# DAX语言各种：运算符

2020年5月3日 22:18

| 算术运算符 | 含义 | 示例 |
| --- | --- | --- |
| + | 加 | 2+1=3 |
| - | 减 | 3-1=2 |
| * | 乘 | 3*3=9 |
| / | 除 | 3/3=1 |
| ^ | 幂 | 2^3=8 |

| 比较运算符 | 含义 |
| --- | --- |
| = | 等于 |
| > | 大于 |
| < | 小于 |
| >= | 大于或等于 |
| <= | 小于或等于 |
| <> | 不等于 |

| 文本运算符 | 含义 |
| --- | --- |
| & | 连接 |

| 逻辑运算符 | 含义 |
| --- | --- |
| && | 相当于and |
| \|\| | 相当于or |
| in | 包含 |
| not in | 不包含 |

# 01.度量值

2020年6月27日 11:25

**一、将度量值全部放在一个表中**

**二、度量值举例**

**总销量 = SUM('销售表'[销售数量])**

**平均销量 = AVERAGE('销售表'[销售数量])**

**最大值 = Max('销售表'[销售数量])**

**最小值 = Min('销售表'[销售数量])**

**订单数量 = Countrows('销售表')**

**门店数量 = DISTINCTCOUNT('销售表'[店号])**

**单店销售量 = [总销量]/[门店数量]**

**CountA函数：计算列中单元格不为空的数目**

**Countblank函数：计算列中单元格为空白数量**

**将字段区隐藏再显示，不仅表格图标发生变化，而且度量值集群表会自动排列在最上方，方便阅读**

# 02.新建列与关系函数

2020年5月5日 1:33

Related函数：（多端找一端）（事实表找维度表）（数据表找基础表）

销售金额 = [销售数量]*RELATED('商品表'[售价])

毛利额 = [销售金额]-([销售数量]*RELATED('商品表'[进价] ))

度量值销售金额 = SUMX('销售表','销售表'[销售数量]*RELATED('商品表'[售价]))

Relatedtable函数：（一端找多端）（维度表找事实表）（基础表找数据表）

例如：在商品表中，添加一列订单数量。【商品表是一端，再多端的销售表里找】

COUNTROWS（表） 统计表格中的行数

订单数量 = COUNTROWS(RELATEDTABLE('销售表'))

Lookupvalue函数：就是Vlookup

语法：Lookupvalue(引用哪张表哪个列，那张表的搜索范围，找自己表里的哪个列)

新建列售价 = LOOKUPVALUE('商品表'[售价],'商品表'[商品编码],'销售表'[商品编码])

如果两个表都有如下字段，不用合并后去查找，使用多条件即可

可以是多个条件，例如：

条件1：'商品表'[商品种类],'销售表'[商品种类]

条件2：'商品表'[杯型],'销售表'[杯型]

# 03.最强大的引擎与筛选表

2020年6月26日 22:41


多条件时，建议使用in或not in

```
度量值 = Calculate([销售总量],'商品表'[商品种类] in {"蔬菜","水果","水产"})
度量值 = Calculate([销售总量], not '商品表'[商品种类] in {"蔬菜","水果","水产"})
```

Calculatetable(表，筛选条件)

与Calculate的区别就是，它可以多表运作筛选，返回一张表

例如：

```
Calculatetable('销售表','商品表'[商品种类]="水果",'商品表'[规格]="盒")
```

# 04. 高级筛选器Filter

2020年5月4日 3:39

**返回一个表，用于表示另一个表或表达式的子集，不能单独使用**
**Filter函数对筛选的表进行横向的逐行扫描，这样的函数叫迭代函数。**

**Countrows(Filter(表，筛选条件))**

**Calculate(表达式(度量值),Filter('表名', 筛选条件))**

**一、效果相同，为什么使用Filter函数?**
**销售数量 = SUM('销售明细表'[销量数量] )**
**销量a = CALCULATE([销售数量],FILTER('商品表','商品表'[类别]="蔬菜" && '商品表'[规格]="盒"))**
**销量b = CALCULATE([销售数量],'商品表'[类别]="蔬菜",'商品表'[规格]="盒")**

**二、什么时候使用Filter函数**
**在Calculate函数中的直接筛选条件里，我们只能输入:**
**‘表’ [列] = 固定值 或 ‘表’ [列] <> 固定值**
**‘表’ [列] >= 固定值 或 ‘表’ [列] <= 固定值**
**‘表’ [列] > 固定值 或 ‘表’ [列] < 固定值**
**但是遇到如下情况，就要使用Filter函数**
**[列]=[度量值]、 [列]=公式、[列]=[列]**
**[度量值]=[度量值]、 [度量值]=公式、[度量值]=固定值**

**例如：盒装苹果销量200盒以上的门店总共卖了多少盒?**
**Filter销售量 = Calculate([销售量],Filter('店铺表',[销售量]>200))**

| 商品种类 | 年份季度 | 销售量 | Filter销售量 |
|---|---|---|---|
| 西红柿(盒) | 2019Q1 | 1856 | 1485 |
| 黄瓜(盒) | 2019Q2 | 5693 | 4554 |
| 苹果(盒) | 2019Q3 | 9981 | 7985 |
| 梨(盒) | 2019Q4 | 15691 | 12553 |
| | 总计 | 33221 | 26577 |

**注意：这里必须是度量值大于200**
**不能是Sum('销售表'[销售数量])>200**

**三、注意事项**
1. **尽量在基础表中使用，不要在数据表中使用**
2. **能使用Calculate函数直接完成，就不要使用Filter函数**

# 05.人生处处是SQL

2020年5月6日 15:55

**语法：SummarizeColumns（第1参数，第2参数，第3参数，第4参数)**

| 位置 | 参数 | 描述 |
| --- | --- | --- |
| 可重复第1参数 | GroupBy_ColumnName | 分组依据，可重复。可用于小计和总计函数 |
| 可选重复第2参数 | FilterTable | 可对原表进行筛选 |
| 可选第重复3参数 | Name | 新增加的列名 |
| 可选重复第4参数 | Expression | 新增加的列的内容表达式 |

表名[字段名]

**一、返回不重复姓名**

**不重复姓名 = SummarizeColumns('重复姓名'[姓名])**

表名1[字段名1],表名2[字段名2]　注：可以不是同一张表

**二、返回多列不重复**

**多列不重复 = SummarizeColumns('多列重复'[年份],'多列重复'[姓名])**

表名1[字段名1],表名2[字段名2],新字段名,新字段聚合表达式　注：可以省略Calculate

**三、返回汇总表【常用】**

**汇总表 = SUMMARIZECOLUMNS('分组求和'[年份],'分组求和'[姓名], "总分",CALCULATE(sum('分组求和'[成绩])))**

表名1[字段名1],表名2[字段名2],Filter(表名,表名[字段名]="筛选条件"),新字段名,新字段聚合表达式
注：可以省略Calculate

**四、返回带筛选功能的汇总表【常用】**

**汇总表筛选 = SUMMARIZECOLUMNS('分组求和'[年份],'分组求和'[姓名], Filter('分组求和','分组求和'[科目]="数学"),"数学",CALCULATE(sum('分组求和'[成绩])))**

# 06.从来就没有什么上下文

2020年6月26日 22:42

**筛选上下文：针对列的筛选**
**行上下文：行上下文，行上下文=当前行**

**重点：**
**1. 行上下文不会自动转换成筛选上下文，如果需要转换，则必须使用Calculate函数**
**2. 度量值自带天然的Calculate函数**

**例如：**
**度量值 = sum('销售表'[销售数量])　把度量值放到矩阵中就可以实现自动筛选**
**但是如果，新建列的时候使用了 新列名 = sum('销售表'[销售数量]) 那么每一行的值都是最终的sum('销售表'[销售数量]) 的总值，不会实现筛选**
**这时，使用Calculate函数转换，即可在新建列上实现自动筛选，新列名=Calculate(Sum('销售表'[销售数量]))**

**回来看一下Filter函数的案例:【因为Filter是逐行扫描，是行上下文，需要转换】**

**Filter销售量 = Calculate([销售量],Filter('店铺表',[销售量]>200))　正确**

**Filter销售量 = Calculate([销售量],Filter('店铺表',Sum('销售表'[销售数量])>200))　错误**

**Filter销售量 = Calculate([销售量],Filter('店铺表',Calculate(Sum('销售表'[销售数量]))>200))　正确**

**为什么[销售量]就不用套Calculate?**
**因为所有的度量值都自带Calculate函数的功能**
**销售量 = Sum('销售表'[销售数量])　等同于　销售量 = Calculate(Sum('销售表'[销售数量]))**

**度量值:**
**总销量 = SUM('表'[销售数量])**

**新建列:**
**总销量1 = SUM('表'[销售数量])**
**总销量2 = CALCULATE(SUM('表'[销售数量]))**

**所谓的筛选上下文，就是度量值，度量值自带天然的Calculate函数**

**没有筛选功能的新建列就是行上下文，如果行上下文想转成筛选上下文，它是不会自动转的，你要手动套上一个Calculate函数**

**度量值:**
**最大值 = max('案例'[成绩]**
**新建表:**
**汇总表筛选 = SUMMARIZECOLUMNS('案例'[年份],'案例'[姓名], Filter('案例','案例'[科目]="数学"),"数学",CALCULATE(max('案例'[成绩])))**

**汇总表筛选 = SUMMARIZECOLUMNS('案例'[年份],'案例'[姓名], Filter('案例','案例'[科目]="数学"),"数学",[最大值]))**

**从来就没有什么上下文，都是软件开发者给它起的名字，你且记住：要筛选就用度量值，或者 calculate(聚合函数) ，因为度量值天生就带calculate**

# 07.当神秘的x系列遇到了Earlier函数

2020年6月27日 21:39

2020年5月5日

SumX、AverageX、MaxX、MinX...它们与Filter函数属于一种类型——行上下文函数

语法：SumX( '表名' ,算术表达式)

[销售数量]*[单价]

1、逐行扫描
2、执行运算，每一行都会返回一个值，例如 15x1.1=16.5
3、SumX函数记住了每一行返回的值，最后把所有的值相加求和。

| 销售数量 | 单价 | 销售金额 |
|---|---|---|
| 15 | 1.1 | 15x1.1=16.5 |
| 16 | 1.5 | 16x1.5=24 |
| 18 | 1.6 | 18x1.6=28.8 |
| 21 | 1.7 | 21x1.7=35.7 |
| 30 | 2.1 | 30x2.1=63 |

使用第06课的课件【表】
总金额 = SUM('表'[销售金额])
总成本 = SUM('表'[销售成本])

新建列：毛利2 = SUMX('表',[销售金额]-[销售成本])
新建列：毛利2 = CALCULATE(SUMX('表',[销售金额]-[销售成本]))

作用：使用Earlier函数得到当前行
经典语句：

```
Calculate([度量值],Filter('表名',[列名]=Earlier([列]))
它与索引列结合起来，可以去关联上一行或上几行
Calculate([度量值],Filter('表名',[索引]=Earlier([索引]-1))
```

应用：观察新建列的数值是什么?整个国产的销量
度量值• 销售量 = SUM('销售表'[销售数量] )
新建列• 列 = CALCULATE([销售量],FILTER('产品表','产品表'[巧克力种类]=EARLIER('产品表'[巧克力种类])))
这个函数需要在实践中不断体验，如果你觉得不好理解，没关系，下节课我们讲VAR函数时会介绍一种可以取代Earlier函数的方法。

```
滚动销量 =
SUMX (
    FILTER (
        FILTER ( '销售表', '销售表'[货号] = EARLIER ( '销售表'[货号] ) ),
        '销售表'[销售日期] <= EARLIER ( '销售表'[销售日期] )
    ),
    '销售表'[销量]
)
```

'销售表'[销售日期] <= EARLIER ( '销售表'[销售日期] )
第一天直接照抄数量
第二天加上第一天
第三天加上第一和第二天
.....以此类推

# 08.安全除法与逻辑判断

2020年6月27日 22:48

**语法：DIVIDE(分子,分母,[替换结果])**

**替换结果可以省略不写，省略时返回为空。**

**If（西瓜坏了，换货，[退货]）**

**第三参数可以省略不写，省略时返回为空。**

**If适合条件比较少的时候使用，如果条件太多，嵌套太多，不方便使用**

**称呼 = IF([性别]="男","先生","女士")**

**switch（表达式，值1，结果1，…，[else]）**

**最后一个参数可以省略不写，省略时返回为空。**

**月份 = SWITCH('switch函数'[月],**

**1, "一月", 2,"二月",**

**3, "三月", 4,"四月",**

**5, "五月", 6,"六月",**

**7, "七月", 8,"八月",**

**9, "九月", 10,"十月",**

**11,"十一月", 12,"十二月",**

**"未能识别")**

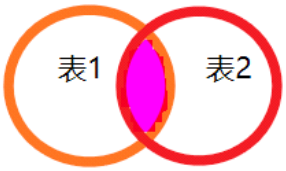
# 09.不请自来和迪卡尔积

2020年6月27日 22:51

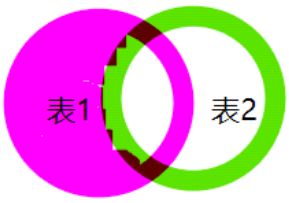
交叉相同：intersect(表1，表2)

除去相同：Except(表1，表2)

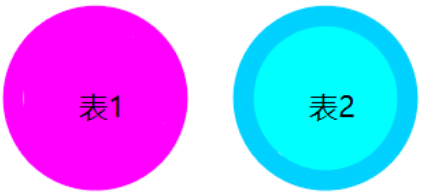
全部：Union(表1，表2)




交叉




除去




全部

## 迪卡尔积：CROSSJOIN（表1，表2）

# 10.起名大师VAR

2020年6月28日 21:05

## 一、代替当前行函数 【不推荐】

```
出现次数 = countrows(
    filter('表','表'[姓名] = EARLIER('表'[姓名])
     &&
    '表'[序号]<=EARLIER('表'[序号])))
```

```
出现次数2 =
VAR myname = '表'[姓名]
VAR index = '表'[序号]
return
COUNTROWS(filter('表','表'[姓名]=myname && '表'[序号]<=index))
```

VAR函数工作过程是先识别当前行中的姓名和序号，并记录下来结果，然后在return中引用，与相前行效果相同

## 二、书写更简洁

```
评价 =
VAR zongfen = '表1'[数学] + '表1'[语文] + '表1'[英语]
return
if (zongfen >=270,"优秀","一般")
```

注意变量名用英语或汉语拼音

## 三、不理解上下文也可以用

```
Filter销售量 = Calculate([销售量],Filter('店铺表',[销售量]>200))
```

正确

```
Filter销售量 = Calculate([销售量],Filter('店铺表',Sum('销售表'[销售数量])>200))
```

错误

```
Filter销售量 = Calculate([销售量],Filter('店铺表',Calculate(Sum('销售表'[销售数量]))>200))
```

正确

如果使用VAR

```
VAR  xsl = Sum('销售表'[销售数量])
return
Calculate([销售量],Filter('店铺表',xsl>200))
```

# 11.你是我的唯一Values和DISTINCT

2020年6月29日 2:59

**返回由一列构成的一个表，该表包含来自指定表或列的非重复值。表中重复值将被删除，返回唯一值。**
**Values函数生成的表是一张虚拟表。**
**在很多时候使用Filter、Calculate、Countrows、SumX、TopN这些函数，需要引用表而不能直接引用列。Values函数的功能是把列转化成包含列的表，这时只需要嵌套一个Values函数就可以引用了。**
**注意：这张表，是不重复的信息表！**

**假设我没有多张表的关系连接，所有数据都在销售表中：**


**在新建表那里生成一张虚拟表，从销售数据表中拿到的不重复门店，这样就可以统计销售量大于300的门店销售量了**

**经典语句：** **Calculate([度量值],Filter(Values(‘表’[列名]),...))**

**DISTINCT(表名[字段名])** **返回：去重后，唯一值的列**

**DISTINCT(表名)** **返回：只包含非重复行的表**

VALUES 返回一列的唯一值列表， DISTINCT同样是该功能，那么二者有什么不同呢?
一个区别是对于空白行的处理上，VALUES包括没有匹配的空白行。但是DISTINCT不返回没有匹配的空白行。
另外的区别是DISTINCT函数允许列名或任何有效的表表达式作为其参数，但VALUES函数仅接受列名或表名作为参数。

## 12. 清除筛选计算占比三个ALL

2020年5月4日 1:57

**ALL函数的功能：清除筛选条件**

**ALL函数的应用：清除表中所有行的筛选器以及创建针对表中所有行的计算非常有用。**

**ALL函数的返回值：已清除筛选的表或列**

**注意事项：ALL 函数的参数必须是对表的引用或对列的引用。 不能将 ALL 函数与表的表达式或列的表达式一起使用。**

**All销售量1= Calculate([销售总量],All('销售表'))**

| 商品种类 | 散 | 盒 | 总计 |
|---|---|---|---|
| 蔬菜 | 52730 | 52730 | 52730 |
| 水果 | 52730 | 52730 | 52730 |
| 肉品 | 52730 | 52730 | 52730 |
| 水产 | 52730 | 52730 | 52730 |
| 总计 | 52730 | 52730 | 52730 |

**一般ALL(表名)会在计算占比时使用**

**占比=[销售总量]/[ALL销售量1]**

**注：配合切片器使用，可以筛选门店、日期等**

**All销售量2= Calculate([销售总量],All('商品表'[商品种类]))**

| 商品种类 | 散 | 盒 | 总计 |
|---|---|---|---|
| 蔬菜 | 31638 | 21092 | 52730 |
| 水果 | 31638 | 21092 | 52730 |
| 肉品 | 31638 | 21092 | 52730 |
| 水产 | 31638 | 21092 | 52730 |
| 总计 | 31638 | 21092 | 52730 |

**注意：ALL函数在引用列的时候，必需与矩阵的行和列在同一张表**

**假设，你ALL引用的是商品表，但是矩阵行和列来自店铺表就会出错。**

**必须保证，ALL所清除的筛选列和初始筛选条件中的筛选列是同一张表的同一列。**

**All销售量3= Calculate([销售总量],All('商品表'[商品种类],'商品表'[商品规格]))**

**引用多个列时，必须是同一张表**

**All销售量3= Calculate([销售总量],All('商品表'),ALL('店铺表'))**

Allexcept函数：除...之外

语法：Allexcept(表名[字段名4]) 等同于 All(表名[字段名1],表名[字段名2],表名[字段名3])

# 12.1 Allselceted函数 【占比显示100%】

2020年5月4日 3:18

| 商品种类 | 商品种类 | 占比1 |
|---|---|---|
| ☑ 蔬菜 | 蔬菜 | 36.21% |
| ☑ 水果 | 水果 | 37.15% |
| ☑ 肉品 | 肉品 | 17.80% |
| ☐ 水产 | 总计 | 91.16% |

**占比1=[销售总量]/[ALL销售量1]**

**All销售量1= Calculate([销售总量],All('销售表'))**

**如果想让这里显示100%，使用Allselceted函数**

**Allselceted销售量= Calculate([销售总量],Allselceted('销售表'))**

**占比2=[销售总量]/[Allselceted销售量]**

| 商品种类 | 商品种类 | 占比1 | 占比2 |
|---|---|---|---|
| ☑ 蔬菜 | 蔬菜 | 36.21% | 39.72% |
| ☑ 水果 | 水果 | 37.15% | 41.58% |
| ☑ 肉品 | 肉品 | 17.80% | 18.70% |
| ☐ 水产 | 总计 | 91.16% | 100.00% |

**总结:**

**1、当ALL参数为表时，忽略所有的筛选条件，无论是该图表内还是外部切片器**

**2、当ALL参数为列时，忽略该列筛选，其它图表字段或外部筛选对基产生作用**

**3、当ALLselected参数为表时，忽略该图表的筛选条件，其它外部筛选对其产生作用**

**4、当ALLselected参数为列时，忽略该图表在该列的筛选条件，其它该图表字段或外部筛选对其它产生作用。**

代码：

2020年6月28日 22:31

**总销量 = SUM('表'[销量] )**
**ALL表 = CALCULATE([总销量],ALL('表'))**
**占比 = [总销量]/[ALL表]**
**Allselceted销量 = Calculate([总销量],ALLSELECTED('表'))**
**占比2 = [总销量]/[Allselceted销量]**

**ALL列 = CALCULATE([总销量],ALL('表'[包装规格]))**
**占比3 = [总销量]/[ALL列]**
**矩阵：行【商品名称】，列【包装规格】，值【占比2】**
**比如：苹果是一个单位100%，盒占54%，散占46%**

**注意：引用列的时候，必需是同一张表**

| 商品名称 | 盒 | 散 | 总计 |
|---|---|---|---|
| 后尖 | 2 | 6 | **8** |
| 黄瓜 | 8 | 9 | **17** |
| 苹果 | 7 | 6 | **13** |
| 前尖 | 3 | 5 | **8** |
| 西瓜 | 3 | 6 | **9** |
| 西红柿 | 5 | 1 | **6** |
| **总计** | **28** | **33** | **61** |

| 商品名称 | 盒 | 散 | 总计 |
|---|---|---|---|
| 后尖 | 61 | 61 | **61** |
| 黄瓜 | 61 | 61 | **61** |
| 苹果 | 61 | 61 | **61** |
| 前尖 | 61 | 61 | **61** |
| 西瓜 | 61 | 61 | **61** |
| 西红柿 | 61 | 61 | **61** |
| **总计** | **61** | **61** | **61** |

| 商品名称 | 盒 | 散 | 总计 |
|---|---|---|---|
| 后尖 | 8 | 8 | **8** |
| 黄瓜 | 17 | 17 | **17** |
| 苹果 | 13 | 13 | **13** |
| 前尖 | 8 | 8 | **8** |
| 西瓜 | 9 | 9 | **9** |
| 西红柿 | 6 | 6 | **6** |
| **总计** | **61** | **61** | **61** |

| 商品名称 | 盒 | 散 | 总计 |
|---|---|---|---|
| 后尖 | 3.28% | 9.84% | **13.11%** |
| 黄瓜 | 13.11% | 14.75% | **27.87%** |
| 苹果 | 11.48% | 9.84% | **21.31%** |
| 前尖 | 4.92% | 8.20% | **13.11%** |
| 西瓜 | 4.92% | 9.84% | **14.75%** |
| 西红柿 | 8.20% | 1.64% | **9.84%** |
| **总计** | **45.90%** | **54.10%** | **100.00%** |

商品名...
- 后尖
- 黄瓜
- 苹果
- 前尖
- 西瓜
- 西红柿

商品...
- 肉
- 蔬菜
- 水果

包装规格
- 盒
- 散

| 商品名称 | 盒 | 散 | 总计 |
|---|---|---|---|
| 后尖 | 0.25 | 0.75 | **1.00** |
| 黄瓜 | 0.47 | 0.53 | **1.00** |
| 苹果 | 0.54 | 0.46 | **1.00** |
| 前尖 | 0.38 | 0.63 | **1.00** |
| 西瓜 | 0.33 | 0.67 | **1.00** |
| 西红柿 | 0.83 | 0.17 | **1.00** |
| **总计** | **0.46** | **0.54** | **1.00** |

# 13. 只有一个值H与S

2020年6月30日 9:09

返回值：“真”或“假”

经典语句：if(Hasonevalue(‘表名’[列名]),[度量值],blank())

例：if(Hasonevalue(‘日期’[年份季度]),[filter销售量],blank())

Filter销售量 = Calculate([销售量],Filter(‘店铺表’,[销售量]>300))

店铺表 = Values(‘销售数据表’[门店])

| 日期 | | | |
|---|---|---|---|
| 2019/1/1 | 2019/12/31 | | |
| | | | |
| 年份季度 | 销售量 | Filter销售量 | Hasonevalue |
| 2019Q1 | 130 | | |
| 2019Q2 | 645 | | |
| 2019Q3 | 1990 | 658 | 658 |
| 2019Q4 | 3220 | 1960 | 1960 |
| 总计 | 5985 | 5210 | |

说明：当求总计时，筛选上下文件是2019年全年，在日期表中，2019年有4个季度，不是唯一值，所以返回空值

总销量 = SUM('销售表'[销售数量])

filter销量 = CALCULATE([总销量],FILTER('门店',[总销量]>100))

Has销量 = IF(HASONEVALUE('销售表'[日期].[日]),[filter销量],BLANK() )

当指定列中只有一个值时返回该值，否则返回替代结果，省略则返回空值。

SELECTEDVALUE 与 IF+HASONEVALUE 等价，而且公式更简洁

if(Hasonevalue(‘表名’[列名]),返回值,blank())

- 如果是唯一值，那就返回值，否则返回空

SELECTEDVALUE (‘表名’[列名],【代替（省略返回空）】)

- 当指定列中只有一个值时返回该值，否则返回替代结果，省略则返回空值。

# 代码：

2020年6月30日 10:33

总销量 = SUM('销售表'[销售数量])

filter销量 = CALCULATE([总销量],FILTER('门店',[总销量]>100))

Has销量 = IF(HASONEVALUE('销售表'[日期].[日]),[filter销量],BLANK() )

SEL销量 = **SELECTEDVALUE（‘表名’ [列名]）**

## 14.武林要以和为贵小伙子你不讲武德

2020年6月30日 18:21


RANKX 适合计算明细级别的排序数据

新建列

单品销售排名 = RANKX(FILTER('表','表'[品种]=EARLIER('表'[品种])),'表'[销售金额])

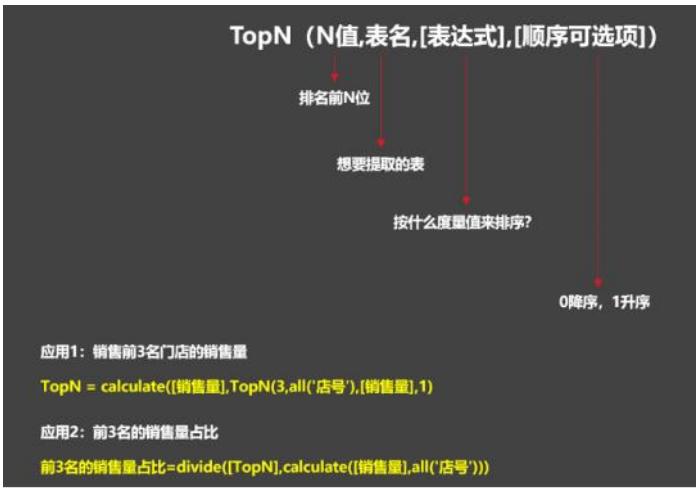
度量值

排名 = RANKX(ALL('表'),[总金额],,DESC)

## 14.1 TopN

2020年6月30日 18:58

**TOPN 则可以批量返回结果，从一张表中返回所有满足条件的前 N 行记录。**




**总金额2 = SUM('表2'[销售金额])**
**前3名 = CALCULATE([总金额2],TOPN(3,VALUES('表2'[店号]),[总金额2],1))**
**占比 = [前3名]/[总金额2]**

# 15.虽然不智能但是很有用【日期时间函数】

2020年6月30日 19:59

**一、DATE 返回日期时间格式的日期值，注意：当年份在 0-99 之间，DATE 返回的年份会此在基础上自动加上 1900。超过 99，直接将值用作年份**

**DATE ( <Year>, <Month>, <Day> )**

**日期 = DATE([年],[月],[日])**

**二、对时间元素的提取**

| | |
|---|---|
| 年 | YEAR |
| 月 | MONTH |
| 日 | DAY |
| 季度 | QUARTER |
| 小时 | HOUR |
| 分钟 | MINUTE |
| 秒 | SECOND |

**当前日期和时间：NOW**

**当前日期：TODAY**

**日期中的星期：WEEKDAY([日期],2)　#2 以周一为起始**

**日期在当年内的周：WEEKNUM ( [日期],2 )　#2 以周一为起始**

**三、返回按指定月数平移后的日期**

**EDATE ( 日期, 向前或向后月份 )　# 使用正数向后，使用负数向前**

**例如：2019年12月9日向后一个月是2020年1月9日**

**四、返回指定月数平移后的月份的最后一天**

**EOMONTH ( 日期, 向前或向后月份 )**

**例如：2019年12月9日向后一个月的最后一天是2020年1月31日**

**五、TIME 将数字形式的小时、分钟和秒转换为日期时间格式**

**TIME ( <Hour>, <Minute>, <Second> )**

★ **六、保质期函数**

**DATEDIFF(起始日期,结束日期,DAY)+1　# 如果计算保质期要加1**

# 16.可以不用但是要会 【文本函数】

2020年5月12日 10:02

**find(查找哪个字符，去哪里查找，从哪一个位置查找，如果找不到返回什么值) 第三和四参数可以省略，返回字符在第几位**

**search(查找哪个字符，去哪里查找，从哪一个位置查找，如果找不到返回什么值) 第三和四参数可以省略，返回字符在第几位**

**注意：Find查找字符区分大小写，search不区分大小写，其它功能完全一样。**

**left从左向右取=left([字段名],取几个字符)**
**right从右向左取=right([字段名],取几个字符)**
**mid从中间开始取=mid([字段名]，从第几个取，取几个)**
**len长度=len([字段名])**

**【新建列】上下装 = if(find("裤",'表'[商品名称],1,999)<999 || find("裙",'表'[商品名称],1,999)<999,"下装","上装")**

# 附:转换函数

2020年6月30日 19:59

## 一、检查值是否为空，并返回 TRUE 或 FALSE

**ISBLANK ( )**

## 二、检查值是否为错误，并返回 TRUE 或 FALSE

**ISERROR ( )**

## 三、检查值是否是逻辑值(TRUE 或 FALSE)，并返回 TRUE 或 FALSE

**ISLOGICAL ( )**

## 四、检查值是否为非文本(空白单元格不是文本)，并返回 TRUE 或 FALSE

**ISNONTEXT ( )**

## 五、检查值是否为数字，并返回 TRUE 或 FALSE

**ISNUMBER ( )**

## 六、检查值是否为文本，并返回 TRUE 或 FALSE

**ISTEXT ( )**

## 七、检查表或表表达式是否为空

**ISEMPTY()**

# 17.这回智能真的来了【时间智能函数】

2020年7月1日 9:32

# 17.1 日期表

2020年7月1日 9:46

```
日期表 = ADDCOLUMNS(
CALENDAR(date(2019,1,1),date(2019,12,31)),
"年", YEAR ( [Date] ),
"季度", ROUNDUP(MONTH([Date])/3,0),
"月", MONTH([Date]),
"周", weeknum([Date]),
"年季度", year([date]) & "Q" & ROUNDUP(MONTH([Date])/3,0),
"年月", year([Date]) * 100 + MONTH([Date]),
"年周", year([Date]) * 100 + weeknum([Date]),
"星期几", WEEKDAY([Date])
)
```

# 17.2 计算累加值

2020年7月1日 9:46

**年初至今(YTD)、季初至今(QTD)和月初至今(MTD)**
**累计是指定维度上值的求和，时间的累计就是在时间维度上计算当期值然后求和而来。**
**比如年累计（MTD）是将当月的值累加求和，下个月的值重新计算。**

**一、计算本年的年初至今(YTD)、季初至今(QTD)和月初至今(MTD)**

```
销量 = SUM('表'[销售])
年初至今 = TOTALYTD([总销量],'表'[日期])
月初至今 = TOTALMTD([总销量],'表'[日期])
季出至今 = TOTALQTD([总销量],'表'[日期])

年初至今 = TOTALYTD([总销量],'表'[日期],"6-30")  # 从7月1日开始计算为一年
```

**如果您单位从每年3月1日开始计算，那么需要考虑平年和闰年，方法请自行在百度搜索**

**二、计算去年同期** → 去年同期

```
去年同期年初至今 = TOTALYTD([总销量],SAMEPERIODLASTYEAR('表'[日期]))
去年同期月初至今 = TOTALMTD([总销量],SAMEPERIODLASTYEAR('表'[日期]))
去年同期季初至今 = TOTALQTD([总销量],SAMEPERIODLASTYEAR('表'[日期]))
```

**同比增长（下降）率=(本期数-去年同期数)/去年同期数×100%**
**环比增长（下降）率=(本期数-上期数)/上期数×100%**

# 17.3 指定时间段的累加值

2020年6月30日 10:29

## Datesbetween（'表'[日期],开始日期，结束日期）

**指定时间销量 = CALCULATE([总销量],DATESBETWEEN('表'[日期],DATE(2019,4,1),DATE(2020,4,30) ))**

## 17.4 一定间隔时间段

2020年6月30日 11:46

**ENDOFMONTH(日期列，年度结束日期)　　# 返回当月最后一天**
**STARTOFMONTH(日期列，年度结束日期)　# 返回当月第一天**

**datesinperiod函数：指定开始日期,向前或向后多少天、月、季度、年**

**DAY,month,quarter,year**

**语法：datesinperiod(日期列,开始日期,移动间隔,移动粒度)**

**每月最后3天销量 = CALCULATE([销售量],DATESINPERIOD('表2'[日期],ENDOFMONTH('表2'[日期]),-3,DAY))**
**每月前3天销量 = CALCULATE([销售量],DATESINPERIOD('表2'[日期],STARTOFMONTH('表2'[日期]),3,DAY))**

**拓展知识：**
**计算每年1月1日的销量 = CALCULATE([销售量],STARTOFyear('表'[日期]))**
**计算每年12月31日销量 = CALCULATE([销售量],ENDOFyear('表'[日期]))**

**计算指定年度开始前5天的销量**
**= CALCULATE([销售量]，DATESINPERIOD('表'[日期],STARTOFMONTH('表'[日期],"5-31"),5,DAY))**

**求最近30天的移动平均**
**= Calculate([销售量], datesinperiod(‘表’[日期]，max(‘表’[日期]),-30,day)/30**

**切记：你的日期列是唯一值，否则使用Values创建一个唯一值的表或者使用去重方法均可**

# 17.5 日期中的最大值与最小值

2020年6月30日 12:21

时间点函数：用于指定某一个特定日期，它们返回的是一个有唯一值的表，这个值就某一个日期。

例如：Firstdate函数求最早日期，Lastdate函数求最晚日期。
Firstdate（’日期表’[日期]） Lastdate （’日期表’[日期])

其实就是：
min（’日期表’[日期]） max（’日期表’[日期])

结果一样，但是有区别， Firstdate函数和Lastdate函数可以直接引用表，但是Min和MAX只能是字段。

Firstdate（ALL(’日期表’[日期]))
Lastdate （ALL(’日期表’[日期]))

# 17.6 Dateadd函数 【同期、环比】

2020年6月30日 12:27

特别注意：（1）日期必需是连续的（2）只能引用日期表，不能是销售表的日期列

常见应用：去年同期销售=calculate([销售额],dateadd（‘日期表’[日期],-1,year))
上月环比销售=calculate([销售额],dateadd（‘日期表’[日期],-1,month))

**今年1月1日至今 = TOTALYTD([总金额],'日期表'[Date])**
**同期 = CALCULATE([今年1月1日至今],DATEADD('日期表'[Date],-1,YEAR))**

**本月1月1日至今 = TOTALMTD([总金额],'日期表'[Date])**
**同期 = CALCULATE([今年1月1日至今],DATEADD('日期表'[Date],-1,month))**

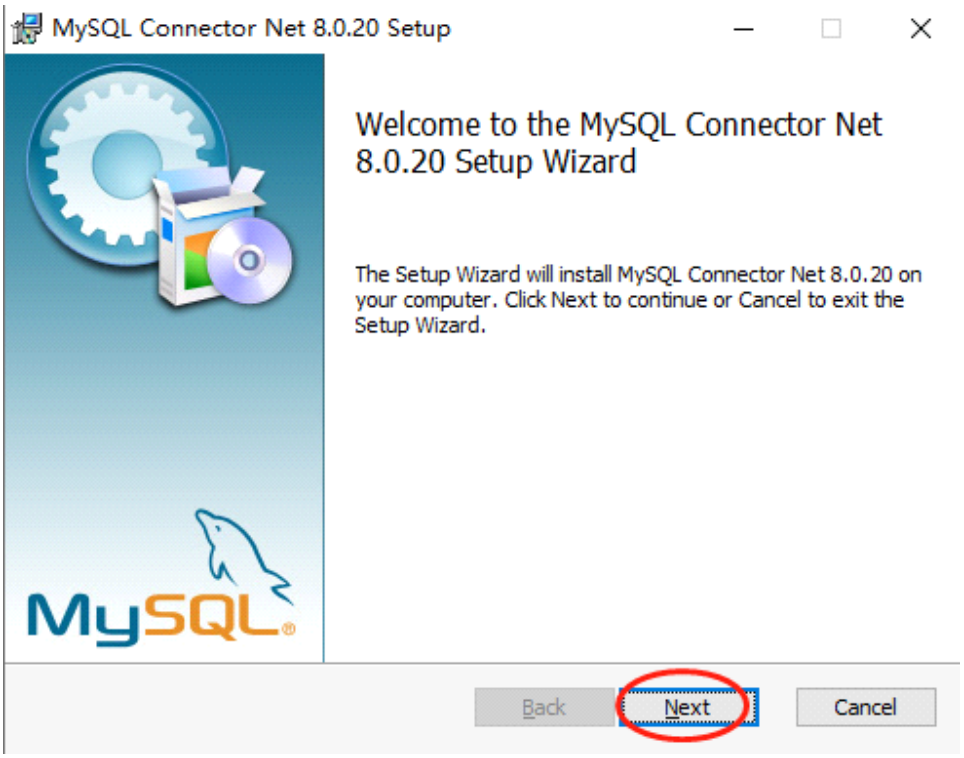
# 18.中文排序

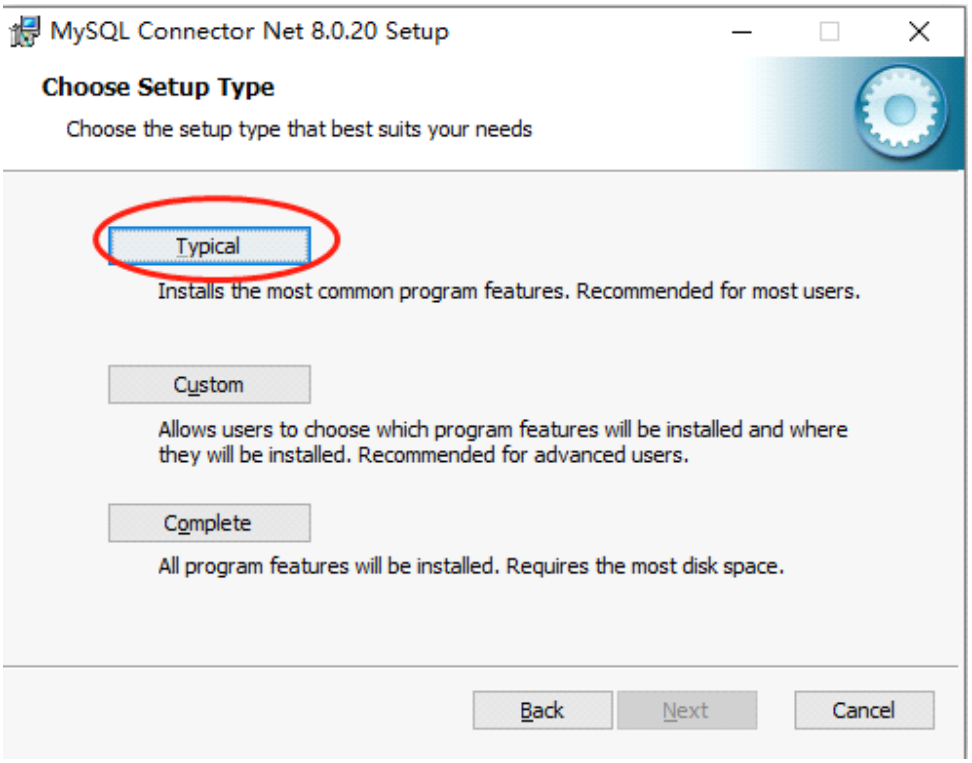
2020年7月2日 4:07

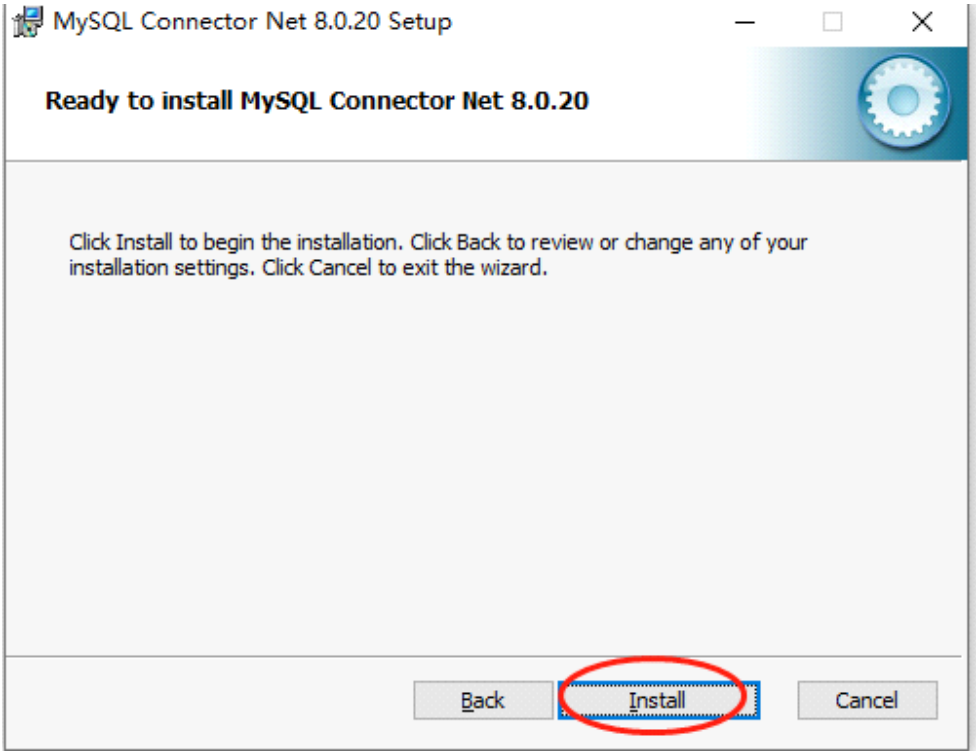
# 19.连接MySQL

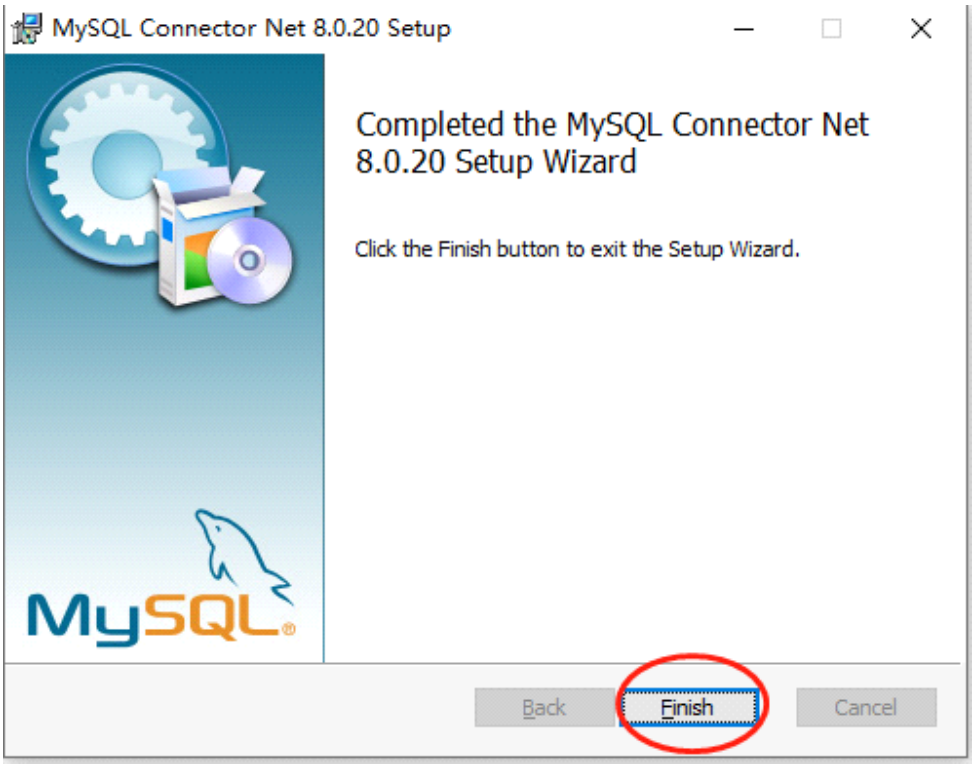
2020年7月2日 4:07

## 1、 到我的评论区置顶链接中下载 ： mysql-connector-net-8.0.20.msi




MySQL Connector Net 8.0.20 Setup
Ready to install MySQL Connector Net 8.0.20
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


MySQL Connector Net 8.0.20 Setup
Completed the MySQL Connector Net 8.0.20 Setup Wizard
Click the Finish button to exit the Setup Wizard.
MySQL
Back
Finish
Cancel

# 20.获取数据、向下钻取与筛选

2020年7月2日 4:18

# DAX语言查询及进阶篇观看地址：

2020年7月1日 20:30

## 微软官方DAX语言查询：中文版

https://docs.microsoft.com/zh-cn/dax/new-dax-functions

## powerbi进阶篇 第一季 销售案例

https://www.bilibili.com/video/BV18C4y1H7b9

## powerbi进阶篇 第二季 商品案例

https://www.bilibili.com/video/BV14Z4y1p7fY

## powerbi进阶篇 第三季 人力与财务案例

https://www.bilibili.com/video/BV1Je411W76y