第 3 章

查　询　表

DAX 具有两大主要功能：查询与计算(计算列与度量值)。DAX 利用查询的筛选、分组和汇总功能来减少可能较大的数据量和较复杂的模型，类似于 SQL 查询的功能。Excel Power Pivot 中的查询表是通过 EVALUATE 声明来完成的。在 Power BI 中创建查询表更为简捷，可直接采用“新建表”完成。

3.1　表函数

3.1.1　统计学基础

在概率与数理统计中，有元素、集合、对象、事件等概念。集合由元素组成，组成集合的每个对象被称为组成该集合的元素。集合内的元素可为数值型、文本型或其他类型，每个集合至少包含两个子集：该集合本身和空值。空集用∅表示，全集用 U 表示。

集合与事件间的关系分为包含($A \in B$)、交($A \cap B$)、并($A \cup B$)、互斥、对立、差($A - B$)几种。

互斥事件与对立事件的区别：互斥事件中的 A、B 不可能同时发生，也可以都不发生，互斥集的 $A \cap B = \varnothing$、$A \cup B = A + B$。对立事件中的 A、B 不可能同时发生，但必须有一个发生，对立集的 $A \cap B = \varnothing$、$A \cup B = \mathrm{U}$。对立事件肯定为互斥事件，而互斥事件不一定是对立事件。

以下是统计学中的一些基本公式。

1. 集合与元素

$$x \in A \text{ 或 } A \ni x \tag{3-1}$$

式(3-1)中：x(小写)是集合中的一个元素，A(大写)是集合。$x \in A$ 相当于计算机语言中的 x in A；$A \ni x$ 相当于计算机语言中的 A contains x。二者代表的意义是相同的。

$$x \notin A \text{ 或 } A \not\ni x \tag{3-2}$$

式(3-2)中：$x \notin A$ 相当于 x not in A；$A \not\ni x$ 相当于 A not contains x。二者代表的意义是相同的。

2. 集合与事件

$$A \cup B \tag{3-3}$$

式(3-3)中：集合具备确定性、互异性、无序性这 3 个特性。互异性是指同一个集合中的元素是互不相同的，因而 $A \cup B = A + B - A \cap B$。

$$\forall X \in A : X \tag{3-4}$$

式(3-4)中：$\forall$ 是给定集合的整体，X 代表的是子集(subset)，A 代表的是集合(set)，冒号(:)代表的是给定集合。例如“$\forall X \in A : X$ 是偶数”代表的意义为“A 集合、X 子集中的所有偶数”。

3.1.2 ALL()

当涉及单表数据演示时，本书主要用到的数据源(工作簿名称 DOCK. xlsx，表名称 DK)见表 3-1。

表 3-1 表名称(DK)

日期	产品	包装方式	入库	出库	吨数	方数	等级
2020/9/2	钢化膜	散装	76	35	18.86	56	A
2020/9/27	蛋糕纸	箱装	32	12	21.27	85	A
2020/11/2	油漆	桶装	55	23	20.89	72	B
2021/9/29		捆	87	33	21.22		C
2021/9/29	净化剂	桶装	93	42	20.89	59	B
2021/10/11	苹果醋	箱装	78	36		65	B
2021/11/14	包装绳	扎	65	32	22.89	69	A

在 DAX 中，ALL()函数主要有两大功能：引用(返回表)及调节器(移除筛选)，该函数看似简单但功能强大却又不好理解，它是 DAX 中一个相当重要的函数，语法如下：

```
ALL([<table> | <column>[, <column>[, <column>[, …]]]] )
```

说明：ALL()函数不单独使用，常用作 FILTER()、CALCULATE()等函数的参数。ALL()在 FILTER()函数中用于引用表或列并返回表。在 CALCULATE()函数中用于更改执行过其他计算的结果集，返回表中的所有行或列中的所有值。忽略可能已应用的任何筛选器。

ALL()函数用于返回表的所有值，表达式如下：

```
EVALUATE                        //ch3 - 001,EVALUATE 对大小写不敏感
ALL('DK')                       //复制表,该表达式与'DK'等效.

-- 类似于微软 OLEDB 中的 SQL 语句 SELECT * FROM [DK$ ]
-- 说明: 表达式中"-- "       "//"均为 DAX 中的单行注释符,二者功能上无区别
```

返回的值是完整的 DK 表数据。

注意：当数据量过大时，应尽量避免使用 ALL()引用复制或引用整个表，特别是复制或引用整个的位于数据模型多端的数据明细表，它会使运行效率低下。

All()函数不仅可以引用表，还可以引用列，它们的用法是一样的，表达式如下：

```
EVALUATE                    //ch3 - 002
ALL('DK'[包装方式])         //对列中的值进行去重

-- 类似于微软 OLEDB 中的 SQL 语句 SELECT DISTINCT 包装方式 FROM [DK $ ]
```

返回的值如图 3-1 所示。

注意：ALL()函数能够使用列时尽量不使用表，ALL('表'[列])方式可返回列的不重复值，在数据量过大时，此操作可极大地提升效率。

ALL()函数可接受单列或多列作为参数，All()函数所有引用列参数必须来自同一张表，表达式如下：

```
EVALUATE        //ch3 - 003
ALL('DK'[包装方式],'DK'[产品] )

-- 类似于微软 OLEDB 中的 SQL 语句 SELECT DISTINCT 包装方式,产品 FROM [DK $ ]
```

返回的值如图 3-2 所示。

包装方式
散装
箱装
桶装
捆
扎

图 3-1 单列引用

包装方式	产品
散装	钢化膜
箱装	蛋糕纸
箱装	苹果醋
桶装	油漆
桶装	净化剂
捆	
扎	包装绳

图 3-2 多列引用

ALL()的去重功能应用举例，表达式如下：

```
M.引用: = COUNTROWS(ALL(DK))                          //7

M.去重: = COUNTROWS(ALL(DK[包装方式],DK[等级]))       //6
```

在 DK 表的 7 行数据中，有两行桶装等级为 B 的数据，去重后的行数为 6。

ALL()函数在 FILTER()中使用时是表函数，用于表的引用或表列的删除重复项，返回的值会移除外部筛选上下文。应用举例，表达式如下：

```
M.行数 A: =  COUNTROWS(ALL('DK'))

M.行数 AF: =  COUNTROWS( FILTER(ALL('DK'), 'DK'[包装方式] = "箱装")
```

ALL()用于扩展行数,FILTER()用于缩减行数,返回的值如图 3-3 所示。

行标签	M.行数A	M.行数AF
捆	7	2
散装	7	2
桶装	7	2
箱装	7	2
扎	7	2
总计	7	2

(a) 行标签为包装方式

行标签	M.行数A	M.行数AF
A	7	2
B	7	2
C	7	2
总计	7	2

(b) 行标签为等级

图 3-3 移除筛选

3.1.3 VALUES()

VALUES()函数用于返回在当前筛选器中计算的列的不同值,该函数也是 DAX 中一个相当重要的且易与 ALL()混淆的函数,语法如下:

```
VALUES(< TableNameOrColumnName >)
```

VALUES()函数删除重复项返回唯一值,表达式如下:

```
EVALUATE        //ch3 - 004
VALUES('DK'[包装方式] )
```

返回的值如图 3-4 所示。

VALUES()函数只接受单列作为参数,表达式如下:

```
EVALUATE        //ch3 - 005
VALUES('DK'[包装方式] ,'DK'[产品] )
```

以上查询引用了二列作为参数,返回的错误提示如图 3-5 所示。

包装方式
散装
箱装
桶装
捆
扎

图 3-4 单列引用

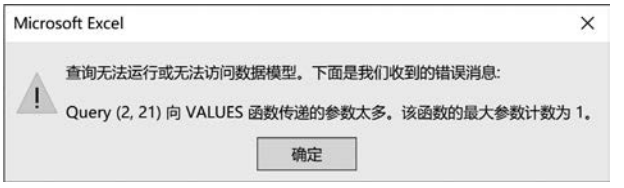


图 3-5 错误提示

创建度量值 M.行数 V 和 M.行数 VF,表达式如下:

```
M.行数 V: = COUNTROWS(VALUES('DK'))

M.行数 VF: = COUNTROWS( FILTER(VALUES('DK'),'DK'[产品] = "蛋糕纸"))
```

VALUES()与 ALL()返回的值对比如图 3-6 所示。

行标签	M.行数V	M.行数VF	M.行数A	M.行数AF
(空白)	1		7	1
包装绳	1		7	1
蛋糕纸	1	1	7	1
钢化膜	1		7	1
净化剂	1		7	1
苹果醋	1		7	1
油漆	1		7	1
总计	**7**	**1**	**7**	**1**

(a) 行标签为产品

行标签	M.行数V	M.行数VF	M.行数A	M.行数AF
捆	1		7	1
散装	1		7	1
桶装	2		7	1
箱装	2	1	7	1
扎	1		7	1
总计	**7**	**1**	**7**	**1**

(b) 行标签为包装方式

图 3-6　对比说明

注意：在度量值中，ALL()函数用于移除筛选，VALUES()函数用于保持外部筛选上下文并返回在当前筛选器中计算列的不同值，ALL()与 VALUES()在筛选的功能上是相反的。

3.2　高级筛选

FILTER()函数是一个基于条件表达式的表函数。它会根据第 2 个参数的筛选条件来筛选第 1 个参数的表，返回的是一个表。其筛选能力远大于 CALCULATE()函数的直接筛选方式，可适用于各类复杂筛选条件，但其运行效率不如 CALCULATE()的直接筛选。FILTER()函数用于获取外部筛选条件下的表，其返回值为表，语法如下：

```
FILTER(<table>,<filter>)
```

说明：FILTER()函数是迭代函数，其运行效率远不如 CALCULATE()函数。当 CALCULATE()函数的直接筛选能满足筛选要求时，尽量不要用 FILTER()函数；只有当 CALCULATE()函数不能满足筛选要求时，再使用 CALCULATE()＋FILTER()的嵌套用法。在使用 FILTER()函数时，FILTER()函数应尽量放在(来自模型中的一端的)查询表中使用，这样可以提升运行效率。

3.2.1　筛选应用

在 DAX 中，FILTER()是一个使用频率很高的函数。出于提升效率的考虑，在使用筛选函数 FILTER()前须谨记两个原则：①能够用 CALCULATE()函数的直接条件解决的就尽量不要用 FILTER()；②使用 FILTER()函数时，第 1 个参数能指定具体列的就尽量不要用整个表。

ALL()函数、DISTINCT()函数、VALUES()函数常用于 FILTER()函数的第 1 个参数。以 ALL()函数在 FILTER()中作筛选参数的应用举例，表达式如下：

```
EVALUATE       //ch3 - 006
FILTER ( ALL ( 'DK'[包装方式] ), 'DK'[包装方式] = "散装" )

-- 类似于微软 OLEDB SQL 语句 SELECT DISTINCT 包装方式 FROM [DK$] WHERE 包装方式 = "散装"
```

返回的值如图 3-7 所示。

在查询的过程中顺便对指定字段进行排序，表达式如下：

```
EVALUATE       //ch3 - 007
FILTER ( ALL ( 'DK'[包装方式], 'DK'[吨数] ),'DK'[吨数] > 21 )
ORDER BY 'DK'[吨数] DESC

-- 类似于 OLEDB SQL 语句 SELECT DISTINCT 包装方式,吨数 FROM [DK$] WHERE 吨数> 21
-- ORDER BY 吨数 DESC
```

返回的值如图 3-8 所示。

包装方式
散装

图 3-7　固定值筛选

包装方式	吨数
扎	22.89
箱装	21.27
捆	21.22

图 3-8　条件筛选

FILTER()函数的第 2 个参数为条件表达式，适用于<列>=[度量]、<列>=(公式)、<列>=<列>、[度量]=[度量]、[度量]=(公式)、[度量]=固定值等比较方式。以上表达式中的=(等号)也可以是< >、>、>=、<、<=等其他比较运算符。FILTER()函数的运算条件还可以为 &&、‖、IN、NOT 等逻辑运算符。

1. 单条件

以<列>=固定值为例，表达式如下：

```
EVALUATE       //ch3 - 008
FILTER ( 'DK', 'DK'[吨数] > 21 )

-- 类似于微软 OLEDB 中的 SQL 语句 SELECT * FROM [DK$] WHERE 吨数> 6
```

返回的值如图 3-9 所示。

日期	产品	包装方式	入库	出库	方数	吨数	等级
2021/11/14	包装绳	扎	65	32	69	22.89	A
2020/9/27	蛋糕纸	箱装	32	12	85	21.27	A
2021/9/29		捆	87	33		21.22	C

图 3-9　单条件筛选

2. 多条件

逻辑运算符有 &&(AND)、‖(OR)、IN、NOT,相关说明参见表 1-8。以逻辑关系与(AND、&&)为例,采用 AND()写法,表达式如下:

```
EVALUATE       //ch3-009
FILTER ( 'DK', AND('DK'[入库] > 60, 'DK'[入库] > 'DK'[出库] * 2.2 ))

-- 类似于 SQL 语句 SELECT * FROM [DK$] WHERE 入库> 60 AND 入库>出库 * 2.2
```

采用 && 写法,表达式如下:

```
EVALUATE
FILTER ( 'DK', 'DK'[入库] > 60&& 'DK'[入库] > 'DK'[出库] * 2.2 )
```

以上两种表达式返回的值如图 3-10 所示。

日期	产品	包装方式	入库	出库	方数	吨数	等级
2021/9/29		捆	87	32		21.22	C
2021/9/29	净化剂	桶装	93	42	59	20.89	B

图 3-10　多条件筛选

3.2.2　综合应用

FILTER()基于第 1 个参数来选择表或列,基于第 2 个参数中的条件表达式实现对表中行的筛选。FILTER()的第 1 个参数可以是具体的表或表函数的返回值,例如 ALL()函数、FILTER()、SUMMARIZE()等表函数的返回值,从而构成表函数的嵌套,第 2 个参数可以是各类复杂的逻辑表达式。

1. 基础应用

表函数的嵌套应用,ALL()用于扩展计算的范围、FILTER()用于减少计算的行数。通过 ALL()函数引用选择去重后的列(包装方式、入库),通过条件表达式来选择(由包装方式、入库二列构成的)新表中符合条件的行(包装方式为散装或箱装),表达式如下:

```
EVALUATE       //ch3-010
FILTER (
    ALL ( 'DK'[包装方式], 'DK'[入库] ),
    'DK'[包装方式] = "散装"||'DK'[包装方式] = "箱装"
)

-- 类似于微软 OLEDB 中的 SQL 语句 SELECT DISTINCT 包装方式,入库 FROM [DK$] WHERE
-- 包装方式 = "散装" OR 包装方式 = "箱装"
```

当 FILTER()函数的第 2 个参数的逻辑表达式为或关系(‖、OR)时,很多情况下也可

以采用 IN 的方式，表达式如下：

```
EVALUATE       //ch3 - 011
FILTER (
    ALL ( 'DK'[包装方式], 'DK'[入库] ),
    'DK'[包装方式] IN { "散装", "箱装" }       //IN 对大小写不敏感
)

-- 类似于微软 OLEDB 中的 SQL 语句 SELECT DISTINCT 包装方式,入库 FROM [DK $ ]WHERE 包装方式
-- IN ("箱装","散装")
```

如果将以上 IN 运算符换成 CONTAINSROW()函数，则对应的表达式如下：

```
EVALUATE      //ch3 - 012
    FILTER ( ALL ( 'DK'[包装方式], 'DK'[入库] ),
    CONTAINSROW (
        { "散装", "箱装" },
        'DK'[包装方式]
    )
)
```

包装方式	入库
箱装	32
散装	76
箱装	78

图 3-11 FILTER()与 ALL()的交互(1)

以上 3 个表达式返回的值是相同的，如图 3-11 所示。

综上，DAX 查询语句中的 FILTER()函数的第 2 个参数类似于 SQL 语句中的 WHERE 用法。本章不再一一列举对应的 SQL 语句。

2. 进阶应用

表函数的嵌套应用举例，以 FILTER()与 FILTER()的嵌套为例，表达式如下：

```
EVALUATE      //ch3 - 013
FILTER (
    FILTER ( ALL ( 'DK'[包装方式], 'DK'[入库] ), 'DK'[包装方式] = "散装" ),
    'DK'[入库] > 60
)
```

两个 FILTER()的嵌套，相当于条件的 &&，以上表达式的等效表达式用法如下：

```
EVALUATE      //ch3 - 014
FILTER (
    ALL ( 'DK'[包装方式], 'DK'[入库] ),
    'DK'[包装方式] = "散装" && 'DK'[入库] > 60
)
```

返回的值如图 3-12 所示。

以 FILTER()与 ALL()的嵌套为例,第 2 个参数采用 IN 逻辑运算符,表达式如下:

```
EVALUATE        //ch3 - 015
FILTER (
    ALL ( 'DK'[包装方式], 'DK'[入库], 'DK'[出库] ),
    'DK'[包装方式]IN { "散装", "箱装" }
&& 'DK'[出库] > 30
)
ORDER BY
    'DK'[包装方式],'DK'[入库] ASC,
    'DK'[出库] DESC
```

返回的值如图 3-13 所示。

包装方式	入库
散装	76

图 3-12 FILTER()与 ALL()的交互(2)

包装方式	入库	出库
散装	76	35
箱装	78	36

图 3-13 FILTER()与 ALL()的交互(3)

仅用于查询或创建表时,以下两个表达式所返回的值是相同的。如果将其放在迭代函数或者 CALCULATE()函数中,则 ALL()函数用于移除筛选,VALUES()函数用于保持外部筛选上下文。二者的返回值将会存在较大的差异,其具体的差异需放在度量值中结合外部筛选上下文才能真正体现出来(本章不进行深入说明)。查询应用,表达式如下。

查询一:

```
DEFINE MEASURE        //ch3 - 016
'DK'[M.入库量] = SUM ( 'DK'[入库] )
EVALUATE
VAR A = SUM ( DK[入库] ) RETURN
FILTER ( ALL ( DK[包装方式] ), [M.入库量] <= A )
```

查询二:

```
DEFINE MEASURE        //ch3 - 017
'DK'[M.入库量] = SUM ( 'DK'[入库] )
EVALUATE
VAR A = SUM ( DK[入库] ) RETURN
FILTER ( VALUES ( DK[包装方式] ), [M.入库量] <= A )
```

查询一与查询二所返回的值如图 3-14 所示。

图 3-14 FILTER()与 ALL()的交互(4)

3.3 表筛选

CALCULATETABLE()函数与 FILTER()函数返回的均是筛选后的表,但二者的计算逻辑是不一样的,CALCULATETABLE()函数先执行所有筛选参数,然后执行第 1 个参数,其相关语法如下:

```
CALCULATETABLE(
    <expression>[,
    <filter1> [, <filter2> [, …]]]
)
```

CALCULATETABLE()的筛选参数为用于定义筛选器(当存在多个筛选器时,可使用逻辑运算符 &&、||、IN 对它们进行计算)或筛选调节器函数(ALL()、ALLEXCEPT()、ALLNOBLANKROW()、KEEPFILTERS()、REMOVEFILTERS()、CROSSFILTER()、USERELATIONSHIP())的布尔表达式(值为 TRUE 或 FALSE)或表表达式。该函数的第 2 个参数的用法与 CALCULATE()的第 2 个参数的用法是一致的,其差异在于第 1 个参数。

在 CALCULATE()及 CALCULATETABLE()函数中,当 ALL()函数作为调节器使用时,其功能与 REMOVEFILTERS()函数是完全一致的。

3.3.1 筛选基础

当筛选条件的表达式为'表'[列]=固定值时,CALCULATETABLE ()函数与 FILTER()函数的函数返回值不存在差异,表达式如下:

```
EVALUATE        //ch3 - 018
CALCULATETABLE('DK','DK'[包装方式] = "箱装")
```

以上表达式等效于 FILTER()表达式的表达式如下:

```
EVALUATE        //ch3 - 019
FILTER('DK','DK'[包装方式] = "箱装")
```

返回的值如图 3-15 所示。

日期	产品	包装方式	入库	出库	方数	吨数	等级
2020/9/27	蛋糕纸	箱装	32	12	85	21.27	A
2021/10/11	苹果醋	箱装	78	36	65		B

图 3-15　筛选的条件为固定值

CALCULATE()及 CALCULATETABLE()函数的第 2 个参数只适用于为'表'[列]=固定值的情形。当存在<列>=[度量]、<列>=(公式)、<列>=<列>、[度量]=[度量]、[度量]=(公式)的情形时，第 2 个参数的筛选只能用 FILTER()函数的条件表达式，举例如下：

```
EVALUATE       //ch3 - 020
FILTER ('DK', 'DK'[入库]> = AVERAGE('DK'[入库]))
```

返回的值如图 3-16 所示。

日期	产品	包装方式	入库	出库	方数	吨数	等级
2020/9/2	钢化膜	散装	76	35	56	18.86	A
2021/9/29		捆	87	33		21.22	C
2021/9/29	净化剂	桶装	93	42	59	20.89	B
2021/10/11	苹果醋	箱装	78	36	65		B

图 3-16　筛选的条件为公式

如果将以上代码中的 FILTER()函数替换为 CALCULATETABLE ()函数，则表达式如下：

```
EVALUATE       //ch3 - 021
CALCULATETABLE('DK', 'DK'[入库]> = AVERAGE('DK'[入库]))
```

返回的错误提示如图 3-17 所示。

图 3-17　报错提示

对以上表达式进行修正，将第 2 个参数的直接表达式换成 FILTER()高级筛选，表达式如下：

```
EVALUATE       //ch3 - 022
CALCULATETABLE (
    'DK',
    FILTER ('DK','DK'[入库]> = AVERAGE ( 'DK'[入库] ))
)
```

显示正常，返回的值如图 3-16 所示。

3.3.2 筛选顺序

在 DAX 中，FILTER()函数为迭代函数，迭代函数产生行上下文，行上下文无法转换为筛选上下文，而 CALCULATE()和 CALCULATETABLE()这两个函数具备上下文转换的能力。FILTER()的行上文应用举例，表达式如下：

```
EVALUATE        //ch3 - 023
FILTER(
    ADDCOLUMNS (
    VALUES ( 'DK'[包装方式] ),
    "DK 数", COUNTROWS ( 'DK' )
    ),
    'DK'[包装方式] IN{"桶装","箱装"}
)
```

筛选的顺序及返回的值如图 3-18 所示。

图 3-18 FILTER()的应用

图 3-18 中，FILTER()函数的行上下文会遍历整个 DK 表，返回的值为 DK 表的总行数，返回的值为非期望的值。

对以上表达式进行修正，将以上表达式上的 FILTER()改为 CALCULATETABLE()，修改后的表达式如下：

```
EVALUATE       //ch3 - 024
CALCULATETABLE(
    ADDCOLUMNS (
    VALUES ( 'DK'[包装方式] ),
    "DK 数", COUNTROWS ( 'DK' )
    ),
    'DK'[包装方式] IN{"桶装","箱装"}
)
```

筛选的顺序及返回的值如图 3-19 所示。

CALCULATETABLE()函数运行时先筛选再运算，FILTER()函数运行时先运算再筛

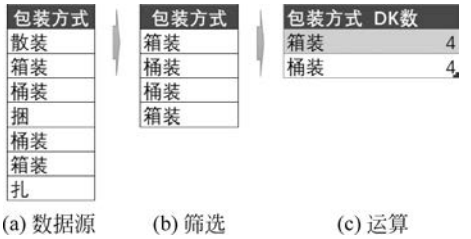


图 3-19　CALCULATETABLE()的应用

选。这是二者运算顺序的区别。图 3-19 中,CALCULATETABLE()函数返回的值为 DK 表中桶装、箱装数据的总行数,返回的值仍为非期望的值。

此时可以对以上表达式进行修正,在 COUNTROWS ('DK')外面套上 CALCULATE(),采用 FILTER()函数,表达式如下:

```
EVALUATE        //ch3 - 025
FILTER(
    ADDCOLUMNS (
    VALUES ( 'DK'[包装方式] ),
    "DK 数", CALCULATE(COUNTROWS ( 'DK' ))
    ),
    'DK'[包装方式] in{"桶装","箱装"}
)
```

或者将以上表达式中的 FILTER()改为 CALCULATETABLE (),修改后的表达式如下:

```
EVALUATE        //ch3 - 026
CALCULATETABLE(
    ADDCOLUMNS (
    VALUES ( 'DK'[包装方式] ),
    "DK 数", CALCULATE(COUNTROWS ( 'DK' ))
    ),
    'DK'[包装方式] in {"桶装","箱装"}
)
```

以上两个表达式的返回值相同,如图 3-20 所示。

在 CALCULATETABLE()中,可在第 2 个参数中使用调节器函数 ALL()移除筛选,表达式如下:

```
EVALUATE
CALCULATETABLE (
    VALUES ( DK[产品] ),                    //受筛选影响的值
    ALL ( DT[日期] )                        //移除筛选,扩大上下文的范围
)
```

以上表达式中 ALL()函数的数据可来自事实表,也可来自查询表,一切视表达式的需要,返回的值如图 3-21 所示。

包装方式	DM数
箱装	2
桶装	2

图 3-20 上下文的转换应用

产品
钢化膜
蛋糕纸
油漆
净化剂
苹果醋
包装绳

图 3-21 上下文的转换应用

3.3.3 差异比较

以下是 FILTER()与 CALCULATETABLE()的一些其他差异比较。

创建查询表,表达式如下:

```
EVALUATE        //ch3 - 027
FILTER(
    ALL ( 'DK'[产品] ),
    'DK'[包装方式] IN {"桶装","箱装"}
)
```

返回的错误提示如图 3-22 所示。

图 3-22 错误提示

修改上面的表达式,将 FILTER()修改为 CALCULATETABLE(),修改后的表达式如下:

```
EVALUATE       //ch3 - 028
CALCULATETABLE(
    ALL ( 'DK'[产品] ),
    'DK'[包装方式] IN {"桶装","箱装"}
)
```

筛选的原理及返回的值如图 3-23 所示。

图 3-23 中存在空值,且未做任何的筛选。在上面的表达式中,ALL()是 CALCULATETABLE()函数的调节器函数,用于移除筛选。将以上表达式中的 ALL()修改为 DISTINCT(),修改

产品	包装方式
钢化膜	散装
蛋糕纸	箱装
油漆	桶装
	捆
净化剂	桶装
苹果醋	箱装
包装绳	扎

(a) 数据源

产品
钢化膜
蛋糕纸
油漆
净化剂
苹果醋
包装绳

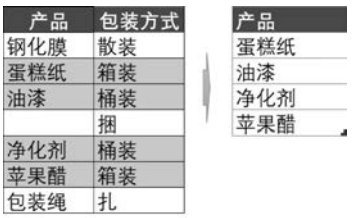
(b) 返回的值

图 3-23 筛选返回值(1)

后的表达式如下：

```
EVALUATE      //ch3 - 029
CALCULATETABLE(
    DISTINCT ( 'DK'[产品] ),
    'DK'[包装方式] IN {"桶装","箱装"}
)
```

筛选的原理及返回的值如图 3-24 所示。



产品	包装方式
钢化膜	散装
蛋糕纸	箱装
油漆	桶装
	捆
净化剂	桶装
苹果醋	箱装
包装绳	扎

(a) 数据源

产品
蛋糕纸
油漆
净化剂
苹果醋

(b) 返回的值

图 3-24 筛选返回值(2)

ALL()、VALUES()、DISTINCT()的比较说明：

(1) ALL()函数用于移除筛选，VALUES()及 DISTINCT()函数用于保持外部筛选上下文并返回在当前筛选器中计算列的不同值，ALL()与 VALUES()在筛选的功能上是相反的。

(2) ALL()函数、VALUES()函数视空行为有效行，并将其显示出来，这一点与 DISTINCT()功能存在差异。DISTINCT()会清除空行，其他方面与 VALUES()功能大体一致。

3.4 集合与事件

常见的集合与事件的关系有并集(UNION)、交集(INTERSECT)、差集(EXCEPT)。其连接的两个数据可来自同一数据源，也可以来自不同的数据源，其返回的数据如图 3-25 所示。

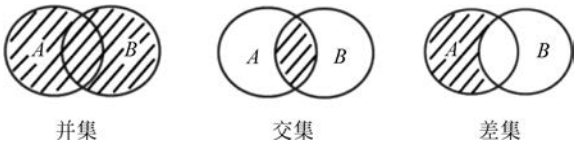


图 3-25　数据的集合

在交集与差集中，如果集合 A 与集合 B 在函数参数中的位置不同，则其返回的值也将不同。

3.4.1　UNION()

UNION()函数用于从一对表创建联合(连接)表，语法如下：

```
UNION(< table_expression1 >, < table_expression2 >[,< table_expression >] … )
```

连接的两个数据来自同一数据源，应用举例，表达式如下：

```
EVALUATE       //ch3 - 030
VAR T1 =
    FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] > 75 )
VAR T2 =
    FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] < 50 )
RETURN
UNION ( T1, T2 )
ORDER BY [入库] DESC
```

运算过程及返回的值如图 3-26 所示。

产品	入库
净化剂	93
	87
苹果醋	78
钢化膜	76
包装绳	65
油漆	55
蛋糕纸	32

(a) 数据源

产品	入库
净化剂	93
	87
苹果醋	78
钢化膜	76

蛋糕纸	32

(b) 集合T1与T2

产品	入库
净化剂	93
	87
苹果醋	78
钢化膜	76
蛋糕纸	32

(c) UNION(T1,T2)

图 3-26　UNION()函数的应用(1)

连接的两个数据来自同一数据模型的不同表格，应用举例，表达式如下：

```
EVALUATE       //ch3 - 031
DISTINCT(      //删除列中重复项且去除空值
    UNION(
    VALUES('合同'[产品]), VALUES('订单'[产品])       //此处的 VALUES 可用 DISTINCT 替换
    )
)//表名的单引号''不能少.
```

返回的值如图 3-27 所示。

产品
保鲜剂
老陈醋
劳保手套
木材
钢材
异型件
钢化膜
油漆
尿素
净化剂
蛋糕纸
苹果醋
包装绳

图 3-27 UNION()函数的应用(2)

3.4.2 INTERSECT()

INTERSECT()函数用于返回两个表的行交集,保留重复项,语法如下:

```
INTERSECT(< table_expression1 >, < table_expression2 >)
```

注意:INTERSECT()函数保留的是左表数据沿袭。INTERSECT(T1, T2)与 INTERSECT (T2, T1)返回的值很多情况下存在差异。

连接的两个数据来自同一数据源,应用举例,表达式如下:

```
EVALUATE        //ch3 - 032
VAR T1 =
    FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] > 60 )
VAR T2 =
    FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] < 80 )
RETURN
    INTERSECT( T1, T2 )
ORDER BY [入库] DESC
```

运算过程及返回的值如图 3-28 所示。

产品	入库
净化剂	93
	87
苹果醋	78
钢化膜	76
包装绳	65
油漆	55
蛋糕纸	32

(a) 数据源

产品	入库
净化剂	93
	87
苹果醋	78
钢化膜	76
包装绳	65

苹果醋	78
钢化膜	76
包装绳	65
油漆	55
蛋糕纸	32

(b) 集合T1与T2

产品	入库
苹果醋	78
钢化膜	76
包装绳	65

(c) INTERSECT(T1,T2)

图 3-28 INTERSECT()函数的应用(1)

连接的两个数据来自同一数据模型的不同表格,应用举例,表达式如下:

```
EVALUATE        //ch3 - 033
DISTINCT( INTERSECT(VALUES('合同'[产品]), VALUES('订单'[产品]) ))
```

产品
保鲜剂
老陈醋
木材
钢材
异型件
钢化膜
油漆
蛋糕纸
苹果醋
包装绳

图 3-29 INTERSECT()函数的应用(2)

返回的值如图 3-29 所示。

3.4.3 EXCEPT()

EXCEPT()函数用于返回一个表的行。这些行出现在第 1 个表,但未出现在第 2 个表,语法如下:

```
EXCEPT(< table_expression1 >, < table_expression2 >
```

EXCEPT()函数保留的是左表数据的沿袭。在 EXCEPT()中,对第 1 个和第 2 个参数的表进行位置交换后将返回不同的值。

连接的两个数据来自同一数据源,应用举例,表达式如下:

```
EVALUATE        //ch3 - 034
VAR T1  =
    FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] > 60 )
VAR T2  =
    FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] < 80 )
RETURN
    EXCEPT( T1, T2 )
ORDER BY [入库] DESC
```

运算过程及返回的值如图 3-30 所示。

产品	入库
净化剂	93
	87
苹果醋	78
钢化膜	76
包装绳	65
油漆	55
蛋糕纸	32

(a) 数据源

产品	入库
净化剂	93
	87
苹果醋	78
钢化膜	76
包装绳	65

苹果醋	78
钢化膜	76
包装绳	65
油漆	55
蛋糕纸	32

(b) 集合T1与T2

产品	入库
净化剂	93
	87

(c) EXCEPT(T1,T2)

图 3-30 EXCEPT()函数的应用(1)

连接的两个数据来自同一数据模型的不同表格,应用举例,表达式如下:

```
EVALUATE        //ch3 - 035
DISTINCT( EXCEPT(     VALUES('合同'[产品]), VALUES('订单'[产品]) ))
```

返回的值如图 3-31 所示。

产品
劳保手套
尿素
净化剂

图 3-31　EXCEPT()函数的应用(2)

3.5　笛卡儿积

3.5.1　CROSSJOIN()

CROSSJOIN()函数用于返回一个表,其中包含参数中所有表的所有行的笛卡儿积。新表中的列是所有参数表中的所有列,语法如下:

```
CROSSJOIN(<table>, <table>[, <table>]…)
```

应用举例,表达式如下:

```
EVALUATE       //ch3-036
VAR T1 = { "箱装", "桶装" }
VAR T2 = { ( 93, 87, 78 ), ( 65, 55, 32 ) }
RETURN
    CROSSJOIN ( T1, T2 )
```

返回的值如图 3-32 所示。

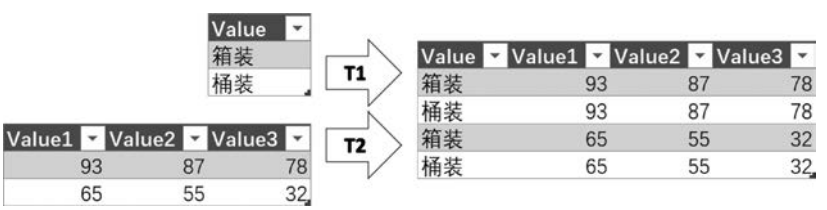


图 3-32　CROSSJOIN()函数的应用

3.5.2　GENERATE()

GENERATE()函数用于返回一个表,其中包含 table1 中的每行与在 table1 的当前行的上下文中计算 table2 所得表之间的笛卡儿乘积,语法如下:

```
GENERATE(<table1>, <table2>)
```

应用举例，表达式如下：

```
EVALUATE      //ch3 - 037
GENERATE (
FILTER(VALUES('DK'[包装方式]), 'DK'[包装方式] IN { "散装", "箱装" } ) ,
FILTER(ALL('DK'[产品],'DK'[入库]), 'DK'[入库] > 80 ) )
```

返回的值如图 3-33 所示。

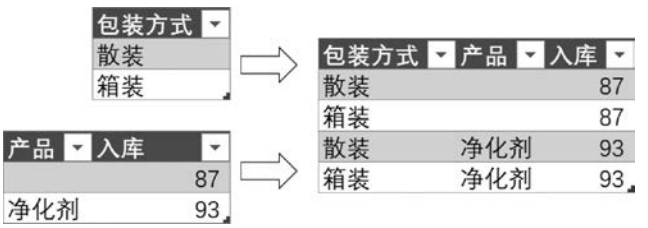


图 3-33　GENERATE()函数的应用(1)

注意：table1 和 table2 中的所有的列名不得相同，否则会返回错误。

将以上表达式进行修正，使两个表之间的产品列出现重名，表达式如下：

```
EVALUATE      //ch3 - 038
GENERATE (
FILTER(VALUES('DK'[产品]), 'DK'[包装方式] IN { "净化剂", "钢化膜" } ) ,
FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] > 80 ) )
```

系统报错提示如图 3-34 所示。

图 3-34　GENERATE()函数的应用(2)

3.5.3　GENERATEALL()

GENERATEALL()函数用于返回一个表，其中包含 table1 中的每行与在 table1 的当前行的上下文中计算 table2 所得表之间的笛卡儿乘积，语法如下：

```
GENERATEALL(< table1 >, < table2 >)
```

应用举例，表达式如下：

```
EVALUATE       //ch3 - 039
GENERATEALL (
FILTER(VALUES('DK'[包装方式]), 'DK'[包装方式] IN { "散装", "箱装" } ) ,
FILTER(ALL('DK'[产品],'DK'[入库]), 'DK'[入库] > 80 ) )
```

返回的值如图 3-35 所示。

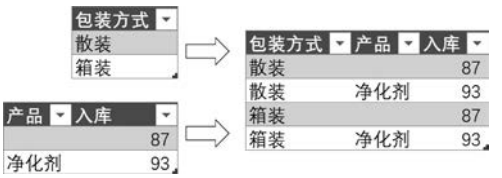


图 3-35　GENERATEALL()函数的应用

对比图 3-34 及图 3-35 后可发现，GENERATE()与 GENERATEALL()函数的区别在于数据的组合方式，其差异对比说明如图 3-36 所示。

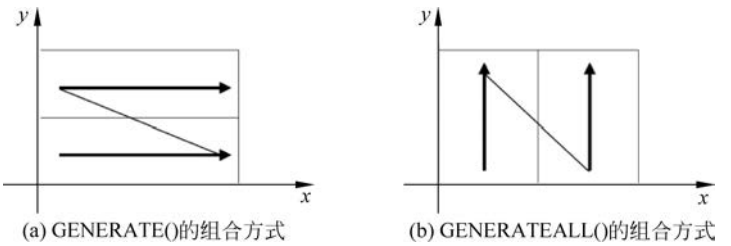


图 3-36　GENERATE()与 GENERATEALL()的对比说明

3.6　连接查询

3.6.1　NATURALINNERJOIN()

NATURALINNERJOIN()函数用于执行一个表与另一个表的内部连接。这些表在两个表的共有列(按名称)上连接。如果两个表没有公共列名，则返回错误，语法如下：

```
NATURALINNERJOIN(< leftJoinTable >, < rightJoinTable >)
```

NATURALINNERJOIN()函数类似于 SQL 中的 INNER JOIN。应用举例，表达式如下：

```
EVALUATE       //ch3 - 040
VAR T1 =
    FILTER (
```

```
        ALL('DK'[产品], 'DK'[包装方式],'DK'[入库]),
        'DK'[入库] > 70
    )
VAR T2 =
    FILTER (
        ALL('DK'[产品],'DK'[入库]),
        'DK'[入库] < 90
    )
RETURN
    NATURALINNERJOIN(T1,T2)
```

返回的值如图 3-37 所示。

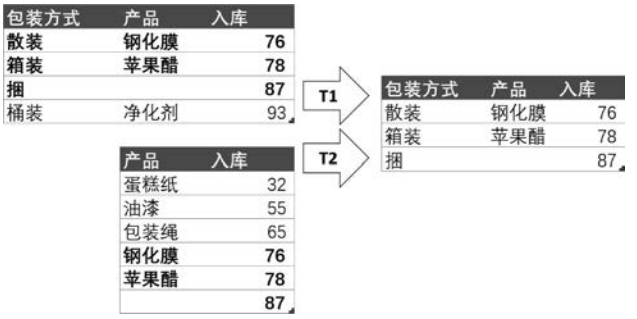


图 3-37 NATURALINNERJOIN()函数的应用

3.6.2 NATURALLEFTOUTERJOIN()

NATURALLEFTOUTERJOIN()用于执行一个表与另一个表的内部连接。这些表在两个表的共有列(按名称)上连接。如果两个表没有公共列名,则返回错误,语法如下:

```
NATURALLEFTOUTERJOIN(<leftJoinTable>, <rightJoinTable>)
```

NATURALLEFTOUTERJOIN()函数类似于 SQL 中的 LEFT JOIN。应用举例,表达式如下:

```
EVALUATE        //ch3 - 041
VAR T1 =
    FILTER ( ALL('DK'[产品], 'DK'[包装方式],'DK'[入库]), 'DK'[入库] > 70 )
VAR T2 =
    FILTER ( ALL('DK'[产品],'DK'[入库]), 'DK'[入库] < 90 )
RETURN
NATURALLEFTOUTERJOIN( T1, T2 )
```

返回的值如图 3-38 所示。

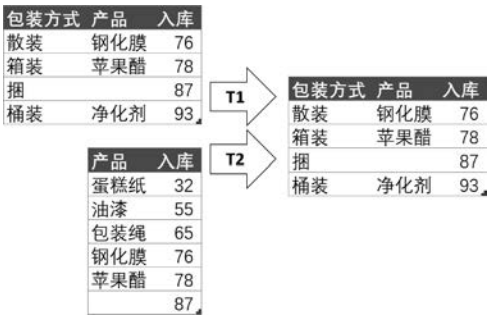


图 3-38 NATURALLEFTOUTERJOIN()函数的应用

3.7 分组

DAX 中的 SUMMARIZE()类似于 SQL 中的 GROUP BY 分组操作。在 SQL 中，WITH ROLLUP 通常与 GROUP BY 子句一起使用，根据维度在分组后进行聚合操作。同理，在 DAX 中 ROLLUP()函数只能在 SUMMARIZE()表达式中一起使用。

3.7.1 SUMMARIZE()

SUMMARIZE()函数用于生成数据汇总表，语法如下：

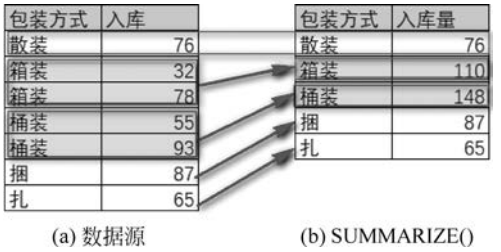
```
SUMMARIZE (
    <table>,
    <groupBy_columnName>
    [, <groupBy_columnName>]…[, <name>, <expression>]…
)
```

以 DK 表中的包装方式列为分组依据，对入库列的数据进行分组统计，表达式如下：

```
EVALUATE        //ch3-042
SUMMARIZE(
    'DK',
    'DK'[包装方式],
    "入库量",SUM('DK'[入库])
)
```

返回的值如图 3-39 所示。

以 DK 表中的包装方式列为分组依据，对入库列的数据进行分组统计，并筛选出统计值(入库量)大于 80 的数据，表达式如下：



包装方式	入库
散装	76
箱装	32
箱装	78
桶装	55
桶装	93
捆	87
扎	65

包装方式	入库量
散装	76
箱装	110
桶装	148
捆	87
扎	65

(a) 数据源　　(b) SUMMARIZE()

图 3-39　SUMMARIZE()函数的应用(1)

```
EVALUATE        //ch3 - 043
FILTER (
    SUMMARIZE (
        'DK',
        'DK'[包装方式],
        "入库量", SUM ( 'DK'[入库] )
    ),
    [入库量] > 80
)
```

返回的值如图 3-40 所示。

筛选 DK 表中包装方式为箱装和桶装的数据,然后以包装方式列为分组依据,对入库列的数据进行分组统计,表达式如下:

```
EVALUATE        //ch3 - 044
SUMMARIZE (
    FILTER (
        'DK',
        'DK'[包装方式] IN {"箱装","桶装"}
    ),
    'DK'[包装方式],
    "入库量", SUM ( 'DK'[入库] )
)
```

返回的值如图 3-41 所示。

包装方式	入库量
箱装	110
桶装	148
捆	87

图 3-40　SUMMARIZE()函数的应用(2)

包装方式	入库量
箱装	110
桶装	148

图 3-41　SUMMARIZE()函数的应用(3)

注意:SUMMARIZE()是个慢函数。如果在 SUMMARIZE()函数的使用过程中需要产生派生列,则建议在外围嵌套 ADDCOLUMNS()函数,不要直接使用 SUMMARIZE()进

行分组聚合，或者优先采用 SUMMARIZECOLUMNS()函数。除非需要在一个或多个分组列上使用 ROLLUP()计算每组的总计。只有派生列使用了某些非常特殊的表达式时才可以考虑直接使用 SUMMARIZE()的分组聚合功能。

3.7.2 SUMMARIZECOLUMNS()

SUMMARIZECOLUMNS()函数用于返回由一组列指定的摘要表，语法如下：

```
SUMMARIZECOLUMNS(
    <groupBy_columnName>
    [, <groupBy_columnName>]…,
    [<filterTable>]…
    [,<name>,<expression>]…
)
```

1. 数据来自同一个表

以包装方式列为分组依据，对入库列的数据进行分组统计，表达式如下：

```
EVALUATE        //ch3-045
SUMMARIZECOLUMNS (
    'DK'[包装方式],
    "入库量", SUM ( 'DK'[入库] ),
    "次数", COUNT ( DK[入库] )
)
```

返回的值如图 3-42 所示。

以包装方式列为分组依据，对入库列的数据进行求平均值，表达式如下：

```
EVALUATE        //ch3-046
SUMMARIZECOLUMNS(
    'DK'[包装方式],
    "入库均量", AVERAGE('DK'[入库])
)
```

返回的值如图 3-43 所示。

包装方式	入库量	次数
散装	76	1
箱装	110	2
桶装	148	2
捆	87	1
扎	65	1

图 3-42 SUMMARIZECOLUMNS()函数的应用(1)

包装方式	入库均量
散装	76
箱装	55
桶装	74
捆	87
扎	65

图 3-43 SUMMARIZECOLUMNS()函数的应用(2)

以 DK 表中的包装方式为分组依据，筛选表中产品为蛋糕纸、净化剂、苹果醋的行，然后对入库列的数据进行分组统计，表达式如下：

```
EVALUATE        //ch3 - 047
SUMMARIZECOLUMNS (
    'DK'[包装方式],
    FILTER ( 'DK', 'DK'[产品] IN {"蛋糕纸", "净化剂", "苹果醋"} ),
    "入库量", SUM ( 'DK'[入库] )
)
```

返回的值如图 3-44 所示。

包装方式	入库量
箱装	110
桶装	93

图 3-44 SUMMARIZECOLUMNS()函数的应用(3)

2. 数据来自数据模型

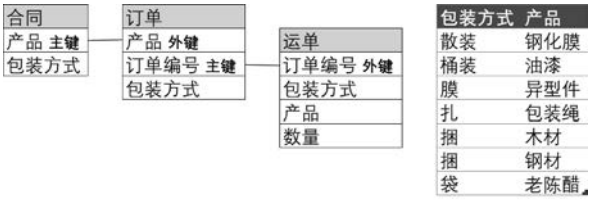
以合同表中的包装方式和运单表中的产品列为组合依据，筛选运单表中数量大于 3 的数据，表达式如下：

```
EVALUATE        //ch3 - 048
SUMMARIZECOLUMNS(
'合同'[包装方式],
'运单'[产品],
FILTER( '运单', '运单'[数量]> 3 ) )
```

等同于以上代码，表达式如下：

```
EVALUATE        //ch3 - 049
SUMMARIZECOLUMNS(
'合同'[包装方式],
'运单'[产品],
CALCULATETABLE( '运单', '运单'[数量]> 3 ) )
```

返回的值如图 3-45 所示。



(a) 数据模型

包装方式	产品
散装	钢化膜
桶装	油漆
膜	异型件
扎	包装绳
捆	木材
捆	钢材
袋	老陈醋

(b) 查询返回的值

图 3-45 SUMMARIZECOLUMNS()函数的应用(4)

以合同表中的包装方式和运单表中的产品为组合，筛选运单表中数量大于 3 的行，对筛选后的数据进行求和、求平均，表达式如下：

```
EVALUATE        //ch3 - 050
SUMMARIZECOLUMNS(
    '合同'[包装方式],
    '运单'[产品],
    CALCULATETABLE( '运单', '运单'[数量]> 3),
    "数量和", SUM('运单'[数量]),
    "均成本", AVERAGE('运单'[成本])
    )
```

返回的值如图 3-46 所示。

包装方式	产品	数量和	均成本
散装	钢化膜	27	5
桶装	油漆	35	9
膜	异型件	18	10
扎	包装绳	22	2
捆	木材	16	6
捆	钢材	8	6
袋	老陈醋	9	8

图 3-46 SUMMARIZECOLUMNS()函数的应用(5)

3.7.3 ROLLUP()

ROLLUP()用于标识 SUMMARIZE()函数中需要计算小计的列。此函数只能在 SUMMARIZE()表达式中使用,语法如下:

```
ROLLUP (
    < groupBy_columnName > [,
    < groupBy_columnName > [, … ] ]
)
```

ROLLUP()函数只在 SUMMARIZE()中使用,并且 ROLLUP()表达式中引用的列不能作为 SUMMARIZE()函数的分组列,表达式如下:

```
EVALUATE        //ch3 - 051
SUMMARIZE(
    '运单',
    ROLLUP ( '合同'[包装方式] ),
    "数量和", SUM ( '运单'[数量] )
)
ORDER BY '合同'[包装方式]
```

在数据分析过程中,数据的小计、汇总等属于上卷(ROLLUP)操作,是数据的颗粒度由细到粗的过程,操作中会忽略某些维度,返回的值如图 3-47 所示。

如图 3-47 所示,数量和列中,空行列对应的值 154 是对列中其他值的总计。与上卷操

包装方式	数量和
	154
袋	11
捆	24
膜	20
散装	30
桶装	37
箱装	7
扎	25

ROLLUP('合同'[包装方式])

图 3-47 ROLLUP()函数的应用(1)

作对应的是下钻(或称“钻取”)操作。数据的颗粒度由粗到细的过程则属于数据的下钻,操作中会细化某些维度。

分类汇总数据并对各包装的数量进行小计,然后对所有产品数量进行统计,表达式如下:

```
EVALUATE        //ch3 - 052
SUMMARIZE (
'运单',
ROLLUP ( '合同'[包装方式], '合同'[产品]),
"数量和", SUM ( '运单'[数量] )
)
```

返回的值如图 3-48 所示。

产品	包装方式	数量和
蛋糕纸	箱装	4
苹果醋	箱装	3
钢化膜	散装	30
油漆	桶装	37
异型件	膜	20
包装绳	扎	25
保鲜剂	袋	2
木材	捆	16
钢材	捆	8
老陈醋	袋	9
	箱装	7
	散装	30
	桶装	37
	膜	20
	扎	25
	袋	11
	捆	24
		154

SUMMARIZE()

ROLLUP()小计

ROLLUP()总计

图 3-48 ROLLUP()函数的应用(2)

3.7.4 ROLLUPISSUBTOTAL()

ROLLUPISSUBTOTAL()不返回值,它只标记 ADDMISSINGITEMS()中要计算小计的列集,该函数只能在 ADDMISSINGITEMS()中使用,语法如下:

```
ROLLUPISSUBTOTAL (
    [< grandTotalFilter >],
    < groupBy_columnName >,
    < isSubtotal_columnName >
    [, [< groupLevelFilter >]
        [, < groupBy_columnName >,< isSubtotal_columnName >
        [, [< groupLevelFilter >] [, … ] ] ] ]
)
```

ROLLUPISSUBTOTAL()函数的参数说明,见表 3-2。

表 3-2 参数说明

参 数	参 数 说 明
grandtotalFilter	(可选) 要应用于总计级别的过滤器
groupBy_columnName	用于根据在其中找到的值创建汇总组的现有列的名称,不能是表达式
isSubtotal_columnName	ISSUBTOTAL 列的名称,列的值使用 ISSUBTOTAL 函数计算
groupLevelFilter	(可选) 要应用于当前级别的过滤器

相关应用举例,参见 ADDKISSINGITEMS()函数章节。

3.7.5 ROLLUPADDISSUBTOTAL()

ROLLUPADDISSUBTOTAL()函数用于标识 SUMMARIZECOLUMNS()函数中需要计算小计的列,返回的值为 TRUE 或 FALSE,语法如下:

```
ROLLUPADDISSUBTOTAL (
    [< grandtotalFilter >],
    < groupBy_columnName >,
    < name >
    [, [< groupLevelFilter >]
        [, < groupBy_columnName >, < name >
        [, [< groupLevelFilter >] [, … ] ] ] ]
)
```

ROLLUPADDISSUBTOTAL()函数的参数说明,见表 3-3。

表 3-3 参数说明

参 数	参 数 说 明
grandtotalFilter	(可选) 要应用于总计级别的筛选器
groupBy_columnName	用于根据列中的值创建摘要组的现有列的名称,不能是表达式
name	ISSUBTOTAL 列的名称,使用 ISSUBTOTAL 函数计算列的值
groupLevelFilter	(可选) 要应用于当前级别的筛选器

SUMMARIZECOLUMNS()中嵌套 ROLLUPADDISSUBTOTAL(),其功能类似于SUMMARIZE()中嵌套 ROLLUP(),用于小计行的标识。

对运单表中的产品、数量列和日期表中的年进行组合,然后标识需要小计的行,应用举例,表达式如下:

```
EVALUATE        //ch3 - 053
TOPN (
    5,
    FILTER (
        SUMMARIZECOLUMNS (
            '运单'[产品],
            '运单'[数量],
            ROLLUPADDISSUBTOTAL ('日期'[年],"年份")
        ),
        [年份] = TRUE
    )
)
```

产品	数量	年	年份
钢化膜	1		TRUE
蛋糕纸	2		TRUE
钢化膜	2		TRUE
油漆	2		TRUE
异型件	2		TRUE

图 3-49 ROLLUPISSUBTOTAL()函数的应用

返回的值共 96 行,(截取前 5 行)需要小计的行已被标识为 TRUE,不需要小计的行被标识为 FALSE,如图 3-49 所示。

图 3-48 中,年所在列的空值部分,其年份对应的标识均为 TRUE,该行是小计行。

3.7.6 ADDMISSINGITEMS()

ADDMISSINGITEMS()用于将具有空值的行添加到 SUMMARIZECOLUMNS()返回的表中,语法如下:

```
ADDMISSINGITEMS (
    [<showAll_columnName>
    [, <showAll_columnName>
    [, … ] ] ],
    <table>
    [,<groupBy_columnName>
    [, [<filterTable>]
    [, <groupBy_columnName>[, [<filterTable>] [, … ] ] ] ] ] ]
)
```

ADDMISSINGITEMS()函数的参数说明见表 3-4。

表 3-4 ADDMISSINGITEMS()函数的参数说明

参 数	参 数 说 明
showAll_columnName	(可选)要为其返回未使用度量值数据的项的列,如果未指定,则返回所有列
table	SUMMARIZECOLUMNS 表
groupBy_columnName	(可选)用于在提供的 table 参数中分组的列
filterTable	(可选)定义返回哪些行的表表达式

该函数用于添加由于新列的表达式返回空值而被 SUMMARIZECOLUMNS()隐藏的行,表达式如下:

```
EVALUATE       //ch3 - 054
ADDMISSINGITEMS (
    '运单'[包装方式] ,
    SUMMARIZECOLUMNS(
        '运单'[包装方式] ,
        "求平均值", AVERAGE('运单'[超额费用] )
    ),
    '运单'[包装方式]
)
```

返回的值如图 3-50 所示。

继续举例 ADDMISSINGITEMS()的应用,表达式如下:

```
EVALUATE       //ch3 - 055
ADDMISSINGITEMS (
    '运单'[包装方式],
    SUMMARIZECOLUMNS (
        '运单'[包装方式],
        "数量和", CALCULATE (
                SUM ( '运单'[数量] ),
                    FILTER ( ALL ( '运单'[数量] ), '运单'[数量] <= 5 )
                )
    ),
    '运单'[包装方式]
)
```

返回的值如图 3-51 所示。

包装方式	求平均值
箱装	1
散装	0.533333333
桶装	2
膜	1.3
扎	0.2
袋	
捆	

图 3-50 添加具有空值的行(1)

包装方式	数量和
箱装	7
散装	7
桶装	7
膜	6
扎	12
袋	2
捆	

图 3-51 添加具有空值的行(2)

3.8 创建表

3.8.1 表构造器{()}

在DAX中表构造器由{}和()组成。{}代表的是一列的数据,()代表的是一行的数据。如果表的数据只有一列,则小括号可以省略,应用举例,表达式如下:

```
EVALUATE        //ch3-056
{ "散装", "箱装", "桶装" }          //标题为"Value"的一列数据,共3行
```

列表内每个括号代表的是一行,当只有一列时,括号可以省略,表达式如下:

```
EVALUATE        //ch3-057
{ ("散装"), ("箱装"), ("桶装") }   //标题为"Value"的一列数据
```

返回的值如图3-52所示。

在表构造器内,{}代表的是列值,()代表的是行值,“,”代表的是数据分隔符。如果需要创建的是一个多列的表,则每行的数据需用小括号分隔,数据与数据间用逗号分隔。以创建一个三行四列的表为例,表达式如下:

```
EVALUATE        //ch3-058
{
    ("箱装","蛋糕纸",2,15),        //()内","为列分隔符
    ("散装","钢化膜",3,5),
    ("桶装","油漆",5,9)
}
```

返回的值如图3-53所示。

Value
散装
箱装
桶装

图3-52 创建单列的表

Value1	Value2	Value3	Value4
箱装	蛋糕纸	2	15
散装	钢化膜	3	5
桶装	油漆	5	9

图3-53 创建3行4列的表

3.8.2 DATATABLE()

DATATABLE()函数用于快速创建结构简单的表,支持的数据类型有整数(INTEGER)、双精度(DOUBLE)、字符串(STRING)、布尔值(BOOLEAN)、货币(CURRENCY)、日期时间(DATETIME),语法如下:

```
DATATABLE (
    ColumnName1, DataType1,
    ColumnName2, DataType2...,
    {
        {Value1, Value2...},
        {ValueN,ValueN + 1...}...
      }
)
```

DATATABLE()表的内容必须是常量，不支持任何 DAX 表达式，应用举例，表达式如下：

```
EVALUATE       //ch3 - 059
DATATABLE (
    "产品", string,
    "包装方式", string,
    "入库",integer,
    {
        {"钢化膜", "散装", 76},
        {"蛋糕纸", "箱装", 32}
    }
)
```

在 DAX 中主要有 6 种数据类型：BOOLEAN、STRING、DOUBLE、INTEGER、CURRENCY、DATETIME。DAX 中，对函数、变量名、文件名的字母大小写不敏感。以上表达式返回的值如图 3-54 所示。

产品	包装方式	入库
钢化膜	散装	76
蛋糕纸	箱装	32

图 3-54　创建简单的表

DATATABLE()函数多见于参数表的创建过程中，应用举例，表达式如下：

```
EVALUATE       //ch3 - 060
DATATABLE (
    "区间", STRING,
    "起始值", CURRENCY,
    "结束值", CURRENCY,
    {
        { "差", 0, 59 },
        { "中", 60, 79 },
        { "良", 80, 89 },
        { "优", 90, 100 }
    }
)
```

返回的值如图 3-55 所示。

3.8.3 ROW()

ROW()函数用于返回一个具有单行的表，其中包含针对每列计算表达式得出的值，语法如下：

```
ROW(< name >, < expression >[[,< name >, < expression >]… ])
```

第 1 个参数 name 为指定的列名，必须用双引号引起来。第 2 个参数为 DAX 表达式。创建 DAX 查询，表达式如下：

```
EVALUATE        //ch3 - 061
ROW("数量和",SUM('运单'[数量]),"成本和",SUM('运单'[成本]))
```

返回的值如图 3-56 所示。

区间	起始值	结束值
差	0	59
中	60	79
良	80	89
优	90	100

图 3-55 创建的参数表

数量和	成本和
154	222

图 3-56 ROW 函数所返回的表

3.9 综合应用

3.9.1 TOPN()

TOPN()函数用于获取指定表中的前 *N* 行数据，语法如下：

```
TOPN(
    < n_value >,
    < table >,
    < orderBy_expression >,
    [< order >[,< orderBy_expression >, [< order >]]… ]
)
```

第 4 个参数 order 为可选参数，用于指定第 3 个参数(orderBy_expression)值的排序方式(默认值为 0，降序；升序为 1)。应用举例，选择运单表中的产品和数量列，并对各产品的数量求和，挑选表中数量和排名前 3 的数据，表达式如下：

```
EVALUATE        //ch3 - 062
TOPN (
    3,
```

```
    SUMMARIZECOLUMNS (
        '运单'[产品],
        "数量和",SUM('运单'[数量])
    ),
    [数量和]
)
```

返回的值如图 3-57 所示。

产品	数量和
包装绳	25
油漆	37
钢化膜	30

图 3-57　创建的表

3.9.2　扩展表理论

表的扩展是从关系的多端向一端进行的。扩展表包含关系多端中的所有列(原生列),以及处于关系一端中的所有的表和列(相关列)。以 SUMMARIZE()应用为例,扩展来自数据多端的表和来自数据一端的列,表达式如下:

```
EVALUATE        //ch3 - 063
SUMMARIZE('运单','合同'[产品])
```

扩展表只会由多端向一端扩展,扩展后的表包含一端的所有列,数据模型及返回的值如图 3-58 所示。

(a) 数据模型　　(b) 查询返回的值

图 3-58　创建分组列

在数据模型中,表的扩展是不能从一端向多端进行的。对以上表达式进行修改,修改后的表达式如下:

```
EVALUATE        //ch3 - 064
SUMMARIZE('合同','运单'[产品])
```

因在合同表的扩展表中找不到产品列,所以系统报错提示,如图 3-59 所示。

扩展表综合应用,在数据模型多端的运单表中获取产品列,对一端的订单表中的产品列进行计数,对一端的装货表中的包装方式列进行计数和统计,对一端的装货表中的质量列进行求和统计,对一端的收货表中的收货人进行去重统计,表达式如下:

图 3-59 返回的错误提示

```
EVALUATE        //ch3 - 065
SUMMARIZE (
    '运单',
    '运单'[产品],
    "产品数", COUNT ( '订单'[产品] ),
    "包装数", COUNT ( '装货'[包装方式] ),
    "装货数", SUM ( '装货'[质量] ),
    "非重复收货人", DISTINCTCOUNT ( '收货'[收货人] )
)
```

返回的值如图 3-60 所示。

产品	产品数	包装数	装货数	非重复收货人
蛋糕纸	2	1	20	1
苹果醋	1	1	12	1
钢化膜	6	1	150	3
油漆	7	1	10	3
异型件	4	1	2	2
包装绳	5	1	1	2
保鲜剂	1	1	250	1
木材	2	1	2	2
钢材	1	1	1.5	1
老陈醋	1	1	150	1

图 3-60 扩展表返回的值

注意：*在多对多关系中不存在表的扩展。*

3.9.3 创建度量表

在日常的数据处理与分析过程中，经常会创建大量的度量值，此时对度量值的管理也将成为工作效率提升的一部分。当模型中新建的度量值特别多的情况下，新增一个空表用于专门收纳度量值是非常有必要的。应用举例，表达式如下：

```
EVALUATE        //ch3 - 066
ROW("度量值",BLANK())
```

在生成的查询表中，选择 Excel 功能区的“表设计”→“表名称”，将表名称命名为度量，如图 3-61 所示。

图 3-61　修改表名称

选择 Excel 功能区的 Power Pivot→“添加到数据模型”，如图 3-62 所示。

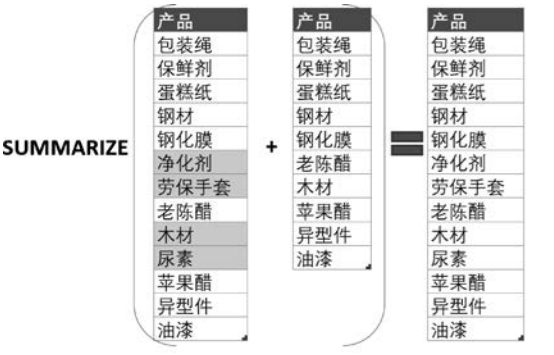
图 3-62　将空表添加到数据模型

3.9.4　创建维度表

创建维度表，用于获取装货表和运单表中的产品字段的去重数据，形成一份完整的产品列表，表达式如下：

```
EVALUATE        //ch3 - 067
SUMMARIZE(
    UNION(VALUES('装货'[产品]),VALUES('运单'[产品]) ),
    [产品]
)
```

SUMMARIZE()函数具备去重功能，实现过程及返回的值如图 3-63 所示。



(a) 查询的数据来源　　(b) 查询返回的值

图 3-63　创建产品列表

创建维度表，用于合并订单表和运单表中产品和包装方式二列的去重后数据，筛选出包装方式含“装”字的数据，表达式如下：

```
EVALUATE       //ch3 - 068
FILTER (
    DISTINCT (
        UNION (
            ALL ( '订单'[产品], '订单'[包装方式] ),
            ALL ( '运单'[产品], '运单'[包装方式] )
            )
    ),
    IFERROR ( FIND ( "装", [包装方式] ), 0 ) > 1
)
```

返回的值如图 3-64 所示。

(a) UNION操作前 (b) UNION与DISTINCT操作后

图 3-64 应用(1)

对产品表中的木材和钢材产品与合同表中的捆、扎、膜包装方式生成笛卡儿积表，然后统计运单表中的产品行数。应用举例，表达式如下：

```
EVALUATE       //ch3 - 069
GENERATE (
    SUMMARIZE (
        FILTER('运单','运单'[产品] IN {"木材","钢材"}),
        '运单'[产品]
    ),
    SUMMARIZE(
        FILTER('合同','合同'[包装方式] IN {"捆","扎","膜"}),
        '合同'[包装方式],
        "数量",
            COUNTROWS (
                RELATEDTABLE ( '运单' )
            )
    )
)
```

返回的值如图 3-65 所示。

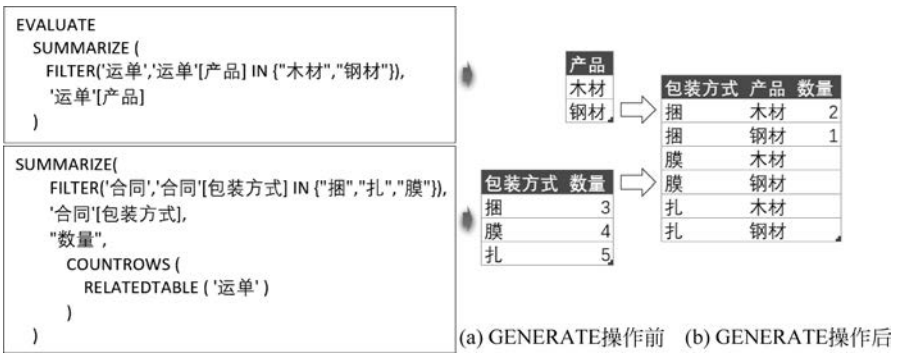


图 3-65　应用(2)

将以上表达式中的 GENERATE()函数替换为 GENERATEALL()函数，返回的值如图 3-66 所示。

包装方式	产品	数量
捆	木材	2
膜	木材	
扎	木材	
捆	钢材	1
膜	钢材	
扎	钢材	

图 3-66　应用(3)

3.10　本章回顾

Power Pivot 数据库也称为 Excel 数据模型，DAX 是 Excel 数据模型的函数公式语言。在使用过程中，处理与分析上百万、千万行级别的数据是常有的事。在未能领悟 DAX 函数的用法及各函数间的细微差异时，如果贸然使用复杂的逻辑去处理大容量的数据，则到时不得不面对数据预期值、准确性及运行效率等的多重考验。

本章对数据模型中常见“筛选、集合与事件、笛卡儿积、连接、投影”等操作分别进行了介绍。本章重点对 ALL()与 VALUES()、FILTER()与 CALCULATETABLE()、SUMMARIZE()与 SUMMARIZECOLUMNS()、GENERATE()与 GENERATEALL()的用法差异进行了比较说明。特别是 ALL()与 VALUES()的说明。ALL()与 VALUES()在 FILTEER()中使用时，它们是表函数；在 CALCULATE()或 CALCULATETABLE()中使用时，它们是调节器。这些知识都是为后续能写出复杂、准确的 DAX 表达式而准备的。

本章的表函数适用于 DAX 表达式中参数为表的应用场景。本章所涉及的知识点较多且有一定的难度。为了方便读者理解，本章案例已做到了尽可能地简化与连贯。在 DAX 学习与使用的过程中，只有在熟悉基础语法之后，才能有更多的精力聚焦于运算逻辑与业务

规则的构建上，从而发挥 DAX 的巨大威力。

以 SUMMARIZE()函数返回的表为例。现打算将 SUMMARIZE()返回的表放置于 SUMX()函数的第 1 个参数(为表函数)中，用于计算各包装方式的入库量(求和)。相关度量值的表达式如下：

```
M.入库量:=
SUMX(
    SUMMARIZE('DK', 'DK'[包装方式], 'DK'[入库]),
    'DK'[入库]
)  -- ch3-070
```

筛选 DK 表中入库量大于 60 的数据，然后对各包装方式的入库量求和，相关度量值表达式如下：

```
M.筛选入库量:=SUMX(
    SUMMARIZE(
        FILTER('DK', 'DK'[入库]>60), 'DK'[包装方式],
        'DK'[入库]
        ),
    'DK'[入库]
)  -- ch3-071
```

创建透视表，将包装方式拖入行标签，勾选 M.入库量和 M.筛选入库量这两个度量值，透视表的结果如图 3-67 所示。

行标签	M.入库量	M.筛选入库量
捆	87	87
散装	76	76
桶装	148	93
箱装	110	78
扎	65	65
总计	**486**	**399**

图 3-67　透视表中的度量值比较

在 DAX 中，有不少函数的单词虽然较长但其实有规律可循，很多的函数其实是多个单词的有机拼接。例如 NATURALINNERJOIN()，它是由 Natural＋Inner＋Join 组成；SUMMARIZECOLUMNS()，它是由 Summarize＋Columns 组成；ADDMISSINGITEMS()则是由 Add＋Missing＋Items 组成。其他长函数均可以此类推，不再一一举例。

在本章示例的基础上，读者仍需扩展领悟及熟悉各函数及用法，以便在后续应用过程中能轻松驾驭各类复杂业务需求。