE(
         [销售额],
         SAMEPERIODLASTYEAR(DATESYTD(日期表[日期]))
     ) -1
)
```

# TIME

**注释：** 将以数字形式给出的小时、分钟和秒转换为时间格式的时间

**语法：**

TIME(hour，minute，second)

**参数：**

hour:0到23数字，大于23的值将除以24，余数作为小时；

minute:0到59的数字，大于59的将自动进位成hour;

second: 0到59的数字，大于59的将自动进位成minute

**返回：**

值，时间格式的值

**示例：**

以下表达式均返回凌晨4点

```
TIME(4,0,0)
TIME(28,0,0)
TIME(0,240,0)
TIME(0,0,14400)
```

计算一天的第25000秒是什么时间

```
TIME(0,0,25000)//返回 6:56:40 AM
```

# HOUR

**注释：** 将时间值中的小时，返回0到23的数字

**语法：** HOUR(日期时间)

**参数：** 日期时间格式，也可以是规范的文本格式

**返回：** 值，0-23的整数

**示例：** 返回订单表每一笔订单时间的小时

```
HOUR(订单表[订单时间])
```

返回文本格式时间中的小时数

```
HOUR("March 15, 2019 4:00 PM")//返回 16
```

# MINUTE

**注释：** 将时间值中的分钟，返回0到59的数字

**语法：**

```
MINUTE(日期时间)
```

**参数：**

日期时间格式，也可以是规范的文本格式

**返回：**

值，0-59的整数

**示例：**

返回订单表每一笔订单时间的分钟

```
MINUTE(订单表[订单时间])
```

返回文本格式时间的分钟数

```
MINUTE("March 16, 2019 4:24 PM")//返回 24
```

# SECOND

**注释：** 将时间值中的秒，返回0到59的数字

**语法：**

```
SECOND(日期时间)
```

**参数：**

日期时间格式，也可以是规范的文本格式

**返回：**

值，0-59的整数

**示例：**

返回订单表每一笔订单时间的秒数

```
SECOND(订单表[订单时间])
```

返回文本格式时间的秒数

```
SECOND("March 17, 2019 4:24:56 PM")//返回 56
```

# CEILING

**注释：** 将数字向上舍入到最接近的基数的倍数

**语法：**

CEILING(value，基数)

**参数：**

第一个参数：需要舍入的数值
第二个参数：基数，按照该基数的整数倍向上舍入。

**返回：**

值

**示例：**

```
CEILING(3.51,0.05)  //返回3.55
```

# ISCROSSFILTERED

**注释：** 参数列被交叉筛选时返回TRUE，否则False

**语法：**

```
ISCROSSFILTERED([列] )
```

**参数：**

参数：列，不能是表达式

**返回：**

值，True/False

**示例：**

外部上下文为日期，如何获得每个时间段在售的产品表中产品数量？

下述表达式的含义是，如果产品表与日期表存在交叉筛选关系，正常计算，否则返回空值

```
在售的产品数 =
IF(
    ISCROSSFILTERED(‘产品表'[产品名称]),
     COUNTROWS(产品表)
)
```

# COMBINEVALUES

**注释：** 将多个字符串连结为一个字符串

**语法：**

COMBINEVALUES(分隔符，表达式1，表达式2……)

**参数：**

第一个参数：分隔符，必须是一个恒定的值

第二个及之后的参数：可以是一个列，也可以是表达式

**返回：**

值，连接后的字符串

**示例：** 将年度和月份连在一起生成年月字段

**COMBINEVALUES**(“-”,日期表[年度],日期表[月份])

# EXACT

**注释：** 比较两个文本字符串，如果完全相同返回True，否则False

**语法：**

```
EXACT(文本1，文本2)
```

**参数：**

参数：文本字符串

**返回：**

值，True / False

**示例：** 检查列1和列2是否完全相同

```
EXACT([列1],[列2])
```

**说明：** EXACT函数区分大小写，但忽略格式差异

# FIXED

**注释：** 将数字舍入到指定的小数位数并将结果作为文本返回

**语法：**

```
FIXED(数字，[小数位数]，[逻辑值])
```

**参数：**

第一个参数：数字或者数字列；

第二个参数：小数点右边的位数，忽略时默认为2；

第三个参数：如果是0或者忽略，则在返回的文本中显示逗号，如果为1，则不显示逗号。

**返回：** 值,文本格式

**示例：**

```
FIXED(12345.678,4) //返回：12,345.6780
```

**说明：**

第二个参数是负数时，将向左舍入；
FIXED函数与格式菜单中设置格式的区别是，格式菜单设置后仍然为数值型，而FIXED函数返回的是文本型数据。

# VALUE

**注释：** 将表示数字的文本字符串转换为数字

**语法：**

```
VALUE(文本)
```

**参数：**

参数：需要转换的文本

**返回：**

值,数值型

**示例：**

```
VALUE(“12,345.678”) //返回：12345.678
```

**说明：**

文本应是可转换为数值的数字、日期或者常量，如果文本不是这些格式，否则将返回错误；
注意该函数与VALUES函数完全不同。

# ROW

**注释：** 返回一个单行的表

**语法：** ROW(字段名，表达式，字段名，表达式，……)

**参数：**
第一个参数:字段名，应使用""括起来;
第二个参数:表达式,应为返回单个值的表达式;
后面的参数同前两个参数，总是要成对出现。

**返回：** 表,只有一行的表

**示例：** 以下表达式返回一个单行表：订单数量和订单总金额

```
ROW(
    "订单数量"，COUNTROW(订单表)，
    "订单销售总额"，SUM(订单表[销售额])
)
```

# CONTAINSROW

**注释：** 判断表中是否含有一行值，如果是则返回True

**语法：**

```
CONTAINSROW (查找值的列表，被查找的字段)
```

**参数：**

第一个参数：查找列表，一般用大括号括起来；

第二个参数：被查找的字段，如果查找的列表的每一行有多个值，则被查找的也对应多个字段。

**返回：** 值，True/False

**示例：**

1，查找产品表中耳机和硬盘的产品信息

```
FILTER(
    '产品',
    CONTAINSROW({"耳机","硬盘"},[产品名称])
)
```

2，查找产品表中采购价为58的耳机的产品信息

```
FILTER(
    '产品',
    CONTAINSROW({(“耳机”,58)},[产品名称],[采购价])
)
```

3，查找产品表中除了耳机和硬盘的其他产品信息

```
FILTER(
    '产品',
    NOT CONTAINSROW({"耳机","硬盘"},[产品名称])
)
```

# FACT

**注释：** 计算阶乘

**语法：**

```
FACT (数字)
```

**参数：** 参数应为非负数,可以是表达式

**返回：** 值

**示例：** 新建列，计算Value列每一个数字的阶乘

```
FACT([Value])
```

**说明：**

如果参数为非整数，则截取整数部分作为参数；
由于阶乘数据的增长十分迅速，如果参数太大，会返回错误



# GCD

**注释：** 计算两个或多个整数的最大公约数

**语法：**

```
GCD (数字1,数字2，……)
```

**参数：**

至少应有一个参数，其他参数可选

**返回：**

值

**示例：** 计算36与5的阶乘的最大公约数

```
GCD(36,FACT(5)) //返回12
```

**说明：** 如果参数为负数或者为非数字，将返回错误



# IFERROR

**注释：** 用来捕获和处理表达式中的错误

**语法：**

IFERROR (正确时返回的结果，如果错误时返回的结果)

**参数：**

第一个参数是用于判断的表达式，如果没有错误，就返回这个表达式的结果；否则返回第二个参数的结果；两个参数均可以是表达式

**返回：**

值,类型取决于参数表达式计算的结果

**示例：** 计算本年销售额除以上年销售额，如果上年销售额为0，则返回9999.

```
IFERROR([本年销售额]/[上年销售额],9999)
```

**说明：** 两个参数的数据类型必须一致，否则返回错误

# ISERROR

**注释：** 检查计算结果是否错误

**语法：**

```
ISERROR(value)
```

**参数：**

需要检查的值或者表达式

**返回：**

值，TRUE或FALSE

**示例：**

计算本年累计销售额的同比增长率，检查上年销售额是否为0，如果为0，返回空。

(实际上可以直接使用DIVIDE来相除,主要是理解思路)

```
IF(
    ISERROR([本年销售额]/[上年销售额]),
    BLANK(),
    [本年销售额]/[上年销售额]
)
```

**说明：**

上述表达式也可使用IFERROR来计算，如下：

```
IFERROR([本年销售额]/[上年销售额],BLANK())
```

# ISEMPTY

**注释：** 检查表是否为空

**语法：**

ISEMPTY (表)

**参数：**

只有一个参数，可以是返回表的表达式

**返回：**

值,True/False

**示例：** 检查订单表中是否有“U盘”的订单

```
ISEMPTY (
    CALCULATETABLE(
        订单表，订单表[产品名称]=“U盘”
    )
)
```

**说明：** 上述表达式还可以用ISBLANK和COUNTROWS组合来替代，如下：

```
ISBLANK (
   COUNTROWS(
       CALCULATETABLE(
           订单表，订单表[产品名称]=“U盘”
       )
   )
)
```

不过ISEMPTY使用起来更简单且性能更优，应直接使用ISEMPTY

# LCM

**注释：** 计算两个或多个整数的最小公倍数

**语法：**

```
LCM(数字1,数字2，……)
```

**参数：**

至少应有一个参数，其他参数可选

**返回：**

值

**示例：** 计算36与4的阶乘的最小公倍数

```
LCM(36,FACT(4)) //返回72
```

**说明：** 如果参数为负数或者为非数字，将返回错误



# MROUND

**注释：** 返回舍入到所需倍数的数字

**语法：**

```
MROUND(需要舍入的数字,倍数)
```

**参数：**

两个参数应该同时为正，或者同时为负

**返回：**

值

**示例：** 下述表达式返回1.4，最接近的0.2的倍数

```
MROUND(1.3,0.2)
```

**说明：** 如果参数分别为整数和负数，将返回错误



# POWER

**注释：** 返回数值的N次方

**语法：**

```
POWER(数值,幂数)
```

**参数：** 参数可以是任何实数

**返回：** 值

**示例：** 下述表达式返回1024

```
POWER(2,10)
```

**说明：** 第二个参数是分数时，可以计算N次方根



# PRODUCT

**注释：** 返回列中数字的乘积

**语法：**

```
PRODUCT(列)
```

**参数：**

表的某一列

**返回：**

值

**示例：** 计算每年的增长率的乘积

```
PRODUCT(表[增长率])
```

**说明：** 仅计算列中的数字。空格，逻辑值和文本将被忽略



# PRODUCTX

**注释：** 返回表中的每一行计算的表达式的乘积。

**语法：**

```
PRODUCTX(表，表达式)
```

**参数：**

第一个参数：计算表达式的表，可以是表达式
第二个参数：每一行执行计算的表达式

**返回：**

值

**示例：**

假设投资表中含有每年以及当年的收益率数据，则以下表达式返回累计的收益率

```
PRODUCTX(投资表，1+[收益率])
```

**说明：**

仅计算列中的数字。空格，逻辑值和文本将被忽略

# QUOTIENT

**注释：** 执行除法并仅返回除法结果的整数部分

**语法：**

```
QUOTIENT(分子，分母)
```

**参数：**

分子和分母都应为数字或者数值型字段

**返回：**

值，整数

**示例：**

```
QUOTIENT(7,2)   //返回3
```

**说明：**

参数非数字时，返回错误;
分母为0时，返回错误;
分母为字段时，该字段中含有一个0值，则整列返回错误。

# RANK.EQ

**注释：** 返回数字列中数字的排名，如果多个值具有相同的排名，则返回该组值的最高排名。

**语法：**

RANK.EQ(值，列，[order])

**参数：**

第一个参数：值，需要排名的值；
第二个参数：对值进行排名的列，不能是表达式；
第三个参数：可选，排序方式，忽略时默认降序，
0 / FALSE / DESC - 下降，　1 / TRUE / ASC - 升序

**返回：**

值

**示例：** 对产品的单位利润进行排序

```
RANK.EQ(产品表[单位利润],产品表[单位利润])
```



# MEDIAN

**注释：** 返回一列数字的中位数

**语法：**

```
MEDIAN(列)
```

**参数：**

数值型的列

**返回：**

值

**示例：** 返回客户年龄的中位数

```
MEDIAN(客户表[年龄])
```

**说明：** 仅计算列中的数字。空格，逻辑值和文本将被忽略



# MEDIANX

**注释：** 返回为表中的每一行计算的表达式的中位数

**语法：**

```
MEDIANX(表，表达式)
```

**参数：**

第一个参数：计算表达式的表，可以是表达式
第二个参数：每一行执行计算的表达式

**返回：** 值

**示例：** 计算北京客户的年龄中位数

```
MEDIANX(
    FILTER(客户表，客户表[城市]=“北京”),
    客户表[年龄]
)
```

**说明：** 仅计算列中的数字。逻辑值和文本将被忽略,空值不会忽略

# SIGN

**注释：** 返回数字的符号，正数返回1，零返回0，负数返回-1，

**语法：**

```
SIGN(数值)
```

**参数：**

参数为数值或者返回数值的表达式

**返回：**

值， 1 / 0 / -1

**示例：** 返回销售价格减去成本的符号，以确定是否盈利

```
SIGN([销售价格]-[销售成本])
```



# UNICODE

**注释：** 返回文本中第一个字符的UNICODE代码

**语法：**

```
UNICODE(文本)
```

**参数：**

参数可以任意文本或者字段

**返回：**

值

**示例：** 判断姓名列的第一个字符是否为英文

```
IF(UNICODE([姓名])<123,”YES”,”NO”)
```

**说明：** 当参数为多个字符时，只返回第一个字符的UNICODE

# ISFILTERED

**注释：** 指定的列上有过滤器返回TRUE，否则False

**语法：** ISFILTERED(表或者列)

**参数：** 参数：用于检查是否有过滤器的表或列

**返回：** 值，True/False

**示例：** 以下两个表达式均返回True：

```
CALCULATE(
    ISFILTERED ( 订单表 ),
    订单表[销售单价] > 50 )

CALCULATE(
    ISFILTERED ( 订单表[销售单价] ),
    订单表[销售单价]  > 50 )
```

以下两个表达式均返回False：

```
CALCULATE(
    ISFILTERED ( 订单表 ),
    产品表[产品名称] =“硬盘” )

CALCULATE(
    ISFILTERED ( 订单表[销售单价] ),
    订单表[销售数量] > 1 )
```

# PATH

**注释：** 返回一个父子层级的字符串，以”|”分隔

**语法：** PATH(子列，父列)

**参数：** 参数：必须是现有列的名称，不能是表达式

**返回：** 值，分隔符连成的字符串

**示例：**

```
1  地理层级 = PATH('地理信息表'[地点],'地理信息表'[上级地点])
```

| 地点 | 上级地点 | 地理层级 |
|---|---|---|
| 海淀区 | 北京 | 亚洲\|中国\|北京\|海淀区 |
| 东城区 | 北京 | 亚洲\|中国\|北京\|东城区 |
| 北京 | 中国 | 亚洲\|中国\|北京 |
| 上海 | 中国 | 亚洲\|中国\|上海 |
| 中国 | 亚洲 | 亚洲\|中国 |
| 亚洲 | | 亚洲 |

**说明：**

父子列必须具有相同的数据类型，文本或者数值
父列中的数据必须存在于子列中

关于父子函数的应用请参考：PowerBI父子函数应用

# PATHCONTAINS

**注释：** 如果指定的项存在于指定的路径中，则返回TRUE

**语法：**

```
PATHCONTAINS(路径，查找项目)
```

**参数：**

路径：PATH函数返回的层次路径
查找项目：文本

**返回：**

值，True/False

**示例：**

```
1  包含北京 = PATHCONTAINS([地理层级],"北京")
```

| 地点 | 上级地点 | 地理层级 | 包含北京 |
|---|---|---|---|
| 海淀区 | 北京 | 亚洲\|中国\|北京\|海淀区 | True |
| 东城区 | 北京 | 亚洲\|中国\|北京\|东城区 | True |
| 北京 | 中国 | 亚洲\|中国\|北京 | True |
| 上海 | 中国 | 亚洲\|中国\|上海 | False |
| 中国 | 亚洲 | 亚洲\|中国 | False |
| 亚洲 | | 亚洲 | False |

**说明：** 如果查找项目为数值，会自动转换为文本进行查找

# PATHITEM

**注释：** 返回PATH函数得到的字符串指定层级的项目，层级按从左向右计数

**语法：**

```
PATHITEM(路径，查找层级，[类型])
```

**参数：**

路径：PATH函数返回的层次路径
查找层级：数值
类型：可选,默认为0,返回文本结果;为1时返回整数

**返回：**

值

**示例：**

```
1 第三层 =
2 PATHITEM(PATH('地理信息表'[地点],'地理信息表'[上级地点]),3)
```

| 地点 | 上级地点 | 地理层级 | 第三层 |
|---|---|---|---|
| 海淀区 | 北京 | 亚洲\|中国\|北京\|海淀区 | 北京 |
| 东城区 | 北京 | 亚洲\|中国\|北京\|东城区 | 北京 |
| 北京 | 中国 | 亚洲\|中国\|北京 | 北京 |
| 上海 | 中国 | 亚洲\|中国\|上海 | 上海 |
| 中国 | 亚洲 | 亚洲\|中国 | |
| 亚洲 | | 亚洲 | |

**说明：**

如果层级参数小于1或者大于PATH返回的层级，返回BLANK
如果返回类型不是有效的结果类型，则返回错误

# PATHITEMREVERSE

**注释：** 返回PATH函数得到的字符串指定层级的项目，层级按从右向左计数

**语法：**

PATHITEMREVERSE(路径，查找层级，[类型])

**参数：**

路径：PATH函数返回的层次路径
查找层级：数值
类型：可选,默认为0,返回文本结果;为1时返回整数

**返回：**

值

**示例：**

```
1  第三层 =
2  PATHITEMREVERSE(PATH('地理信息表'[地点],'地理信息表'[上级地点]),3)
```

| 地点 | 上级地点 | 地理层级 | 第三层 |
|---|---|---|---|
| 海淀区 | 北京 | 亚洲\|中国\|北京\|海淀区 | 中国 |
| 东城区 | 北京 | 亚洲\|中国\|北京\|东城区 | 中国 |
| 北京 | 中国 | 亚洲\|中国\|北京 | 亚洲 |
| 上海 | 中国 | 亚洲\|中国\|上海 | 亚洲 |
| 中国 | 亚洲 | 亚洲\|中国 | |
| 亚洲 | | 亚洲 | |

**说明：**

如果层级参数小于1或者大于PATH返回的层级，返回BLANK
如果返回类型不是有效的结果类型，则返回错误

# PATHLENGTH

**注释：** 返回PATH函数得到的字符串的层级数

**语法：**

```
PATHLENGTH(路径)
```

**参数：**

路径：PATH函数返回的层次路径

**返回：**

值

**示例：**

```
1  PATH长度 =
2  PATHLENGTH(PATH('地理信息表'[地点],'地理信息表'[上级地点]))
```

| 地点 | 上级地点 | 地理层级 | PATH长度 |
|---|---|---|---|
| 海淀区 | 北京 | 亚洲\|中国\|北京\|海淀区 | 4 |
| 东城区 | 北京 | 亚洲\|中国\|北京\|东城区 | 4 |
| 北京 | 中国 | 亚洲\|中国\|北京 | 3 |
| 上海 | 中国 | 亚洲\|中国\|上海 | 3 |
| 中国 | 亚洲 | 亚洲\|中国 | 2 |
| 亚洲 | | 亚洲 | 1 |

PowerBI星球

Data
Analysis
Expressions

# PowerBI星球学习社群

采悟的知识星球，**6000+** PowerBI学习者已加入

加入即可解锁会员专属权益：

**零基础轻松上手视频课程**（可永久回看）
从0到1，帮你快速入门

**DAX实践系列视频课程**（可永久回看）
掌握常用的PowerBI数据分析套路

**财务报表分析视频课程**（可永久回看）
从财务报表数据到美观的可视化作品

**销售业务分析视频课程**（可永久回看）
帮你快速拥有自己的可视化作品

**公众号文章源文件**
已分享600+ 原创文章案例

**365天不限次的提问**
遇到问题随时提问

**会员专属微信群**
日常PowerBI问题，随时和大家交流

**前期沉淀的所有精华内容**
历史已发布的内容都可永久查看

以及，未来一年的更多资源分享.....

**成为会员，开启加速学习之旅**

微信扫码立即加入

PowerBI星球

Data
Analysis
Expressions

关注"PowerBI星球"

帮你轻松上手Power BI