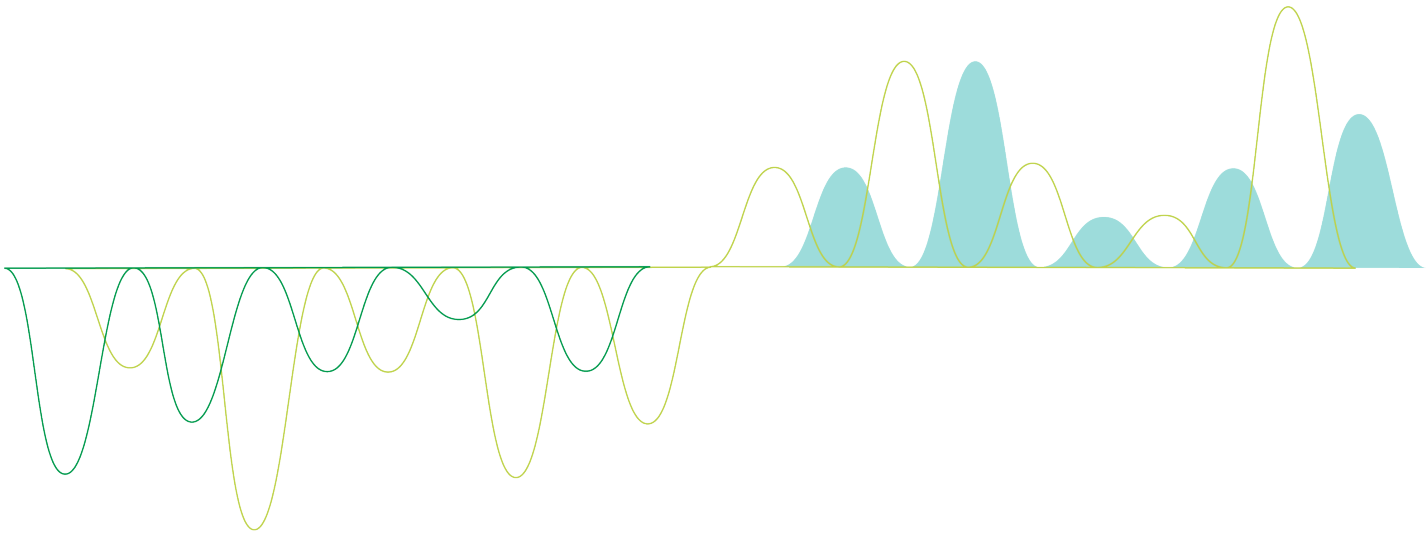
脚本语法和图表函数

Qlik Sense®
May 2022



Contents

1 什么是 Qlik Sense?

Qlik Sense 是一个数据分析平台。使用 Qlik Sense, 您可以自己分析和发现数据。您可以在群组内和组织之间共享知识和分析数据。Qlik Sense 可让您自问自答和发掘深入的见解。Qlik Sense 可让您和您的同事相互协作, 共同决策。

1.1 在 Qlik Sense 中可以做什么?

大多数商业智能 (BI) 产品可以帮助您回答已经得到了解的问题。但是对于后续产生的问题怎么办? 比如说他人阅读您的报表或者查看您的可视化内容之后提出的问题? 凭借 Qlik Sense 的相关经验, 您可以逐步回答更深层次的问题, 从而逐渐发现深入的见解。使用 Qlik Sense, 您可以自由挖掘数据, 通过一些简单的单击操作即可在每一个步骤中不断学习, 并且以结果为基础, 一步一步深入挖掘。

1.2 Qlik Sense 如何工作?

Qlik Sense 随时为您生成信息视图。Qlik Sense 不需要预定义的静态报告, 也不需要依赖其他用户 - 只需要单击几下和自主学习即可。每次单击时, Qlik Sense 会立即作出响应, 并使用新计算的数据集和特定于选择内容的可视化内容更新应用程序中的每个 Qlik Sense 可视化内容和视图。

应用模式

您可以创建自己可以重复使用的 Qlik Sense 应用程序, 并且可以修改和与他人共享, 而不需要部署和管理大量的商业应用程序。该应用模式可以帮助您自行询问和回答下一个问题, 而不必向专家求助新的报告或可视化对象。

关联体验

Qlik Sense 自动管理数据中的所有关系, 并且使用**green/white/gray**指示来向您提供信息。选择内容以绿色高亮显示, 相关的数据以白色表示, 而排除(不相关)的数据则以灰色显示。这种即时反馈可以让您思考新的问题和不断探索发现新的见解。

协作性和可移动性

Qlik Sense 还可以让您随时随地与同事进行协作。所有 Qlik Sense 功能(包括相关经验和协作)均可在移动设备上使用。使用 Qlik Sense, 您可以练习提问和回答问题, 还可以与您的同事一起跟踪问题(不管您身在何处)。

1.3 如何部署 Qlik Sense?

可以部署的 Qlik Sense 有两种版本: Qlik Sense Desktop 和 Qlik Sense Enterprise。

Qlik Sense Desktop

这是一种便于安装的单用户版本, 通常安装在本地计算机上。

Qlik Sense Enterprise

此版本用于部署 Qlik Sense 站点。站点是一台或多台连接到一个共用逻辑存储库或中心节点的服务器机器的集合。

1.4 如何管理 Qlik Sense 站点

使用 Qlik Management Console，能够以简单和直观的方式配置、管理和监控 Qlik Sense 站点。您可以管理许可证、访问权限和安全规则，还可以配置节点和数据源连接，以及在其他许多活动和资源之间同步内容和用户。

1.5 扩展 Qlik Sense 并根据自己的目的进行调整

Qlik Sense 可为您提供灵活的 API 和 SDK 用于开发自己的扩展名，并为不同的目的调整和整合 Qlik Sense，例如：

构建扩展程序和插件

在这里，您可以使用 JavaScript 进行 Web 开发，从而在 Qlik Sense 应用程序构建具有自定义可视化内容的扩展程序，或者利用插件 API 构建包含 Qlik Sense 内容的网站。

构建客户端

您可以在您自己的应用程序中使用 .NET 和嵌入式 Qlik Sense 对象中构建客户端。如果您使用的语言可以使用 Qlik Sense 客户端协议处理 WebSocket 通信，您就还可以使用这种编程语言来构建原生客户端。

构建服务器工具

使用服务和用户目录 API，您可以创建自己的工具来管理 Qlik Sense 站点。

连接到其他数据源

创建 Qlik Sense 连接器以从自定义数据源检索数据。

2 脚本语法概述

2.1 脚本语法简介

在脚本中，定义逻辑中所包含的数据源名称、表格名称和字段名称。此外，存取权限定义中的字段也在脚本中详加定义。脚本由一系列连续执行的语句构成。

Qlik Sense 命令行语法和脚本语法在 Backus-Naur 形式符号或 BNF 代码中进行了介绍。

代码的第一行早在新建 Qlik Sense 文件时已生成。这些数字解释变量的默认值根据操作系统的区域设置派生。

脚本由一系列连续执行的脚本语句和关键字构成。所有脚本语句必须以分号“**;**”结束。

可以在 **LOAD** 语句中使用表达式和函数转换已经加载的数据。

对于使用逗号、制表符或分号作为分隔符的表格文件，可能会使用 **LOAD** 语句。**LOAD** 语句会默认加载文件的全部字段。

通用数据库可通过 ODBC 或 OLE DB 数据库连接器进行访问。此处使用的是一般标准 SQL 语句。SQL 语法接受不同 ODBC 驱动程序之间的区别。

此外，可以使用自定义连接器访问其他数据源。

2.2 什么是 Backus-Naur 形式？

Qlik Sense 命令行语法和脚本语法在 Backus-Naur 形式符号（也称为 BNF 代码）中进行了介绍。

下表提供了在 BNF 代码中使用的符号的列表，以及如何解释这些符号的说明：

符号

符号	说明
|	逻辑 OR：任何一侧的符号均可使用。
( )	定义优先级的括号：用于构建 BNF 语法。
[ ]	方括号：括号内项目为可选项。
{ }	大括号：括号内项目可不重复或重复多次。
符号	非终端语法类别，即：可进一步分隔为其他符号。例如，上述符号的复合体，其他非终端符号和文本字符串等。
::=	开端标记，用于符号定义模块。
LOAD	由文本字符串构成的终端符号。应依照其在脚本内的原样写入。

所有终端符号使用 **bold face** 字体呈现。例如，“(”应解释为定义优先级的括号，但是“**(**”则应解释为脚本内的一个字符。

示例：

alias 语句的说明如下：

alias fieldname **as** aliasname { , fieldname **as** aliasname}

这应该解释为文本字符串“alias”，其后为任意字段名称，文本字符串“as”以及任意别名。可以指定“fieldname as alias”的任意数量的其他组合，其间用逗号分隔。

下面是正确的语句：

```
alias a as first;
alias a as first, b as second;
alias a as first, b as second, c as third;
```

下面则是错误的语句：

```
alias a as first b as second;
alias a as first { , b as second };
```

2 脚本语句和关键字

Qlik Sense 脚本由许多语句组成。语句可以是脚本常规语句或脚本控制语句。某些语句可前置前缀。

常规语句通常用于以某种方式或其他方式操作数据。这些语句可能被脚本中的行编号覆盖且必须总是以分号";"终止。

控制语句通常用于控制脚本执行流程。控制语句中每一个子句必须保持在一个脚本行内，并且可能以分号或换行符终止。

前缀可用于常规语句，但不可用以控制语句。**when** 和 **unless** 前缀可用作少数指定控制语句子句的后缀。

在下一子章节，您可看到有关所有脚本语句，控制语句和前缀的字母索引表。

所有脚本关键字可以大小写字符的任意组合输入。用于脚本中的字段和变量名要区分大小写。

2.3 脚本控制语句

Qlik Sense 脚本由许多语句组成。语句可以是脚本常规语句或脚本控制语句。

控制语句通常用于控制脚本执行流程。控制语句中每一个子句必须保持在一个脚本行内，并且可能以分号或换行符终止。

前缀从不用于控制语句，除了 **when** 和 **unless** 前缀可用于少数指定的控制语句。

所有脚本关键字可以大小写字符的任意组合输入。

脚本控制语句概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Call

call 控制语句可调用必须由先前的 **sub** 语句定义的子例程。

```
Call name ( [ paramlist ])
```

Do..loop

do..loop 控制语句是一个脚本迭代构造，可不断执行一个或几个语句，直到逻辑条件得到满足为止。

```
Do..loop [ ( while | until ) condition ] [statements]
[exit do [ ( when | unless ) condition ] [statements]
loop [ ( while | until ) condition ]
```

Exit script

此控制语句可以停止执行脚本。可以插入到脚本的任何位置。

```
Exit script[ (when | unless) condition ]
```

For each ..next

for each..next 控制语句是一个脚本迭代构造，可为逗号分隔列表中的每个值执行一个或几个语句。列表中的每个值均会执行由 **for** 和 **next** 限定的循环中的语句。

```
For each..next var in list
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
next [var]
```

For..next

for..next 控制语句是一个带有计数器的脚本迭代构造。指定的高低限值之间的计数器变量的每个值均会执行由 **for** 和 **next** 限定的循环中的语句。

```
For..next counter = expr1 to expr2 [ stepexpr3 ]
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
Next [counter]
```

If..then

if..then 控制语句是一个脚本选择结构，其可根据一个或几个逻辑条件按照不同路径强制执行脚本。

*由于 **if..then** 语句是控制语句，并以分号或换行符结束，四个可能子句（**if..then**、**elseif..then**、**else** 和 **end if**）中任意一个子句都不得跨越行边界。*

```
If..then..elseif..else..end if condition then
  [ statements ]
{ elseif condition then
  [ statements ] }
[ else
  [ statements ] ]
end if
```

Sub

sub..end sub 控制语句用于定义可从 **call** 语句中调用的子例程。

```
Sub..end sub name [ ( paramlist )] statements end sub
```

Switch

switch 控制语句是一个脚本选择项构造，根据表达式值，以不同路径强制执行脚本。

```
Switch..case..default..end switch expression {case valuelist [ statements ]}
[default statements] end switch
```

Call

call 控制语句可调用必须由先前的 **sub** 语句定义的子例程。

语法：

```
Call name ( [ paramlist ])
```

参数：

参数

参数	说明
name	子例程的名称。
paramlist	实际参数逗号分隔符列表，用以发送至子例程。此列表中每个项目都可为字段名，变量或任意表达式。

由一条 **call** 语句调用的子例程必须由在脚本执行期间更先遇到的 **sub** 语句定义。

参数会复制到子例程中，此外，如果在 **call** 语句中的参数是一个变量而非表达式，重新将其复制到现有子例程中。

限制：

- 由于该 **call** 语句是一个控制语句，以分号或换行符结束，因此不能跨越行边界。
- 当在控制语句中用 `Sub..end sub` 定义子例程时，例如 `if..then`，则只能从同一控制语句中调用该子例程。

示例：

该示例列出文件夹及其子文件夹中所有和 Qlik 相关的文件，并将文件信息存储在表格中。假设您已创建了指向文件夹的名为 Apps 的连接。

参考文件夹调用了 DoDir 子例程，将 'lib://Apps' 作为参数。在子例程内，存在递归调用 `Call DoDir (Dir)`，让函数在子文件夹中递归式查找文件。

```
sub DoDir (Root)    For Each Ext in 'qvw', 'qvo', 'qvs', 'qvt', 'qvd', 'qvc', 'qvf'       For
Each File in filelist (Root&'\*.' &Ext)          LOAD             '$(File)' as Name,
      FileSize( '$(File)' ) as Size,               FileTime( '$(File)' ) as FileTime
autogenerate 1;       Next File    Next Ext       For Each Dir in dirlist (Root&'\*' )
Call DoDir (Dir)    Next Dir End Sub  Call DoDir ('lib://Apps')
```

Do..loop

do..loop 控制语句是一个脚本迭代构造，可不断执行一个或几个语句，直到逻辑条件得到满足为止。

语法：

```
Do [ ( while | until ) condition ] [statements]
[exit do [ ( when | unless ) condition ] [statements]
loop[ ( while | until ) condition ]
```

*由于 **do..loop** 语句是控制语句，并以分号或换行符结束，三个可能子句（**do**、**exit do** 和 **loop**）中任意一个子句都不得跨越行边界。*

参数：

参数

参数	说明
condition	用于评估 True 或 False 的逻辑表达式。
statements	一个或多个 Qlik Sense 脚本语句的任意组。
while / until	**while** 或 **until** 条件子句在任何 **do..loop** 语句中必须只能出现一次，即要么在 **do** 之后，要么在 **loop** 之后。只有首次遇到时才会解释每一个条件，但在循环中每次遇到时都求值。
exit do	如果在循环内遇到 **exit do** 子句，则脚本执行会转移至表示循环结束的 **loop** 子句之后的第一个语句。**exit do** 子句可通过选择性使用 **when** 或 **unless** 后缀变为有条件子句。

示例：

```
// LOAD files file1.csv..file9.csv
Set a=1;
Do while a<10
LOAD * from file$(a).csv;
Let a=a+1;
Loop
```

End

End 脚本关键字用于关闭 **If**、**Sub** 和 **Switch** 子句。

Exit

Exit 脚本关键字是 **Exit Script** 语句的一部分，但可用来退出 **Do**、**For** 或 **Sub** 子句。

Exit script

此控制语句可以停止执行脚本。可以插入到脚本的任何位置。

语法：

```
Exit Script [ (when | unless) condition ]
```

由于该 **exit script** 语句是一个控制语句，以分号或换行符结束，因此不能跨越行边界。

参数：

参数

参数	说明
condition	用于评估 True 或 False 的逻辑表达式。
when / unless	**exit script** 语句可通过选择性使用 **when** 或 **unless** 子句变为有条件子句。

示例：

```
//Exit script
Exit Script;

//Exit script when a condition is fulfilled
Exit Script when a=1
```

For..next

for..next 控制语句是一个带有计数器的脚本迭代构造。指定的高低限值之间的计数器变量的每个值均会执行由 **for** 和 **next** 限定的循环中的语句。

语法：

```
For counter = expr1 to expr2 [ step expr3 ]
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
Next [counter]
```

表达式 *expr1*、*expr2* 和 *expr3* 仅会在首次进入循环时进行求值。计数器变量的值可通过循环内的语句进行更改，但这并非出色的编程做法。

如果在循环内遇到 **exit for** 子句，则脚本执行会转移至表示循环结束的 **next** 子句之后的第一个语句。**exit for** 子句可通过选择性使用 **when** 或 **unless** 后缀变为有条件子句。

*由于 **for..next** 语句是控制语句，并以分号或换行符结束，三个可能子句（**for..to..step**、**exit for** 和 **next**）中任意一个子句都不得跨越行边界。*

参数：

参数

参数	说明
counter	一个变量名。如果 *counter* 在 **next** 之后指定，变量名必须与对应的 **for** 之后查找的变量名相同。
expr1	一个表达式，可决定与应执行循环有关的 *counter* 变量的第一个值。
expr2	一个表达式，可决定与应执行循环有关的 *counter* 变量的最后一个值。
expr3	一个表达式，可决定每执行一次循环 *counter* 变量增加的值。
condition	用于评估 True 或 False 的逻辑表达式。
statements	一个或多个 Qlik Sense 脚本语句的任意组。

Example 1: 加载文件序列

```
// LOAD files file1.csv..file9.csv
for a=1 to 9

      LOAD * from file$(a).csv;

next
```

Example 2: 加载随即文件数量

在本例中，我们假定有数据文件 *x1.csv*、*x3.csv*、*x5.csv*、*x7.csv* 和 *x9.csv*。加载在使用 `if rand( )<0.5 then` 条件的随机点停止。

```
for counter=1 to 9 step 2

      set filename=x$(counter).csv;
      if rand( )<0.5 then

            exit for unless counter=1

      end if
      LOAD a,b from $(filename);

next
```

For each..next

for each..next 控制语句是一个脚本迭代构造，可为逗号分隔列表中的每个值执行一个或几个语句。列表中的每个值均会执行由 **for** 和 **next** 限定的循环中的语句。

语法：

特殊语法可以生成带有当前目录内文件和目录名称的列表。

```
for each var in list
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
```

```
next [var]
```

参数：

参数

参数	说明
var	脚本变量名称，可为每次循环执行获取列表中的新值。如果 **var** 在 **next** 之后指定，变量名必须与对应的 **for each** 之后查找的变量名相同。

var 变量的值可通过循环内的语句进行更改，但这并非出色的编程做法。

如果在循环内遇到 **exit for** 子句，则脚本执行会转移至表示循环结束的 **next** 子句之后的第一个语句。**exit for** 子句可通过选择性使用 **when** 或 **unless** 后缀变为有条件子句。

*由于 **for each..next** 语句是控制语句，并以分号或换行符结束，三个可能子句（**for each**、**exit for** 和 **next**）中任意一个子句都不得跨越行边界。*

语法：

```
list := item { , item }
item := constant | (expression) | filelist mask | dirlist mask |
fieldvaluelist mask
```

参数

参数	说明
constant	任何数字或字符串。请注意，直接在脚本中写入的字符串必须附上单引号。没有单引号的字符串将被解释为变量，而变量的值之后将被使用。数字不必用单引号引起来。
expression	任意表达式。
mask	文件名称或文件夹名称掩码，包括任何有效的文件名称字符及标准通配符，比如 ***** 和 **?**。 您可以使用绝对文件路径或 lib:// 路径。
condition	用于评估 True 或 False 的逻辑表达式。
statements	一个或多个 Qlik Sense 脚本语句的任意组。
filelist mask	该语法会在匹配文件名称掩码的当前目录中生成逗号分隔的全部文件列表。 *此参数仅在标准模式下支持库连接。*

参数	说明
dirlist mask	该语法会在匹配文件夹名称掩码的当前文件夹中生成逗号分隔的全部文件夹列表。 *此参数仅在标准模式下支持库连接。*
fieldvaluelist mask	此语法迭代已经加载到 Qlik Sense 的字段值。

Qlik Web 存储提供程序连接器 以及其它 DataFiles 连接不支持使用通配符(和 ?)字符的筛选器掩码。*

Example 1: 加载文件列表

```
// LOAD the files 1.csv, 3.csv, 7.csv and xyz.csv for each a in 1,3,7,'xyz'    LOAD * from
file$(a).csv; next
```

Example 2: 在磁盘上创建文件列表

此示例加载文件夹中所有 Qlik Sense 相关文件的列表。

```
 sub DoDir (Root)    for each Ext in 'qvw', 'qva', 'qvo', 'qvs', 'qvc', 'qvf', 'qvd'
for each File in filelist (Root&'/*.' &Ext)          LOAD             '$(File)' as Name,
         FileSize( '$(File)' ) as Size,             FileTime( '$(File)' ) as FileTime
 autogenerate 1;        next File     next Ext    for each Dir in dirlist (Root&'/*' )
call DoDir (Dir)     next Dir  end sub  call DoDir ('lib://DataFiles')
```

Example 3: 迭代字段值

此示例迭代已加载的 FIELD 值列表，并生成新字段 NEWFIELD。对每个 FIELD 值，都会创建两条 NEWFIELD 记录。

```
 load * inline [ FIELD one two three ];  FOR Each a in FieldValueList('FIELD') LOAD '$(a)' &'-
'&RecNo() as NEWFIELD AutoGenerate 2; NEXT a
```

最终生成的表格如下所示：

Example table

NEWFIELD
one-1
one-2
two-1
two-2
three-1
three-2

If..then..elseif..else..end if

if..then 控制语句是一个脚本选择结构，其可根据一个或几个逻辑条件按照不同路径强制执行脚本。

控制语句通常用于控制脚本执行流程。在图表表达式中，改用 **if** 条件函数。

语法：

```
If  condition then
  [ statements ]
{ elseif condition then
  [ statements ] }
[ else
  [ statements ] ]
end if
```

由于 **if..then** 语句是控制语句，并以分号或换行符结束，四个可能子句（**if..then**、**elseif..then**、**else** 和 **end if**）中任意一个子句都不得跨越行边界。

参数：

参数

参数	说明
condition	求值为 True 或 False 的逻辑表达式。
statements	一个或多个 Qlik Sense 脚本语句的任意组。

Example 1:

```
if a=1 then

      LOAD * from abc.csv;
      SQL SELECT e, f, g from tab1;

end if
```

Example 2:

```
if a=1 then; drop table xyz; end if;
```

Example 3:

```
if x>0 then

      LOAD * from pos.csv;

elseif x<0 then

      LOAD * from neg.csv;

else
```

```
    LOAD * from zero.txt;
end if
```

Next

Next 脚本关键字用于结束 **For** 循环。

Sub..end sub

sub..end sub 控制语句用于定义可从 **call** 语句中调用的子例程。

语法：

```
Sub name [ ( paramlist )] statements end sub
```

自变量将复制到子例程，而如果 **call** 语句中相应实际参数是变量名，则退出子例程时这些参数将再次从现有子例程中复制回来。

如果子例程通过 **call** 语句调用的形式参数比实际参数多，额外的形式参数将初始化为 NULL 值，且可在子例程中用作局部变量。

参数：

参数

参数	说明
name	子例程的名称。
paramlist	子例程形式参数变量名列表的逗号分隔符列表。这些可用作子例程内的任意变量。
statements	一个或多个 Qlik Sense 脚本语句的任意组。

限制：

- 由于 **sub** 语句是控制语句，并以分号或行尾结束，两个可能子句（**sub** 和 **end sub**）中任意一个子句都不得跨越行边界。
- 当在控制语句中用 `Sub..end sub` 定义子例程时，例如 `if..then`，则只能从同一控制语句中调用该子例程。

Example 1:

```
Sub INCR (I,J)
I = I + 1
Exit Sub when I < 10
J = J + 1
End Sub
Call INCR (X,Y)
```

Example 2: - 参数传递

```
Sub ParTrans (A,B,C)
```

```
A=A+1
B=B+1
C=C+1
End Sub
A=1
X=1
C=1
Call ParTrans (A, (X+1)*2)
```

以上结果将在本地子例程内，A 将初始化为 1，B 将初始化为 4，C 将初始化为 NULL 值。

退出子例程时，全局变量 A 会将 2 作为值(从子例程复制回来)。第二个实际参数 “(X+1)*2” 不会复制回来，因为其不是变量。最终，全局变量 C 不会受到子例程调用的影响。

Switch..case..default..end switch

switch 控制语句是一个脚本选择项构造，根据表达式值，以不同路径强制执行脚本。

语法：

```
Switch expression {case valuelist [ statements ]} [default statements] end switch
```

*由于 **switch** 语句是控制语句，并以分号或换行符结束，四个可能子句(**switch**、**case**、**default** 和 **end switch**) 中任意一个子句都不得跨越行边界。*

参数：

参数

参数	说明
expression	任意表达式。
valuelist	逗号分隔的值列表，可以在其中比较表达式的值。执行此脚本将继续沿用第一组中在值列表和表达式中相等的值的语句。值列表中的每一个值都可以是任意表达式。如果在任意 **case** 子句中都无匹配值，则将执行 **default** 子句下的语句(如果指定)。
statements	一个或多个 Qlik Sense 脚本语句的任意组。

示例：

```
Switch I
Case 1
LOAD '$(I): CASE 1' as case autogenerate 1;
Case 2
LOAD '$(I): CASE 2' as case autogenerate 1;
Default
LOAD '$(I): DEFAULT' as case autogenerate 1;
End Switch
```

To

To 脚本关键字可用于多个脚本语句。

2.4 脚本前缀

前缀可用于常规语句，但不可用以控制语句。**when** 和 **unless** 前缀可用作少数指定控制语句子句的后缀。

所有脚本关键字可以大小写字符的任意组合输入。用于脚本中的字段和变量名要区分大小写。

脚本前缀概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Add

可将前缀 **Add** 添加至脚本中的任何 **LOAD** 或 **SELECT** 语句，以指定其应当将记录添加至另一个表。它还指定此语句应在部分重新加载中运行。**Add** 前缀还可用在 **Map** 语句中。

```
Add [only] [Concatenate[(tablename )]] (loadstatement | selectstatement)
Add [ Only ] mapstatement
```

Buffer

QVD 文件可通过 **buffer** 前缀自动创建和维护。该前缀可用于脚本中大多数 **LOAD** 和 **SELECT** 语句。这表示 QVD 文件可用于*缓存*/缓冲该语句产生的结果。

```
Buffer[(option [ , option])] ( loadstatement | selectstatement )
option::= incremental | stale [after] amount [(days | hours)]
```

Concatenate

如果要进行串联的两个表格具有不同的字段集，仍然可以使用 **Concatenate** 前缀强制串联两个表格。

```
Concatenate[ (tablename ) ] ( loadstatement | selectstatement )
```

Crosstable

crosstable 前缀用于将交叉表转换为垂直表，也就是将包括许多列的宽表格转换为长表格，转换时将列标题放到单个属性列中。

```
Crosstable (attribute field name, data field name [ , n ] ) ( loadstatement |
selectstatement )
```

First

First 前缀(属于 **LOAD** 或 **SELECT (SQL)** 语句) 前缀用于从数据源表格加载记录的一组最大数。

```
First n( loadstatement | selectstatement )
```

Generic

使用 **generic** 前缀可以打开和加载通用数据库。

```
Generic ( loadstatement | selectstatement )
```

Hierarchy

hierarchy 前缀用于将父子层次表格转换成在 Qlik Sense 数据模型中有用的表格。此前缀可能置于 **LOAD** 或 **SELECT** 语句前面，并会使用加载的语句结果作为表格转换的输入。

```
Hierarchy (NodeID, ParentID, NodeName, [ParentName], [PathSource],
[PathName], [PathDelimiter], [Depth])(loadstatement | selectstatement)
```

HierarchBelongsTo

此前缀用于将父子层次表格转换成在 Qlik Sense 数据模型中有用的表格。此前缀可能置于 **LOAD** 或 **SELECT** 语句前面，并会使用加载的语句结果作为表格转换的输入。

```
HierarchyBelongsTo (NodeID, ParentID, NodeName, AncestorID, AncestorName,
[DepthDiff])(loadstatement | selectstatement)
```

Inner

可在 **join** 和 **keep** 前缀前面使用 **inner** 前缀。如果用于 **join** 之前，说明应使用内部联接。由此生成的表格仅包含原始数据表格(其中链接字段值在两个表格中均有呈现)的字段值组合。如果用于 **keep** 之前，说明在 Qlik Sense 中存储这些表格之前，首先应使两个原始数据表格缩减为它们的共同交集。.

```
Inner ( Join | Keep) [ (tablename) ](loadstatement |selectstatement )
```

IntervalMatch

IntervalMatch 前缀用于创建表格以便将离散数值与一个或多个数值间隔进行匹配，并且任选匹配一个或多个额外关键值。

```
IntervalMatch (matchfield)(loadstatement | selectstatement )
IntervalMatch (matchfield,keyfield1 [ , keyfield2, ... keyfield5 ] )
(loadstatement | selectstatement )
```

Join

join 前缀可连接加载的表格和现有已命名的表格或最近创建的数据表。

```
[Inner | Outer | Left | Right ] Join [ (tablename ) ]( loadstatement |
selectstatement )
```

Keep

keep 前缀类似于 **join** 前缀。与 **join** 前缀一样，该前缀可用来将加载的表格与现有的命名表格或最后一个之前创建的数据表格进行比较，而不是将加载的表格与现有的表格进行合并，它可以在将表格存储在 Qlik Sense 中之前，根据表格数据的交集减少一个或同时减少两个表格。这种比较相当于对所有共同字段进行自然联接，即等同于相应联接的方式。但是，这两个表格并未合并，而将作为两个单独命名的表格保留在 Qlik Sense 中。

```
(Inner | Left | Right) Keep [(tablename ) ]( loadstatement | selectstatement
)
```

Left

可在 **Join** 和 **Keep** 前缀前面使用 **left** 前缀。

如果用于 **join** 之前，说明应使用左侧联接。由此生成的表格仅包含原始数据表格的字段值组合，在原始数据表格中，链接字段值呈现在第一个表格中。如果用于 **keep** 之前，说明首先应使第二原始数据表格缩减为其与第一表格间的共同交集，然后才可在 Qlik Sense 中存储此表格。

```
Left ( Join | Keep) [ (tablename) ](loadstatement |selectstatement )
```

Mapping

mapping 前缀用于创建映射表，例如，此映射表在脚本运行期间可用于替换字段值和字段名。

```
Mapping  ( loadstatement | selectstatement )
```

Merge

可将前缀 **Merge** 添加至脚本中的任何 **LOAD** 或 **SELECT** 语句，以指定加载的表格应当合并到另一表中。它还指定此语句应在部分重新加载中运行。

```
合并 [only] [(SequenceNoField [, SequenceNoVar])] On ListOfKeys [Concatenate
[(TableName)]] (loadstatement | selectstatement)
```

NoConcatenate

NoConcatenate 前缀强制将两个使用相同字段集的加载表格处理为两个单独的内部表格(当它们以其他方式自动串联时)。

```
NoConcatenate( loadstatement | selectstatement )
```

Outer

显式 **Join** 前缀前面可带前缀 **Outer** 以指定外部联接。在外部联接中，生成两个表格之间的所有组合。由此生成的表格包含原始数据表格(其中链接字段值在两个表格中均有呈现)的字段值组合。**Outer** 关键字是可选型，并用作未指定联接前缀时的默认联接类型。

```
Outer Join [ (tablename) ](loadstatement |selectstatement )
```

Partial reload

完全重新加载总是从删除现有数据模型中的所有表开始，然后运行加载脚本。*部分加载 (page 56)* 将不进行该操作。相反，它会保留数据模型中的所有表格，然后仅执行 **Load** 和 **Select** 语句，这些语句带有前缀 **Add**、**Merge** 或 **Replace** 前缀。其他数据表不受该命令的影响。**only** 参数表示只应在部分重新加载期间执行该语句，而在完全重新加载期间应忽略该语句。下表总结了部分和完全重新加载的语句执行情况。

Replace

Replace前缀 可添加至脚本中的任何 **LOAD** 或 **SELECT** 语句，以指定加载的表格应当替代另一表格。它还指定此语句应在部分重新加载中运行。**Replace** 前缀还可用在 **Map** 语句中。

```
Replace [only] [Concatenate[(tablename) ]] (loadstatement | selectstatement)
Replace [only] mapstatement
```

Right

可在 **Join** 和 **Keep** 前缀前面使用 **right** 前缀。

如果用于 **join** 之前，说明应使用右侧联接。由此生成的表格仅包含原始数据表格的字段值组合，原始数据表格中的链接字段值呈现在第二个表格中。如果用于 **keep** 之前，说明首先应使第一原始数据表格缩减为其与第二表格间的共同交集，然后才可在 Qlik Sense 中存储此表格。

```
Right (Join | Keep) [(tablename)](loadstatement |selectstatement )
```

Sample

LOAD 或 **SELECT** 语句的 **sample** 前缀用于从数据源载入记录的随机样本。

```
Sample p ( loadstatement | selectstatement )
```

Semantic

可通过 **semantic** 前缀加载包含两个记录之间关系的表格。例如，这可以是表格内的自引用，即其中一个记录指向另一个记录，如所属的父项或祖先。

```
Semantic ( loadstatement | selectstatement)
```

Unless

unless 前缀和后缀用于创建确定是否应计算语句或 **exit** 子句的条件子句。它可以被看作是完整的 **if..end if** 语句的简洁替代形式。

```
(Unless condition statement | exitstatement Unless condition )
```

When

when 前缀和后缀用于创建确定是否应执行语句或 **exit** 子句的条件子句。它可以被看作是完整的 **if..end if** 语句的简洁替代形式。

```
( When condition statement | exitstatement when condition )
```

Add

可将前缀 **Add** 添加至脚本中的任何 **LOAD** 或 **SELECT** 语句，以指定其应当将记录添加至另一个表。它还指定此语句应在部分重新加载中运行。**Add** 前缀还可用在 **Map** 语句中。

要使部分重新加载正常工作，必须在触发部分重新加载之前使用数据打开应用程序。

使用**重新加载**按钮执行部分重新加载。您还可使用Qlik Engine JSON API。

语法：

```
Add [only] [Concatenate[(tablename)]] (loadstatement | selectstatement)
```

```
Add [only] mapstatement
```

在正常(非部分)重新加载期间，**Add LOAD** 构造将作为普通 **LOAD** 语句作用。记录将生成并存储在表中。

如果使用了 **Concatenate** 前缀，或者存在具有相同字段集的表，则记录将附加到相关的现有表中。否则，**Add LOAD** 构造将创建新表。

局部重新加载有相同作用。唯一的差异在于 **Add LOAD** 构造将永远不会新建表格。从上一个脚本执行中总是存在一个相关的表，记录应该附加到该表中。

无须检查副本。因此，使用 **Add**前缀的语句通常包含 distinct 限定符或 where 子句来保护副本。

Add Map...Using 语句在部分脚本执行期间也会导致映射发生。

参数：

参数

参数	说明
only	可选限定符表示仅应在部分重新加载期间执行语句。在正常(非部分)重新加载期间，应忽略此项。

示例和结果：

示例	结果
Tab1: LOAD Name, Number FROM Persons.csv; Add LOAD Name, Number FROM newPersons.csv;	常规重新加载期间，将会从 *Persons.csv* 加载数据，并存储到 Qlik Sense 表格 Tab1 中。*NewPersons.csv* 中的数据随后会串联至相同的 Qlik Sense 表格。 在部分重新加载期间，将会从 *NewPersons.csv* 加载数据，并存储到 Qlik Sense 表格 Tab1 中。无须检查副本。
Tab1: SQL SELECT Name, Number FROM Persons.csv; Add LOAD Name, Number FROM NewPersons.csv where not exists(Name);	要检查副本，可查看以前加载的表格内是否存在 Name。 常规重新加载期间，将会从 *Persons.csv* 加载数据，并存储到 Qlik Sense 表格 Tab1 中。*NewPersons.csv* 中的数据随后会串联至相同的 Qlik Sense 表格。 在部分重新加载期间，将会从 *NewPersons.csv* 加载数据，并存储到 Qlik Sense 表格 Tab1 中。要检查副本，可查看以前加载的表格数据内是否存在 Name。
Tab1: LOAD Name, Number FROM Persons.csv; Add Only LOAD Name, Number FROM NewPersons.csv where not exists(Name);	常规重新加载期间，将会从 *Persons.csv* 加载数据，并存储到 Qlik Sense 表格 Tab1 中。忽略加载 *NewPersons.csv* 的语句。 在部分重新加载期间，将会从 *NewPersons.csv* 加载数据，并存储到 Qlik Sense 表格 Tab1 中。要检查副本，可查看以前加载的表格数据内是否存在 Name。

Buffer

QVD 文件可通过 **buffer** 前缀自动创建和维护。该前缀可用于脚本中大多数 **LOAD** 和 **SELECT** 语句。这表示 QVD 文件可用于缓存/缓冲该语句产生的结果。

语法：

```
Buffer [(option [ , option])] ( loadstatement | selectstatement )
option::= incremental | stale [after] amount [(days | hours)]
```

如果未使用任何选项，则首次执行脚本时创建的 QVD 缓冲将无限期使用。

缓冲文件存储在*缓冲*子文件夹中，通常是 *C:\ProgramData\Qlik\Sense\Engine\Buffers*（服务器安装）或 *C:\Users\{user}\Documents\Qlik\Sense\Buffers* (Qlik Sense Desktop)。

QVD 文件的名称是计算名称，整个后续 **LOAD** 或**SELECT** 语句或其他区别性信息的 160 位十六进制散列。这就意味着 QVD 缓冲在后续**LOAD** 或**SELECT** 语句有任何更改的情况下都会无效。

QVD 缓冲通常在创建脚本的应用程序内整个脚本执行过程不再引用时移除，或者在创建脚本的应用程序不再存在时移除。

参数：

参数

参数	说明
增量	incremental 选项可实现仅读取基础文件的一部分。文件先前大小存储在 QVD 文件中的 XML 页眉中。这对日志文件特别有用。上一步载入的全部记录都可从 QVD 文件读取，而后续新记录可从原始数据源读取，这样就可创建一个 QVD 更新文件。 incremental 选项只能用于 **LOAD** 语句和文本文件。在更改或删除旧数据的情况下，不能使用增量加载。
stale [after] amount [(days | hours)]	amount 即指定时间周期的数字。可能要使用小数。如果省略，则假定单位为天数。 stale after 选项通常与 DB 源一起使用，DB 源在初始数据上并无简单的时间戳。相反，您可以指定 QVD 快照将能用多久。stale after 子句仅陈述自 QVD 缓冲创建时间计起的时间周期，此后其即被视为无效。QVD 缓冲在此之前会被用作数据源，在此之后则使用原始数据源。QVD 缓冲文件随后会自动更新，同时新周期开始。

限制：

当然也存在许多限制，最明显的一点就是在任意复杂语句的核心必须有一个文件 **LOAD** 或 **SELECT** 语句。

Example 1:

```
Buffer SELECT * from MyTable;
```

Example 2:

```
Buffer (stale after 7 days) SELECT * from MyTable;
```

Example 3:

```
Buffer (incremental) LOAD * from MyLog.log;
```

Concatenate

如果要进行串联的两个表格具有不同的字段集，仍然可以使用 **Concatenate** 前缀强制串联两个表格。此语句可以强制串联现有的已命名表格或之前创建的最新逻辑表格。

语法：

```
Concatenate[ (tablename ) ] ( loadstatement | selectstatement )
```

串联大体上与 **SQL UNION** 语句相同，但有两点不同：

- 不管表格是否包含相同的字段名，**Concatenate** 前缀都可使用。
- 相同的记录不会随 **Concatenate** 前缀删除。

参数：

参数

参数	说明
tablename	现有表格的名称。

示例：

```
Concatenate LOAD * From file2.csv;
Concatenate SELECT * From table3;
tab1:
LOAD * From file1.csv;
tab2:
LOAD * From file2.csv;
.. .. ..
Concatenate (tab1) LOAD * From file3.csv;
```

Crosstable

crosstable 前缀用于将交叉表转换为垂直表，也就是将包括许多列的宽表格转换为长表格，转换时将列标题放到单个属性列中。

语法：

```
crosstable (attribute field name, data field name [ , n ] ) ( loadstatement |
selectstatement )
```

参数：

参数

参数	说明
attribute field name	包含属性值的字段。

参数	说明
data field name	包含数据值的字段。
n	要被转换成常规形式的表格前面的限定符字段数量。默认值为 1。

交叉表是常见的表格类型，特点是在两个或更多标题数据的正交列表之间显示值矩阵，其中有一个标题数据用作列标题。一个典型的示例就是每月下有一列。**crosstable** 前缀的结果是，列标题（如月份名称）将存储在一个字段（属性字段），而列数据（月份数）将存储在第二个字段：数据字段。

示例

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
tmpData: //Crosstable (MonthText, Sales) Load * inline [ Product, Jan 2021, Feb 2021, Mar 2021, Apr 2021, May 2021, Jun 2021 A, 100, 98, 103, 63, 108, 82 B, 284, 279, 297, 305, 294, 292 C, 50, 53, 50, 54, 49, 51];   //Final: //Load Product, //Date(Date#(MonthText,'MMM YYYY'),'MMM YYYY') as Month, //Sales  //Resident tmpData;  //Drop Table tmpData;
```

结果

结果表

产品	Jan 2021	Feb 2021	Mar 2021	Apr 2021	2021 年 5 月	Jun 2021
A	100	98	103	63	108	82
B	284	279	297	305	294	292
C	50	53	50	54	49	51

解释

这个例子展示了一个交叉表，每个月一列，每个产品一行。按照目前的格式，该数据不容易分析。最好将所有数字放在一个字段中，将所有月份放在另一个字段中，也就是说，放在一个三列表格中。接下来，让我们看看如何对交叉表进行这样的转换。

转换交叉表

取消对脚本的注释并运行它。

```
tmpData: Crosstable (MonthText, Sales) Load * inline [ Product, Jan 2021, Feb 2021, Mar 2021, Apr 2021, May 2021, Jun 2021 A, 100, 98, 103, 63, 108, 82 B, 284, 279, 297, 305, 294, 292 C, 50, 53, 50, 54, 49, 51];   Final: Load Product, Date(Date#(MonthText,'MMM YYYY'),'MMM YYYY') as Month, Sales  Resident tmpData;  Drop Table tmpData;
```

结果

结果表

产品	月	销售额值
A	Jan 2021	100
A	Feb 2021	98
A	Mar 2021	103
A	Apr 2021	63
A	May 2021	108
A	Jun 2021	82
B	Jan 2021	284
B	Feb 2021	279
B	Mar 2021	297
B	Apr 2021	305
B	May 2021	294
B	Jun 2021	292
C	Jan 2021	50
C	Feb 2021	53
C	Mar 2021	50
C	Apr 2021	54
C	May 2021	49
C	Jun 2021	51

解释

交叉表被转换成一个垂直表，其中一列用于月份，另一列用于销售额值。

First

First 前缀（属于 **LOAD** 或 **SELECT (SQL)** 语句）前缀用于从数据源表格加载记录的一组最大数。

语法：

```
First n ( loadstatement | selectstatement )
```

参数：

参数

参数	说明
n	求值为整数的任意表达式，表示需要读取的最大记录数。 *n* 可包含在括号中（如 *(n)*），但这并不是强制要求。

示例：

```
First 10 LOAD * from abc.csv;
First (1) SQL SELECT * from Orders;
```

Generic

使用 **generic** 前缀可以打开和加载通用数据库。

通用数据库/数据源包含结构化重复数据，例如，地址列表或产品规格工作表，其中按照类似属性重复描述实体。

语法：

```
Generic( loadstatement | selectstatement )
```

示例：

```
Generic LOAD * from abc.csv;
Generic SQL SELECT * from table1;
```

通过 **generic** 语句加载的表格不可自动连接。

示例

示例 1

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
GenericDB:
Generic Load *;
Load * inline [
Region, Attribute, Value

US, Name, AAA
US, Address, A123
US, Phone, 001-123

US, Name, BBB
US, Address, B456
US, Phone, 002-456
```

```
SWE, Name, CCC
SWE, Address, C7789
SWE, Phone, 003-789 ];
```

结果

结果表

区域	名称	Address	Phone
SWE	CCC	C7789	003-789
US	AAA	A123	001-123
US	AAA	A123	002-456
US	AAA	B456	001-123
US	AAA	B456	002-456
US	BBB	A123	001-123
US	BBB	A123	002-456
US	BBB	B456	001-123
US	BBB	B456	002-456

示例 2

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Sheet1:
Generic Load * inline [
object, attribute, value
ball, color, red
ball, diameter, 10 cm
ball, weight, 100 g
box, color, black
box, height, 16 cm
box, length, 20 cm
box, weight, 500 g
box, width, 10 cm ];
```

结果

结果表

对象	颜色	直径	长度	高度	宽度	重量
ball	red	10 cm	-	-	-	100 g
box	black	-	20 cm	16 cm	10 cm	500 g

Hierarchy

hierarchy 前缀用于将父子层次表格转换成在 Qlik Sense 数据模型中有用的表格。此前缀可能置于 **LOAD** 或 **SELECT** 语句前面，并会使用加载的语句结果作为表格转换的输入。

此前缀创建了一个扩展节点表格，通常其与输入的表格具有相同数目的记录，但除此之外，所有层次结构级别均存储于单独的字段内。路径字段可以在树结构中使用。

语法：

```
Hierarchy (NodeID, ParentID, NodeName, [ParentName, [PathSource, [PathName, [PathDelimiter, Depth]]]])(loadstatement | selectstatement)
```

输入表格必须为相邻节点表格。相邻表格内每个记录对应一个节点，并且拥有一个包含父节点参考的字段。此类表格内的节点存储在一个记录上，但节点仍拥有任意数量的子节点。表格可能包含更多描述节点属性的字段。

此前缀创建了一个扩展节点表格，通常其与输入的表格具有相同数目的记录，但除此之外，所有层次结构级别均存储于单独的字段内。路径字段可以在树结构中使用。

输入表格通常只有一个节点记录，此时输出表格包含相同的记录数。但是，节点有时会带有多个父节点，即一个节点由输入表格中的几个记录表示。在此情况下，输出表格拥有的记录可能多于输入表格。

未见于节点 ID 列且具父级 ID 的所有节点均会被视为根节点。此外，仅带有根节点连接(直接或间接)的节点会被加载，因此可避免循环引用。

更多包含父节点名称，节点路径和节点深度的字段会被创建。

参数：

参数

参数	说明
NodeID	包含节点 ID 的字段名称。此字段必须存在于输入表格中。
ParentID	包含父节点的节点 ID 的字段名称。此字段必须存在于输入表格中。
NodeName	包含节点名称的字段名称。此字段必须存在于输入表格中。
ParentName	即用于命名新建 **ParentName** 字段的字符串。如果省略，则无法创建此字段。
ParentSource	包含用于构建节点路径的节点名称的字段名称。可选参数。如果省略，则会使用 **NodeName**。
PathName	用于命名新建 **Path** 字段的字符串，该字段包含从根节点到节点的路径。可选参数。如果省略，则无法创建此字段。
PathDelimiter	在新建 **Path** 字段中用作分隔符的字符串。可选参数。如果省略，则会以 '/' 代替。

参数	说明
Depth	用于命名新建 **Depth** 字段的字符串，该字段包含层次结构中的节点深度。可选参数。如果省略，则无法创建此字段。

示例：

```
Hierarchy(NodeID, ParentID, NodeName, ParentName, NodeName, PathName, '\', Depth) LOAD *
inline [
NodeID, ParentID, NodeName
1, 4, London
2, 3, Munich
3, 5, Germany
4, 5, UK
5, , Europe
];
```

NodeID	ParentID	NodeName	NodeName1	NodeName2	NodeName3	ParentName	PathName	Depth
1	4	London	Europe	UK	London	UK	Europe\UK\London	3
2	3	Munich	Europe	Germany	Munich	Germany	Europe\Germany\Munich	3
3	5	Germany	Europe	Germany	-	Europe	Europe\Germany	2
4	5	UK	Europe	UK	-	Europe	Europe\UK	2
5		Europe	Europe	-	-	-	Europe	1

HierarchyBelongsTo

此前缀用于将父子层次表格转换成在 Qlik Sense 数据模型中有用的表格。此前缀可能置于 **LOAD** 或 **SELECT** 语句前面，并会使用加载的语句结果作为表格转换的输入。

此前缀可创建一个包含上下级层次结构关系的表格。上级字段随后可用于选择层次结构的整个树形结构。输出表格通常包含几个节点记录。

语法：

```
HierarchyBelongsTo (NodeID, ParentID, NodeName, AncestorID, AncestorName,
[DepthDiff])(loadstatement | selectstatement)
```

输入表格必须为相邻节点表格。相邻表格内每个记录对应一个节点，并且拥有一个包含父节点参考的字段。此类表格内的节点存储在一个记录上，但节点仍拥有任意数量的子节点。表格可能包含更多描述节点属性的字段。

此前缀可创建一个包含上下级层次结构关系的表格。上级字段随后可用于选择层次结构的整个树形结构。输出表格通常包含几个节点记录。

更多包含节点深度差异的字段会被创建。

参数：

参数

参数	说明
NodeID	包含节点 ID 的字段名称。此字段必须存在于输入表格中。
ParentID	包含父节点的节点 ID 的字段名称。此字段必须存在于输入表格中。
NodeName	包含节点名称的字段名称。此字段必须存在于输入表格中。
AncestorID	用于命名新建上级组件 ID 字段的字符串，该字段包含祖先节点的 ID。
AncestorName	用于命名新建上级字段的字符串，该字段包含祖先节点的名称。
DepthDiff	用于命名新 **DepthDiff** 字段的字符串，该字段包含层次结构中相对于祖先节点的节点深度。可选参数。如果省略，则无法创建此字段。

示例：

```
HierarchyBelongsTo (NodeID, AncestorID, NodeName, AncestorID, AncestorName, DepthDiff) LOAD *
inline [
NodeID, AncestorID, NodeName
1, 4, London
2, 3, Munich
3, 5, Germany
4, 5, UK
5, , Europe
];
```

Results

NodeID	AncestorID	NodeName	AncestorName	DepthDiff
1	1	London	London	0
1	4	London	UK	1
1	5	London	Europe	2
2	2	Munich	Munich	0
2	3	Munich	Germany	1
2	5	Munich	Europe	2
3	3	Germany	Germany	0
3	5	Germany	Europe	1
4	4	UK	UK	0
4	5	UK	Europe	1
5	5	Europe	Europe	0

Inner

可在 **join** 和 **keep** 前缀前面使用 **inner** 前缀。如果用于 **join** 之前，说明应使用内部联接。由此生成的表格仅包含原始数据表格(其中链接字段值在两个表格中均有呈现)的字段值组合。如果用于 **keep** 之前，说明在 Qlik Sense 中存储这些表格之前，首先应使两个原始数据表格缩减为它们的共同交集。

语法：

```
Inner ( Join | Keep) [ (tablename) ](loadstatement |selectstatement )
```

参数：

参数

参数	说明
tablename	可以将命名的表格与加载的表格进行比较。
loadstatement或 selectstatement	**LOAD** 或 **SELECT** 语句适用于加载的表格。

示例

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Table1: Load * inline [ Column1, Column2 A, B 1, aa 2, cc 3, ee ];  Table2: Inner Join Load * inline [ Column1, Column3 A, C 1, xx 4, yy ];
```

结果

结果表

Column1	Column2	Column3
A	B	C
1	aa	xx

解释

本例演示了内部联接输出，其中仅联接第一个(左)表和第二个(右)表中的值。

IntervalMatch

IntervalMatch 前缀用于创建表格以便将离散数值与一个或多个数值间隔进行匹配，并且任选匹配一个或多个额外关键值。

语法：

```
IntervalMatch (matchfield)(loadstatement | selectstatement )
IntervalMatch (matchfield,keyfield1 [ , keyfield2, ... keyfield5 ] )
(loadstatement | selectstatement )
```

IntervalMatch 前缀必须置于加载时间间隔的 **LOAD** 或 **SELECT** 语句之前。在使用此语句和 **IntervalMatch** 前缀之前，包含离散数据点的字段(以下所示的 Time)必须已经加载到 Qlik Sense。此前缀不会从数据库表格中读取此字段。此前缀将加载的时间间隔表格转换为包含其他列(离散数值数据点)的表格。另外其扩展了记录数，以使新表格对离散数据点、时间间隔和关键字段值的每个可能组合都有一条记录。

时间间隔可以重叠，离散值可以链接所有匹配的时间间隔。

使用关键字段扩展 IntervalMatch 前缀时，可用于创建既能将离散数值和一个或多个数值时间间隔匹配的表格，同时又能匹配一个或多个额外关键字段值。

要避免未定义的时间间隔限值遭到忽略，可能必须让 NULL 可以映射到构成时间间隔下限或上限的其他字段。这可通过 **NullAsValue** 语句或显式测试来处理，显式测试可在任何离散数值数据点前后使用数值很好地替代 NULL。

参数：

参数

参数	说明
matchfield	一个字段，包含链接至时间间隔的离散数值。
keyfield	一个字段，包含转换中匹配的额外属性。
loadstatement orselectstatement	必会生成一个表格，其中第一个字段包含每个时间间隔的下半部限制，第二个字段包含每个时间间隔的上半部限制，如果使用关键字匹配，则第三个及此后的任何字段包含显示于 **IntervalMatch** 语句中的关键字段值。时间间隔总是封闭区间，即间隔的端点包括在时间间隔之中。非数值限值会导致时间间隔遭到忽略(未定义)。

Example 1:

在以下两个表格中，第一个表格列出了众多离散事件，第二个表格定义了不同订单生产的开始时间和结束时间。借助 **IntervalMatch** 前缀，可以逻辑连接两个表格，从而找出哪些订单受到干扰的影响，哪些订单依据哪次轮班处理。

```
EventLog:
LOAD * Inline [
Time, Event, Comment
00:00, 0, Start of shift 1
01:18, 1, Line stop
02:23, 2, Line restart 50%
04:15, 3, Line speed 100%
08:00, 4, Start of shift 2
11:43, 5, End of production
];

OrderLog:
LOAD * INLINE [
Start, End, Order
01:00, 03:35, A
02:30, 07:58, B
03:04, 10:27, C
```

```
07:23, 11:43, D
];

//Link the field Time to the time intervals defined by the fields Start and End.
Inner Join IntervalMatch ( Time )
LOAD Start, End
Resident OrderLog;
```

表格 **OrderLog** 现在包含额外一列：*Time*。记录的数量也可以扩展。

Table with additional column

Time	Start	End	Order
00:00	-	-	-
01:18	01:00	03:35	A
02:23	01:00	03:35	A
04:15	02:30	07:58	B
04:15	03:04	10:27	C
08:00	03:04	10:27	C
08:00	07:23	11:43	D
11:43	07:23	11:43	D

Example 2: (使用 keyfield)

与上述示例相同，添加 *ProductionLine* 作为关键字段。

```
EventLog:
LOAD * Inline [
Time, Event, Comment, ProductionLine
00:00, 0, Start of shift 1, P1
01:00, 0, Start of shift 1, P2
01:18, 1, Line stop, P1
02:23, 2, Line restart 50%, P1
04:15, 3, Line speed 100%, P1
08:00, 4, Start of shift 2, P1
09:00, 4, Start of shift 2, P2
11:43, 5, End of production, P1
11:43, 5, End of production, P2
];

OrderLog:
LOAD * INLINE [
Start, End, Order, ProductionLine
01:00, 03:35, A, P1
02:30, 07:58, B, P1
03:04, 10:27, C, P1
07:23, 11:43, D, P2
];
```

```
//Link the field Time to the time intervals defined by the fields Start and End and match the
values
// to the key ProductionLine.
Inner Join
IntervalMatch ( Time, ProductionLine )
LOAD Start, End, ProductionLine
Resident OrderLog;
```

现在可以按照以下方式创建表格框：

Tablebox example

ProductionLine	Time	Event	Comment	Order	Start	End
P1	00:00	0	Start of shift 1	-	-	-
P2	01:00	0	Start of shift 1	-	-	-
P1	01:18	1	Line stop	A	01:00	03:35
P1	02:23	2	Line restart 50%	A	01:00	03:35
P1	04:15	3	Line speed 100%	B	02:30	07:58
P1	04:15	3	Line speed 100%	C	03:04	10:27
P1	08:00	4	Start of shift 2	C	03:04	10:27
P2	09:00	4	Start of shift 2	D	07:23	11:43
P1	11:43	5	End of production	-	-	-
P2	11:43	5	End of production	D	07:23	11:43

Join

join 前缀可连接加载的表格和现有已命名的表格或最近创建的数据表。

语法：

```
[inner | outer | left | right ]Join [ (tablename ) ]( loadstatement |
selectstatement )
```

该联接是所有共同字段之间的自然联接。Join 语句可位于 **inner**, **outer**, **left** 或**right** 等前缀的前面。

参数：

参数

参数	说明
tablename	可以将命名的表格与加载的表格进行比较。
loadstatement或 selectstatement	**LOAD** 或 **SELECT** 语句适用于加载的表格。

示例：

```
Join SELECT * from table1;
```

```
tab1:

LOAD * from file1.csv;

tab2:

LOAD * from file2.csv;

.. .. ..

join (tab1) LOAD * from file3.csv;
```

示例

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Table1: Load * inline [ Column1, Column2 A, B 1, aa 2, cc 3, ee ];  Table2: Join Load * inline [ Column1, Column3 A, C 1, xx 4, yy ];
```

结果表

Column1	Column2	Column3
A	B	C
1	aa	xx
2	cc	-
3	ee	-
4	-	yy

解释

在本例中，Table1 和 Table2 这两个表合并为一个标记为 Table1 的表。在这种情况下，**join** 前缀通常用于将多个表连接到一个表中，以对单个表的值执行聚合。

Keep

keep 前缀类似于 **join** 前缀。与 **join** 前缀一样，该前缀可用来将加载的表格与现有的命名表格或最后一个之前创建的数据表格进行比较，而不是将加载的表格与现有的表格进行合并，它可以在将表格存储在 Qlik Sense 中之前，根据表格数据的交集减少一个或同时减少两个表格。这种比较相当于对所有共同字段进行自然联接，即等同于相应联接的方式。但是，这两个表格并未合并，而将作为两个单独命名的表格保留在 Qlik Sense 中。

语法：

```
(inner | left | right) keep [(tablename ) ]( loadstatement | selectstatement )
```

keep 前缀必须置于 **inner**、**left** 或 **right** 前缀之一的前面。

Qlik Sense 脚本语言中的显式 **join** 前缀可完全联接这两个表格。结果会生成一个表格。在许多情况下，这种联接将产生很大的表格。Qlik Sense 的主要功能之一是使多个表格之间关联，而不是联接这些表格，这种关联可以大大减少占用的内存，提高处理速度并且灵活多变。因此，在 Qlik Sense 脚本中一般应避免使用显式联接。保存功能旨在减少需要使用显式联接的情况。

参数：

参数

参数	说明
tablename	可以将命名的表格与加载的表格进行比较。
loadstatement或 selectstatement	**LOAD** 或 **SELECT** 语句适用于加载的表格。

示例：

```
Inner Keep LOAD * from abc.csv;
Left Keep SELECT * from table1;
tab1:
LOAD * from file1.csv;
tab2:
LOAD * from file2.csv;
.. .. ..
Left Keep (tab1) LOAD * from file3.csv;
```

Left

可在 **Join** 和 **Keep** 前缀前面使用 **left** 前缀。

如果用于 **join** 之前，说明应使用左侧联接。由此生成的表格仅包含原始数据表格的字段值组合，在原始数据表格中，链接字段值呈现在第一个表格中。如果用于 **keep** 之前，说明首先应使第二原始数据表格缩减为其与第一表格间的共同交集，然后才可在 Qlik Sense 中存储此表格。

是否按同一名称查找字符串函数？请参阅：Left (page 734)

语法：

```
Left ( Join | Keep) [ (tablename) ](loadstatement | selectstatement)
```

参数：

参数

参数	说明
tablename	可以将命名的表格与加载的表格进行比较。
loadstatement或 selectstatement	**LOAD** 或 **SELECT** 语句适用于加载的表格。

示例

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Table1: Load * inline [ Column1, Column2 A, B 1, aa 2, cc 3, ee ];  Table2: Left Join Load * inline [ Column1, Column3 A, C 1, xx 4, yy ];
```

结果

结果表

Column1	Column2	Column3
A	B	C
1	aa	xx
2	cc	-
3	ee	-

解释

本例演示了左联接输出，其中仅联接第一个(左)表中的值。

Mapping

mapping 前缀用于创建映射表，例如，此映射表在脚本运行期间可用于替换字段值和字段名。

语法：

```
Mapping( loadstatement | selectstatement )
```

该 **mapping** 前缀可置于 **LOAD** 或 **SELECT** 语句之前，并将正在加载的语句的结果存储为映射表。映射提供了一种有效的途径在脚本执行过程中替代字段值，例如：将 US、U.S. 或替换为 USA。映射表必须有两个字段，第一个字段包含比较值，第二个字段包含所需的映射值。映射表暂时存储在内存中，执行脚本后将自动删除。

使用 **Map ... Using** 语句、**Rename Field** 语句、**Applymap()** 函数或 **Mapsubstring()** 函数可访问映射表的内容。

示例：

在此例中，我们加载了销售人员和国家代码(表示销售人员所居住的国家)的列表。我们使用表格将国家代码映射到国家，以便将国家代码替换为国家名称。在映射表中仅定义了三个国家，其他国家代码已映射到'Rest of the world'。

```
// Load mapping table of country codes:
map1:
mapping LOAD *
Inline [
CCode, Country
```

```
Sw, Sweden
Dk, Denmark
No, Norway
] ;
// Load list of salesmen, mapping country code to country
// If the country code is not in the mapping table, put Rest of the world
Salespersons:
LOAD *,
ApplyMap('map1', CCode,'Rest of the world') As Country
Inline [
CCode, Salesperson
Sw, John
Sw, Mary
Sw, Per
Dk, Preben
Dk, Olle
No, Ole
Sf, Risttu] ;
// We don't need the CCode anymore
Drop Field 'CCode';
```

最终生成的表格如下所示：

Mapping table

Salesperson	Country
John	Sweden
Mary	Sweden
Per	Sweden
Preben	Denmark
Olle	Denmark
Ole	Norway
Risttu	Rest of the world

合并

可将前缀 **Merge** 添加至脚本中的任何 **LOAD** 或 **SELECT** 语句，以指定加载的表格应当合并到另一表中。它还指定此语句应在部分重新加载中运行。

典型的用例是当您加载更改日志并希望用此来将 inserts、updates 和 deletes 应用到现有的表时。

> *要使部分重新加载正常工作，必须在触发部分重新加载之前使用数据打开应用程序。*

使用**重新加载**按钮执行部分重新加载。您还可使用Qlik Engine JSON API。

语法：

```
Merge [only] [(SequenceNoField [, SequenceNoVar])] On ListOfKeys [Concatenate
[(TableName)]] (loadstatement | selectstatement)
```

参数:

参数

参数	描述
only	可选限定符表示仅应在部分重新加载期间执行语句。在正常(非部分)重新加载期间,忽略此语句。
SequenceNoField	包含定义操作顺序的时间戳或序列号的字段的名称。
SequenceNoVar	为要合并的表分配 SequenceNoField 最大值的变量的名称。
ListOfKeys	指定主键的字段名的逗号分隔列表。
Operation	LOAD 语句的第一个字段必须包含操作作为文本字符串:'Insert'、'Update' 或 'Delete'。也接受 'i'、'u' 和 'd'

一般功能

在正常(非部分)重新加载期间,**Merge LOAD** 构造用作普通的 **Load** 语句,但是具有额外的功能,用于移除旧的过时记录和标记为删除的记录。**LOAD** 语句的第一个字段必须包含有关操作的信息:Insert、Update 或 Delete。

对于每个加载的记录,将记录标识符与以前加载的记录进行比较,并且只保留最新的记录(根据序列号)。如果最新记录标有 Delete,则不保留任何记录。

目标表格

要修改的表由字段集决定。如果已经存在具有相同字段集(第一个字段除外;操作)的表,则该表将是要修改的相关表。或者,可以使用**连接**前缀来指定表。如果未确定目标表,则 **Merge LOAD** 构造的结果将存储在新表中。

如果使用了连接前缀,则生成的表具有一组字段,这些字段对应于现有表和待合并输入的并集。因此,目标表可能会获得比用作合并输入的更改日志更多的字段。

局部重新加载有和完全重新加载相同的作用。一项区别是,部分重新加载很少创建新表。除非使用了 **Only** 子句,否则始终存在字段集与上一次脚本执行中字段集相同的目标表。

序号

如果加载的更改日志是累积日志,即它包含已加载的更改,则可以在 **Where** 子句中使用该参数 SequenceNoVar 来限制输入数据量。**Merge LOAD** 然后可仅用于加载记录,其中字段 SequenceNoField 大于 SequenceNoVar。在完成之后,**Merge LOAD** 把新的值分配至 SequenceNoVar,其中最大值在 SequenceNoField 字段中可见。

运算

Merge LOAD 字段可以少于目标表。不同的操作对缺失字段的处理方式不同:

插入:**Merge LOAD** 中缺少但存在于目标表中的字段在目标表中获取空值。

删除:缺少字段不会影响结果。相关记录仍将被删除。

更新:**Merge LOAD** 中列出的字段将在目标表中更新。缺少的字段不会更改。这意味着以下两个语句不完全相同:

- Merge on Key Concatenate Load 'U' as Operation, Key, F1, Null() as F2 From ...;
- Merge on Key Concatenate Load 'U' as Operation, Key, F1 From ...;

第一条语句更新列出的记录,并将 F2 更改为 NULL。第二个不改变 F2,而是将值留在目标表中。

Merge LOAD 不能用于带有通配符的记录,例如,带有星号表示所有值的 Section Access 表。

示例

示例 1:与指定表的简单合并

在该示例中,将加载具有三行的名为 Persons 的内联表格。**Merge** 之后如下更改表格:

- 添加行 *Mary, 4*。
- 删除行 *Steven, 3*。
- 将数字 *5* 分配给 *Jake*。

LastChangeDate 变量在执行 **Merge** 之后设置为 *ChangeDate* 列中的最大值。

加载脚本

将示例脚本添加到应用程序并运行。要查看结果,将结果列中列出的字段添加到应用程序中的工作表。

```
Set DateFormat='D/M/YYYY'; Persons: load * inline [ Name, Number Jake, 3 Jill, 2 Steven, 3 ];
Merge (ChangeDate, LastChangeDate) on Name Concatenate(Persons) LOAD * inline [ Operation,
ChangeDate,   Name,     Number Insert,    1/1/2021,     Mary,     4 Delete,    1/1/2021,
Steven,  Update,    2/1/2021,     Jake,     5 ];
```

结果

在 **Merge Load** 加载之前,所得表格如下显示:

Resulting table

Name	Number
Jake	3
Jill	2
Steven	3

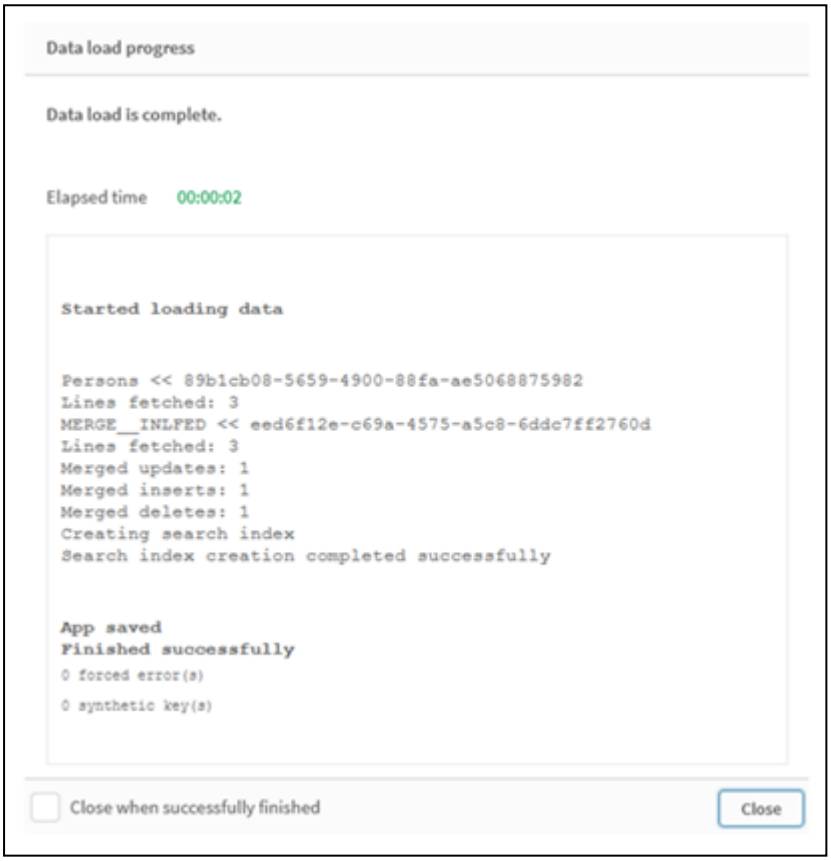
在 **Merge Load** 之后,表格如下显示:

Resulting table

ChangeDate	Name	Number
2/1/2021	Jake	5
-	Jill	2
1/1/2021	Mary	4

在加载数据之后，**数据加载进度**对话框显示执行的操作：

数据加载进度对话框



示例 2:缺少字段的数据加载脚本

在本例中，加载了与上述相同的数据，但现在每个人都有一个 ID。

Merge 如下更改表格：

- 添加行 *Mary, 4*。
- 删除行 *Steven, 3*。
- 将数字 *5* 分配给 *Jake*。
- 将数字 *6* 分配给 *Jill*。

加载脚本

这里我们使用两个 **Merge Load** 语句，一个用于‘Insert’和‘Delete’，另一个用于‘Update’。

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Set DateFormat='D/M/YYYY'; Persons: Load * Inline [ PersonID, Name, Number 1, Jake, 3 2, Jill, 2 3, Steven, 3 ];  Merge (ChangeDate, LastChangeDate) on PersonID Concatenate(Persons) Load * Inline [ Operation, ChangeDate,   PersonID, Name,     Number Insert,    1/1/2021,     4,  Mary,     4 Delete,    1/1/2021,     3,        Steven,  ];  Merge (ChangeDate, LastChangeDate) on PersonID Concatenate(Persons) Load * Inline [ Operation, ChangeDate, PersonID, Number Update,    2/1/2021,     1,        5 Update,    3/1/2021,     2,        6 ];
```

结果

在 **Merge Load** 语句之后，表格如下显示：

Resulting table

PersonID	ChangeDate	Name	Number
1	2/1/2021	Jake	5
2	3/1/2021	Jill	6
4	1/1/2021	Mary	4

请注意，第二个 **Merge** 语句不包括字段 **Name**，因此，名称没有更改。

示例 3: 数据加载脚本 - 使用带有 ChangeDate 的 Where 子句部分重新加载

在下面的示例中，**Only** 参数指定仅在部分重新加载期间执行 **Merge** 命令。更新根据先前捕获的 LastChangeDate 进行过滤。在完成 **Merge** 之后，LastChangeDate 变量被分配合并期间处理的 ChangeDate 列的最大值

加载脚本

```
Merge Only (ChangeDate, LastChangeDate) on Name Concatenate(Persons) LOAD Operation, ChangeDate,  Name, Number  from [lib://ChangeFilesFolder/BulkChangesInPersonsTable.csv] (txt) where ChangeDate >= $(LastChangeDate);
```

NoConcatenate

NoConcatenate 前缀强制将两个使用相同字段集的加载表格处理为两个单独的内部表格(当它们以其他方式自动串联时)。

语法：

```
NoConcatenate( loadstatement | selectstatement )
```

示例：

```
LOAD A,B from file1.csv;
NoConcatenate LOAD A,B from file2.csv;
```

Only

可以将 **Only** 脚本关键字用作聚合函数，或者在部分重新加载前缀 **Add**、**Replace** 和 **Merge** 中作为语法的一部分。

Outer

显式 **Join** 前缀前面可带前缀 **Outer** 以指定外部联接。在外部联接中，生成两个表格之间的所有组合。由此生成的表格包含原始数据表格(其中链接字段值在两个表格中均有呈现)的字段值组合。**Outer** 关键字是可选型，并用作未指定联接前缀时的默认联接类型。

语法：

```
Outer Join [ (tablename) ](loadstatement |selectstatement )
```

参数：

参数

参数	说明
tablename	可以将命名的表格与加载的表格进行比较。
loadstatement或 selectstatement	**LOAD** 或 **SELECT** 语句适用于加载的表格。

示例

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Table1: Load * inline [ Column1, Column2 A, B 1, aa 2, cc 3, ee ];  Table2: Outer Join Load * inline [ Column1, Column3 A, C 1, xx 4, yy ];
```

结果表

Column1	Column2	Column3
A	B	C
1	aa	xx
2	cc	-
3	ee	-
4	-	yy

解释

在本例中，Table1 和 Table2 这两个表合并为一个标记为 Table1 的表。在这种情况下，**外部**前缀通常用于将多个表连接到一个表中，以对单个表的值执行聚合。

部分加载

完全重新加载总是从删除现有数据模型中的所有表开始，然后运行加载脚本。

局部重新加载没有该作用。相反，它会保留数据模型中的所有表格，然后仅执行 **Load** 和 **Select** 语句，这些语句带有前缀 **Add**、**Merge** 或 **Replace** 前缀。其他数据表不受该命令的影响。**only** 参数表示只应在部分重新加载期间执行该语句，而在完全重新加载期间应忽略该语句。下表总结了部分和完全重新加载的语句执行情况。

语句	完全重新加载	部分加载
Load ...	语句将会运行	语句将不会运行
Add/Replace/Merge Load ...	语句将会运行	语句将会运行
Add/Replace/Merge Only Load ...	语句将不会运行	语句将会运行

与完全重新加载相比，部分重新加载有几个好处：

- 速度更快，因为只需要加载最近更改的数据。对于大数据集，差异是显著的。
- 因为加载的数据更少，所以消耗的内存更少。
- 更可靠，因为对源数据的查询运行速度更快，减少了网络问题的风险。

要使部分重新加载正常工作，必须在触发部分重新加载之前使用数据打开应用程序。

使用**重新加载**按钮执行部分重新加载。您还可使用Qlik Engine JSON API。

示例

示例 1

加载脚本

将示例脚本添加到应用程序并进行部分重新加载。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
T1: Add only Load distinct recno()+10 as Num autogenerate 10;
```

结果

Resulting table

Num	Count(Num)
11	1

Num	Count(Num)
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1

解释

该语句仅在部分重新加载期间执行。如果省略了“distinct”前缀，则 **Num** 字段的计数将随着后续每次部分重新加载而增加。

示例 2

加载脚本

将示例脚本添加到应用程序。进行完全重新加载并查看结果。接下来，进行部分重新加载并查看结果。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
 T1: Load recno() as ID, recno() as Value autogenerate 10;  T1: Replace only Load recno() as ID, repeat(recno(),3) as Value autogenerate 10;
```

结果

Output table after full reload

ID	Value
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Output table after partial reload

ID	Value
1	111
2	222
3	333
4	444
5	555
6	666
7	777
8	888
9	999
10	101010

解释

第一个表在完全重新加载期间加载，第二个表在部分重新加载期间仅替换第一个表。

Replace

可以将 **Replace** 脚本关键字用作字符串函数，或者在部分重新加载中用作前缀。

Replace

Replace前缀 可添加至脚本中的任何 **LOAD** 或 **SELECT** 语句，以指定加载的表格应当替代另一表格。它还指定此语句应在部分重新加载中运行。**Replace** 前缀还可用在 **Map** 语句中。

要使部分重新加载正常工作，必须在触发部分重新加载之前使用数据打开应用程序。

使用**重新加载**按钮执行部分重新加载。您还可使用Qlik Engine JSON API。

语法：

```
Replace [only] [Concatenate[(tablename)]] (loadstatement | selectstatement)
```

```
Replace [only] mapstatement
```

在正常(非部分) 重新加载期间，**Replace LOAD** 构造将作为普通**LOAD** 语句作用，但是将由 **Drop Table** 替代。首先删除旧表，然后生成记录并作为新表存储。

如果使用了 **Concatenate** 前缀，或者存在具有相同字段集的表，则此项将为要放弃的相关表格。否则，将没有要放弃的表，并且 **Replace LOAD** 构造将与普通 **LOAD** 一致。

局部重新加载有相同作用。唯一的区别是，上一个脚本执行中总是有一个表要删除。**Replace LOAD** 构造总是首先删除旧表，然后创建新表。

Replace Map...Using 语句在部分脚本执行期间也会导致映射发生。

参数：

参数

参数	说明
only	可选限定符表示仅应在部分重新加载期间执行语句。在正常(非部分)重新加载期间，应忽略此项。

示例和结果：

示例	结果

`Tab1:`
`Replace LOAD *`
`from File1.csv;`
	在常规和部分重新加载期间，Qlik Sense 表格 Tab1 起初会被删除。此后，将会从 File1.csv 加载新数据，并存储到 Tab1。

`Tab1:`
`Replace only LOAD`
`* from File1.csv;`
	在常规重新加载期间，此语句会被忽略。
	
	在部分重新加载期间，任意 Qlik Sense 表格以前命名的 Tab1 起初会被删除。此后，将会从 File1.csv 加载新数据，并存储到 Tab1。

`Tab1:`
`LOAD a,b,c from`
`File1.csv;`
`Replace LOAD`
`a,b,c from`
`File2.csv;`
	在常规重新加载期间，文件 File1.csv 首先会读取至 Qlik Sense 表格 Tab1，然后立即删除并由从 File2.csv 加载的新数据替换。来自 File1.csv 的全部数据会丢失。
	
	在部分重新加载期间，整个 Qlik Sense 表格 Tab1 起初会被删除。此后，使用从 File2.csv 加载的新数据进行替换。

`Tab1:`
`LOAD a,b,c from`
`File1.csv;`
`Replace only LOAD`
`a,b,c from`
`File2.csv;`
	常规重新加载期间，将会从 File1.csv 加载数据，并存储到 Qlik Sense 表格 Tab1 中。File2.csv 会被忽略。
	
	在部分重新加载期间，整个 Qlik Sense 表格 Tab1 起初会被删除。此后，使用从 File2.csv 加载的新数据进行替换。来自 File1.csv 的全部数据会丢失。

Right

可在 **Join** 和 **Keep** 前缀前面使用 **right** 前缀。

如果用于 **join** 之前，说明应使用右侧联接。由此生成的表格仅包含原始数据表格的字段值组合，原始数据表格中的链接字段值呈现在第二个表格中。如果用于 **keep** 之前，说明首先应使第一原始数据表格缩减为其与第二表格间的共同交集，然后才可在 Qlik Sense 中存储此表格。

是否按同一名称查找字符串函数？请参阅：Right (page 742)

语法：

```
Right (Join | Keep) [(tablename)](loadstatement |selectstatement )
```

参数：

参数

参数	说明
tablename	可以将命名的表格与加载的表格进行比较。
loadstatement或 selectstatement	**LOAD** 或 **SELECT** 语句适用于加载的表格。

示例

加载脚本

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Table1: Load * inline [ Column1, Column2 A, B 1, aa 2, cc 3, ee ];  Table2: Right Join Load * inline [ Column1, Column3 A, C 1, xx 4, yy ];
```

结果

结果表

Column1	Column2	Column3
A	B	C
1	aa	xx
4	-	yy

解释

本例演示了右联接输出，其中仅联接第二个(右)表中的值。

Sample

LOAD 或 **SELECT** 语句的 **sample** 前缀用于从数据源载入记录的随机样本。

语法：

```
Sample  p ( loadstatement | selectstatement )
```

参数：

参数

参数	说明
p	用于评估大于 0，且小于或等于 1 的数字的任意表达式。该数字表示未来读取给定记录的可能性。 所有记录都将被读取，但只有其中的一部分会加载到 Qlik Sense 中。

示例：

```
Sample 0.15 SQL SELECT * from Longtable;
Sample(0.15) LOAD * from Longtab.csv;
```

括号可有可无。

Semantic

可通过 **semantic** 前缀加载包含两个记录之间关系的表格。例如，这可以是表格内的自引用，即其中一个记录指向另一个记录，如所属的父项或祖先。

语法：

```
Semantic( loadstatement | selectstatement)
```

Semantic 加载将会创建显示在筛选器窗格中用于导航数据的语义字段。

不能串联通过 **semantic** 语句加载的表格。

示例：

```
Semantic LOAD * from abc.csv;
Semantic SELECT Object1, Relation, Object2, InverseRelation from table1;
```

Unless

unless 前缀和后缀用于创建确定是否应计算语句或 exit 子句的条件子句。它可以被看作是完整的 **if..end if** 语句的简洁替代形式。

语法：

```
(Unless condition statement | exitstatement Unless condition )
```

如果**condition**评估结果为 False，则仅将执行 **statement** 或 **exitstatement**。

unless 前缀可以在已有一个或多个其他语句的语句中使用，包括其他 **when** 或 **unless** 前缀。

参数：

参数

参数	说明
condition	用于评估 True 或 False 的逻辑表达式。
statement	任意 Qlik Sense 脚本语句，不包括控制语句。
exitstatement	**exit for**、**exit do** 或 **exit sub** 子句或 **exit script** 语句。

示例：

```
exit script unless A=1;
unless A=1 LOAD * from myfile.csv;
unless A=1 when B=2 drop table Tab1;
```

When

when 前缀和后缀用于创建确定是否应执行语句或 **exit** 子句的条件子句。它可以被看作是完整的 **if..end if** 语句的简洁替代形式。

语法：

```
(when condition statement | exitstatement when condition )
```

如果评估条件为 True，则将仅执行 **statement** 或 **exitstatement**。

when 前缀可以在已有一个或多个其他语句的语句中使用，包括其他 **when** 或 **unless** 前缀。

语法：

参数

参数	说明
condition	用于评估 True 或 False 的逻辑表达式。
statement	任意 Qlik Sense 脚本语句，不包括控制语句。
exitstatement	**exit for**、**exit do** 或 **exit sub** 子句或 **exit script** 语句。

Example 1:

```
exit script when A=1;
```

Example 2:

```
when A=1 LOAD * from myfile.csv;
```

Example 3:

```
when A=1 unless B=2 drop table Tab1;
```

2.5 脚本常规语句

常规语句通常用于以某种方式或其他方式操作数据。这些语句可能被脚本中的行编号覆盖且必须总是以分号“;”终止。

所有脚本关键字可以大小写字符的任意组合输入。用于脚本中的字段和变量名要区分大小写。

脚本常规语句概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Alias

alias 语句用于设置别名。根据此别名，每当后面的脚本中出现相应字段时其都会重命名。

```
Alias fieldname as aliasname {,fieldname as aliasname}
```

Autonumber

此语句为脚本执行期间遇到的字段中每个不同的计算值创建唯一的整数值。

```
AutoNumber fields [Using namespace] ]
```

Binary

binary 语句用于加载另一个 QlikView 文档的数据，包括区域权限数据。

```
Binary  [path] filename
```

comment

提供一种从数据库和电子表格显示表格注释(元数据)的方式。可以忽略在应用程序中不显示的字段名。如果有字段名多次出现，将使用最后出现的值。

```
Comment field *fieldlist using mapname
Comment field fieldname with comment
```

comment table

提供一种从数据库或电子表格显示表格注释(元数据)的方式。

```
Comment table tablelist using mapname
Comment table tablename with comment
```

Connect

该功能在 Qlik Sense SaaS 中不可用。

CONNECT 语句用于定义 Qlik Sense 通过 OLE DB/ODBC 接口访问通用数据库。对于 ODBC，首先需要用 ODBC 管理员指定数据源。

```
ODBC Connect TO connect-string [ ( access_info ) ]
OLEDB CONNECT TO connect-string [ ( access_info ) ]
CUSTOM CONNECT TO connect-string [ ( access_info ) ]
LIB CONNECT TO connection
```

Declare

Declare 语句用于创建字段定义，您可以在其中定义字段或函数之间的关系。可以使用一组字段定义来自动生成可用作维度的导出字段。例如，您可以创建日历定义，并用它来从日期字段生成相关的维度，如年份、月份、周和日期。

```
definition_name:
Declare [Field[s]] Definition [Tagged tag_list ]
[Parameters parameter_list ]
Fields field_list
[Groups group_list ]

<definition name>:
Declare [Field][s] Definition
Using <existing_definition>
[With <parameter_assignment> ]
```

Derive

Derive 语句用于根据通过使用 **Declare** 语句创建的字段定义生成导出字段。您可以指定要为其导出字段的数据字段，或者根据字段标签明确或默认导出它们。

```
Derive [Field[s]] From [Field[s]] field_list Using definition
Derive [Field[s]] From Explicit [Tag[s]] (tag_list) Using definition
Derive [Field[s]] From Implicit [Tag[s]] Using definition
```

Direct Query

DIRECT QUERY 语句可让您使用 Direct Discovery 函数通过 ODBC 或 OLE DB 连接访问表格。

```
Direct Query [path]
```

Directory

Directory 语句用于定义在后续 **LOAD** 语句中查找数据文件的目录，直到出现新的 **Directory** 语句。

```
Directory [path]
```

Disconnect

Disconnect 语句用于终止当前 ODBC/OLE DB/自定义连接。此语句为可选。

```
Disconnect
```

drop field

在执行脚本期间，可以使用 **drop field** 语句随时从数据模型和内存中删除一个或多个 Qlik Sense 字段。

> ***drop field*** *和* ***drop fields*** *都是允许的格式，效果完全一样。如果未指定任何表格，字段将从全部表格中删除。*

```
Drop field  fieldname [ , fieldname2 ...] [from tablename1 [ , tablename2
...]]
drop fields fieldname [ , fieldname2 ...] [from tablename1 [ , tablename2
...]]
```

drop table

在执行脚本期间，可以使用 **drop table** 语句随时从数据模型和内存中删除一个或多个 Qlik Sense 内部表格。

*可以同时接受 **drop table** 和 **drop tables** 格式。*

```
Drop table  tablename [, tablename2 ...]
drop tables[ tablename [, tablename2 ...]
```

Execute

Execute 语句用于在 Qlik Sense 加载数据的同时运行其他程序。例如，需要执行转换。

```
Execute commandline
```

FlushLog

FlushLog 语句用于强制 Qlik Sense 将脚本缓冲区的内容写入脚本日志文件。

```
FlushLog
```

Force

force 语句用于强制 Qlik Sense 将后面的 **LOAD** 和 **SELECT** 语句的字段名称和字段值写入方式解释为仅限大写字母，仅限小写字母，总是首字母大写或它们的原初显示形式(大小写混合)。此语句可以根据不同的惯例关联表格的字段值。

```
Force ( capitalization | case upper | case lower | case mixed )
```

LOAD

LOAD 语句可以加载以下来源的字段：文件、脚本中定义的数据、预先载入的输入表格、网页、后续 **SELECT** 语句产生的结果或自动生成的数据。还可从分析连接加载数据。

```
 Load [ distinct ] *fieldlist
[( from file [ format-spec ] |
from_field fieldassource [format-spec]
inline data [ format-spec ] |
resident table-label |
autogenerate size )]
[ where criterion | while criterion ]
[ group_by groupbyfieldlist ]
[order_by orderbyfieldlist ]
[extension pluginname.functionname(tabledescription)]
```

Let

let 语句是 **set** 语句的补充，用于定义脚本变量。相对于 **set** 语句，**let** 语句在其被分配到变量之前可以在脚本运行时计算等号“=”右边的表达式。

```
Let variablename=expression
```

Loosen Table

在使用 **Loosen Table** 语句执行脚本期间，一个或多个 Qlik Sense 内部数据表格可以明确声明松散耦合。当表格是松散耦合时，表格中字段值之间的所有关联都会被移除。通过独立加载松散耦合表格(未相互连接的表格)的每个字段可以达到类似的效果。在对数据结构临时独立的不同部分进行测试时，松散耦合会非常有用。松散耦合表格在表格查看器中将以虚线进行标识。执行脚本之前，

在脚本中使用一个或多个 **Loosen Table** 语句将使 Qlik Sense 忽略任何松散耦合表设置。

```
tablename [ , tablename2 ...]
Loosen Tables tablename [ , tablename2 ...]
```

Map ... using

map ... using 语句用于将某些字段值或表达式映射为特定映射表的值。映射表可通过 **Mapping** 语句创建。

```
Map  *fieldlist Using  mapname
```

NullAsNull

NullAsNull 语句用于关闭 NULL 语句之前设置的从 **NullAsValue** 值到字符串值的转换。

```
NullAsNull *fieldlist
```

NullAsValue

NullAsValue 语句用于指定应将 NULL 值转换为值的字段。

```
NullAsValue *fieldlist
```

Qualify

Qualify 语句用于打开字段名限制条件，即字段名将表格名作为前缀。

```
Qualify *fieldlist
```

Rem

rem 语句用于插入备注或注释到脚本，或暂时关闭脚本语句而无需移除脚本。

```
Rem string
```

Rename Field

此脚本函数用于在加载一个或多个现有 Qlik Sense 字段后对其进行重命名。

```
Rename field (using mapname|oldname to newname{ , oldname to newname })
```

```
Rename Fields (using mapname | oldname to newname{ , oldname to newname })
```

Rename Table

此脚本函数用于在加载一个或多个现有 Qlik Sense 内部表格后对其进行重命名。

```
Rename table  (using mapname | oldname to newname{ , oldname to newname })
Rename Tables (using mapname | oldname to newname{ , oldname to newname })
```

Section

使用 **section** 语句，可以定义随后的 **LOAD** 和 **SELECT** 语句应视为数据还是访问权限定义。

```
Section (access | application)
```

Select

可通过标准 SQL **SELECT** 语句从 ODBC 数据源或 OLE DB 提供商选择字段。然而，是否接受 **SELECT** 语句取决于所使用的 ODBC 驱动程序或 OLE DB 提供程序。

```
Select [all | distinct | distinctrow | top n [percent] ] *fieldlist

From tablelist

[Where criterion ]

[Group by fieldlist [having criterion ] ]

[Order by fieldlist [asc | desc] ]

[ (Inner | Left | Right | Full)Join tablename on fieldref = fieldref ]
```

Set

set 语句用于定义脚本变量。这些变量可用来替代字符串，路径和驱动程序等。

```
Set variablename=string
```

Sleep

sleep 语句用于在指定的时间暂停脚本执行。

```
Sleep n
```

SQL

SQL 语句可通过 ODBC 或 OLE DB 连接发送任意 SQL 命令。

```
SQL sql_command
```

SQLColumns

sqlcolumns 语句用于返回描述 ODBC 或 OLE DB 数据源的列以建立 **connect** 的字段集。

```
SQLColumns
```

SQLTables

sqltables 语句用于返回描述 ODBC 或 OLE DB 数据源的表格以建立 **connect** 的字段集。

```
SQLTables
```

SQLTypes

sqltypes 语句用于返回描述 ODBC 或 OLE DB 数据源的类型以建立 **connect** 的字段集。

```
SQLTypes
```

Star

该字符串用于呈现数据库中字段的全部值设置，可以通过 **star** 语句设置。星号可以影响随后的 **LOAD** 和 **SELECT** 语句。

```
Star is [ string ]
```

Store

Store 语句创建 QVD、CSV 或 text 文件。

```
Store [ *fieldlist from] table into filename [ format-spec ];
```

Tag

此脚本语句提供了一种将标记分配给一个或多个字段或表格的方法。如果试图标记应用程序中不存在的字段或表格，则将忽略该标记。如果出现字段名和标记名冲突的情况，将使用最后出现的值。

```
Tag[field|fields] fieldlist with tagname
Tag [field|fields] fieldlist using mapname
Tag table tablelist with tagname
```

Trace

trace 语句用于将字符串写入**脚本执行进度**窗口和脚本日志文件中。这对调试过程很有用。使用在 **trace** 语句之前计算的 **$** 变量扩展可以自定义消息。

```
Trace string
```

Unmap

Unmap 语句可禁用由之前的 **Map ... Using** 语句为后来加载的字段指定的字段值映射。

```
Unmap *fieldlist
```

Unqualify

Unqualify 语句用于断开前面由 **Qualify** 语句打开的字段名限制条件。

```
Unqualify *fieldlist
```

Untag

此脚本语句提供了一种从字段或表中删除标记的方法。如果试图取消标记应用程序中不存在的字段或表格，则将忽略该取消标记。

```
Untag[field|fields] fieldlist with tagname
Tag [field|fields] fieldlist using mapname
Tag table tablelist with tagname
```

Alias

alias 语句用于设置别名。根据此别名，每当后面的脚本中出现相应字段时其都会重命名。

语法：

```
alias fieldname as aliasname {,fieldname as aliasname}
```

参数：

参数

参数	说明
fieldname	源数据中字段的名称
aliasname	可用于替换原名称的别名

示例和结果：

示例	结果
`Alias ID_N as NameID;`	
`Alias A as Name, B as Number, C as Date;`	通过此语句定义的名称更改可用于全部后续 **SELECT** 和 **LOAD** 语句。通过新的 **alias** 语句可以在脚本后面的任何位置定义字段名的新别名。

AutoNumber

此语句为脚本执行期间遇到的字段中每个不同的计算值创建唯一的整数值。

您还可在 **LOAD** 语句内使用 *autonumber (page 399)* 函数，但是这在您希望使用优化负载时会造成一些限制。您可创建优化负载，方法是先从 **QVD** 文件将数据载入，然后使用 **AutoNumber** 语句来转换值为符号键。

语法：

```
AutoNumber *fieldlist [Using namespace] ]
```

参数：

参数

参数	说明
*字段列表	逗号分隔的字段列表，其中值应当由唯一整数值替代。 您可在字段名称中使用通配字符 ? 和 * 来包括具有匹配名称的所有字段。您还可使用 * 来包括所有字段。您需要在使用了通配符时引用字段名称。
命名空间	**可选择使用**命名空间。如果您希望创建命名空间，可使用该选项，在其中不同字段中的相同值共用相同键。 如果不使用该选项，所有字段将具有单独的键索引。

限制：

如果您在脚本中有数个 **LOAD** 语句，则需要将 **AutoNumber** 语句置于最终 **LOAD** 语句之后。

示例 - 带有 AutoNumber 的脚本

脚本示例

在本例中，首次加载数据时不使用 **AutoNumber** 语句。然后添加 **AutoNumber** 语句以显示效果。

示例中所使用的数据：

将以下数据作为数据加载编辑中的内联加载载入，以创建以下脚本示例。将 **AutoNumber** 语句注释掉。

```
RegionSales: LOAD *,  Region &'|'& Year &'|'& Month as KeyToOtherTable INLINE [ Region, Year,
Month, Sales North,     2014,   May,    245 North,      2014,   May,    347 North,      2014,   June,   127 S
```

```
June,   645 South,      2013,   May,    367 South,      2013,   May,    221 ];
&'|'& Year &'|'& Month as KeyToOtherTable INLINE [Region, Year, Month, Budget North,  2014,
May,    200 North,  2014,   May,    350 North,  2014,   June,   150 South,  2014,   June,
500 South,  2013,   May,    300 South,  2013,   May,    200 ];  //AutoNumber KeyToOtherTable;
```

创建可视化

在 Qlik Sense 工作表中创建两个表格可视化。将 **KeyToOtherTable**、**Region**、**Year**、**Month** 和 **Sales** 作为维度添加到第一个表格。将 **KeyToOtherTable**、**Region**、**Year**、**Month** 和 **Budget** 作为维度添加到第二个表格。

结果

RegionSales 表格

KeyToOtherTable	Region	Year	Month	Sales
North|2014|June	North	2014	June	127
North|2014|May	North	2014	May	245
North|2014|May	North	2014	May	347
South|2013|May	South	2013	May	221
South|2013|May	South	2013	May	367
South|2014|June	South	2014	June	645

预算表格

KeyToOtherTable	Region	Year	Month	Budget
North|2014|June	North	2014	June	150
North|2014|May	North	2014	May	200
North|2014|May	North	2014	May	350
South|2013|May	South	2013	May	200
South|2013|May	South	2013	May	300
South|2014|June	South	2014	June	500

解释

该示例显示了链接两个表的复合字段 **KeyToOtherTable**。未使用 **AutoNumber**。请注意 **KeyToOtherTable** 值的长度。

添加 AutoNumber 语句

取消对加载脚本中的 **AutoNumber** 语句的注释。

```
AutoNumber KeyToOtherTable;
```

结果

RegionSales 表格

KeyToOtherTable	Region	Year	Month	Sales
1	North	2014	June	127
1	North	2014	May	245
2	North	2014	May	347
3	South	2013	May	221
4	South	2013	May	367
4	South	2014	June	645

预算表格

KeyToOtherTable	Region	Year	Month	Budget
1	North	2014	June	150
1	North	2014	May	200
2	North	2014	May	350
3	South	2013	May	200
4	South	2013	May	300
4	South	2014	June	500

解释

KeyToOtherTable 字段值已替换为唯一的整数值，因此，字段值的长度已减小，从而节省内存。两个表中的键字段都受 **AutoNumber** 的影响，并且表保持链接。尽管此例主要用于演示目的，但对于包含大量行的表格将非常有意义。

Binary

binary 语句用于加载另一个 Qlik Sense 应用程序或者 QlikView 文档的数据，包括区域权限数据。不包括应用程序的其他元素，例如，表格、故事、可视化、主条目或变量。

在脚本中仅允许一个 **binary** 语句。**binary** 语句必须是脚本的第一个语句，甚至在通常位于脚本开头的 SET 语句之前。

语法：

```
binary [path] filename
```

参数：

参数

参数	说明
path	文件路径，即对文件夹数据连接的引用。如果此文件不在 Qlik Sense 工作目录下，则需要使用此路径。 **示例：*'lib://Table Files/'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：*c:\data*** • 相对于包含此脚本行的应用程序。 **示例：*data***
filename	文件名称，包括文件扩展名 .qvw 或 .qvf。

限制：

您不能使用 **binary** 通过引用应用程序 ID 从相同 Qlik Sense Enterprise 部署上的应用程序加载数据。您只能从 *.qvf* 文件进行加载。

示例

字符串	说明
`Binary lib://DataFolder/customer.qvw;`	在此例中，文件必须位于与**文件夹**数据连接中。这可能是您的管理员在 Qlik Sense 服务器上创建的文件夹。在数据加载编辑器中单击**创建新连接**，然后在**文件位置**下选择**文件夹**。
`Binary customer.qvf;`	在此例中，文件必须位于 Qlik Sense 工作目录下。
`Binary c:\qv\customer.qvw;`	此例使用绝对文件路径，仅在传统脚本模式下起作用。

Comment field

提供一种从数据库和电子表格显示表格注释(元数据)的方式。可以忽略在应用程序中不显示的字段名。如果有字段名多次出现，将使用最后出现的值。

语法：

```
comment [fields] *fieldlist using mapname
comment [field] fieldname with comment
```

使用的映射表格应有两列，第一列包含字段名，第二列包含注释。

参数：

参数

参数	说明
**fieldlist*	要添加注释的字段的逗号分隔列表。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。
mapname	先前在映射 **LOAD** 或映射 **SELECT** 语句中读取的映射表的名称。
fieldname	要添加注释的字段名。
comment	要添加到字段的注释。

Example 1:

```
commentmap:
mapping LOAD * inline [
a,b
Alpha,This field contains text values
Num,This field contains numeric values
];
comment fields using commentmap;
```

Example 2:

```
comment field Alpha with AFieldContainingCharacters;
comment field Num with '*A field containing numbers';
comment Gamma with 'Mickey Mouse field';
```

Comment table

提供一种从数据库或电子表格显示表格注释(元数据)的方式。

可以忽略在应用程序中不显示的表格名。如果有表格名多次出现，将使用最后出现的值。关键字可用于从数据源中读取注释。

语法：

```
comment [tables] tablelist using mapname
comment [table] tablename with comment
```

参数：

参数

参数	说明
tablelist	(table{,table})

参数	说明
mapname	先前在映射 **LOAD** 或映射 **SELECT** 语句中读取的映射表的名称。
tablename	要添加注释的表格的名称。
comment	comment 是应添加到表格的注释。

Example 1:

```
Commentmap:
mapping LOAD * inline [
a,b
Main,This is the fact table
Currencies, Currency helper table
];
comment tables using Commentmap;
```

Example 2:

```
comment table Main with 'Main fact table';
```

Connect

CONNECT 语句用于定义 Qlik Sense 通过 OLE DB/ODBC 接口访问通用数据库。对于 ODBC，首先需要用 ODBC 管理员指定数据源。

该功能在 Qlik Sense SaaS 中不可用。

此语句仅在标准模式下支持文件夹数据连接。

语法：

```
ODBC CONNECT TO connect-string
OLEDB CONNECT TO connect-string
CUSTOM CONNECT TO connect-string
LIB CONNECT TO connection
```

参数：

参数

参数	说明
connect-string	connect-string ::= datasourcename { ; conn-spec-item } 连接字符串是数据源名称和一个包含一个或多个连接规范项的可选列表。如果数据源名称包含空白，或列出了任何连接规范项，则连接字符串必须用引号引起来。 **datasourcename** 必须为定义的 ODBC 数据源或用于定义 OLE DB 提供程序的字符串。 conn-spec-item ::=**DBQ=**database_specifier |**DriverID=**driver_specifier |**UID=**userid |**PWD=**password 不同数据库之间的连接规范项可能不同。对于某些数据库而言，还可能出现除上述项目之外的其他项。对于 OLE DB，一些特定连接规范项是强制而非可选的。
connection	存储在数据加载编辑器中的数据连接的名称。

如果将 **ODBC** 放置在 **CONNECT** 之前，那么将使用 ODBC 接口；否则将使用 OLE DB。

使用 **LIB CONNECT TO** 连接到使用存储的数据连接（在数据加载编辑器中创建）的数据库。

Example 1:

```
ODBC CONNECT TO 'Sales
DBQ=C:\Program Files\Access\Samples\Sales.mdb';
```

通过此语句定义的数据源由后面的 **Select (SQL)** 语句使用，直到出现新的 **CONNECT** 语句。

Example 2:

```
LIB CONNECT TO 'DataConnection';
```

Connect32

此语句的使用方式与 **CONNECT** 语句相同，但是可强制 64 位系统使用 32 位 ODBC/OLE DB 提供程序。不适用定制连接。

Connect64

此语句与 **CONNECT** 语句以相同的方式使用，但是可强制使用 64 位提供程序。不适用定制连接。

Declare

Declare 语句用于创建字段定义，您可以在其中定义字段或函数之间的关系。可以使用一组字段定义来自动生成可用作维度的导出字段。例如，您可以创建日历定义，并用它来从日期字段生成相关的维度，如年份、月份、周和日期。

您可以使用 **Declare** 设置新的字段定义，或者根据现有定义创建字段定义。

设置新的字段定义

语法：

```
definition_name:
Declare [Field[s]] Definition [Tagged tag_list ]
[Parameters parameter_list ]
Fields field_list
```

参数：

参数	说明
definition_name	字段定义的名称，以冒号为结尾。  *请勿将 autoCalendar 用作字段定义的名字，因为该名称保留用于自动生成的日历模板。* **示例：** `Calendar:`
tag_list	以逗号分隔的标记列表，要应用到根据字段定义导出的字段。应用标记是可选的，但如果不应用用于指定排序顺序的标记，如 $date、$numeric 或 $text，则导出字段将默认按加载顺序排序。 **示例：** `'$date'Thank you for bringing this to our attention, and apologies for the inconvenience.`
parameter_list	以逗号分隔的参数列表。定义的参数的格式为 `name=value` 且已分配初始值，在重新使用字段定义时可以将其覆盖。可选。 **示例：** `first_month_of_year = 1`
field_list	在使用字段定义时生成的以逗号分隔的字段列表。定义的字段的格式为 `<expression> As field_name tagged tag`。使用 `$1` 来引用应根据导出的字段生成的数据字段。 **示例：** `Year($1) As Year tagged ('$numeric')`

示例：

```
Calendar:
DECLARE FIELD DEFINITION TAGGED '$date'
   Parameters
```

```
      first_month_of_year = 1
   Fields
      Year($1) As Year Tagged ('$numeric'),
      Month($1) as Month Tagged ('$numeric'),
      Date($1) as Date Tagged ('$date'),
      Week($1) as Week Tagged ('$numeric'),
      Weekday($1) as Weekday Tagged ('$numeric'),
      DayNumberOfYear($1, first_month_of_year) as DayNumberOfYear Tagged ('$numeric')
;
```

现在，已经定义日历，您可以将其应用到已加载的日期字段，在此例中为使用 **Derive** 子句的 OrderDate 和 ShippingDate。

重复使用现有的字段定义

语法：

```
<definition name>:
Declare [Field][s] Definition
Using <existing_definition>
[With <parameter_assignment> ]
```

参数：

参数	说明
definition_name	字段定义的名称，以冒号为结尾。 **示例：** `MyCalendar:`
existing_definition	在创建新的字段定义时要重复使用字段定义。新的字段定义与它基于的定义具有相同的功能，不包括如果您使用 parameter_assignment 更改在字段表达式中使用的值。 **示例：** `Using Calendar`
parameter_assignment	以逗号分隔的参数赋值列表。定义的参数赋值的格式为 `name=value`，并且将覆盖在基本字段定义中所设置的参数值。可选。 **示例：** `first_month_of_year = 4`

示例：

在此例中，我们重复使用在前面的示例中所创建的日历定义。在这种情况下，我们想要使用从四月份开始的财政年。这可以通过将值 4 赋给 first_month_of_year 参数来实现，这会影响到所定义的 DayNumberOfYear 字段。

下例假设您使用样本数据和前面示例中的字段定义。

```
MyCalendar:
DECLARE FIELD DEFINITION USING Calendar WITH first_month_of_year=4;

DERIVE FIELDS FROM FIELDS OrderDate,ShippingDate USING MyCalendar;
```

当您重新加载数据脚本时，在表格编辑器中提供了所生成的字段，且名称为 OrderDate.MyCalendar.* 和 ShippingDate.MyCalendar.*。

Derive

Derive 语句用于根据通过使用 **Declare** 语句创建的字段定义生成导出字段。您可以指定要为其导出字段的数据字段，或者根据字段标签明确或默认导出它们。

语法：

```
Derive [Field[s]] From [Field[s]] field_list Using definition
Derive [Field[s]] From Explicit [Tag[s]] tag_list Using definition
Derive [Field[s]] From Implicit [Tag[s]] Using definition
```

参数：

参数

参数	说明
definition	在导出字段时使用的字段定义的名称。 **示例：Calendar**
field_list	根据字段定义应从导出的字段生成的以逗号分隔的数据字段列表。数据字段应是您已经在脚本中加载的字段。 **示例：OrderDate, ShippingDate**
tag_list	以逗号分隔的标记列表。可以通过使用任何已列出的标记为所有数据字段生成导出字段。标签列表应当包含在圆括号中。 **示例：('$date','$timestamp')**

示例：

- 导出特定数据字段的字段。
  在这种情况下，我们指定 OrderDate 和 ShippingDate 字段。
  `DERIVE FIELDS FROM FIELDS OrderDate,ShippingDate USING Calendar;`
- 使用特定标记导出所有字段的字段。
  在这种情况下，我们使用 `$date` 标记根据 Calendar 导出所有字段的字段。
  `DERIVE FIELDS FROM EXPLICIT TAGS ('$date') USING Calendar;`
- 使用字段定义标记导出所有字段的字段。
  在这种情况下，我们使用与 Calendar 字段定义相同的标记(在此例中为 `$date`)导出所有数据字段的字段。
  `DERIVE FIELDS FROM IMPLICIT TAG USING Calendar;`

Direct Query

DIRECT QUERY 语句可让您使用 Direct Discovery 函数通过 ODBC 或 OLE DB 连接访问表格。

语法：

```
DIRECT QUERY DIMENSION fieldlist [MEASURE fieldlist] [DETAIL fieldlist] FROM
tablelist
[WHERE where_clause]
```

DIMENSION、**MEASURE** 和 **DETAIL** 关键字可以按任何顺序使用。

DIMENSION 和 **FROM** 关键字子句在所有 **DIRECT QUERY** 语句中都是必需的。**FROM** 关键字必须在 **DIMENSION** 关键字后面。

DIMENSION 关键字后面直接指定的字段在内存中加载，并且可用于创建内存中数据与 Direct Discovery 数据之间的关联。

> ***DIRECT QUERY** 语句不能包含 **DISTINCT** 或 **GROUP BY** 子句。*

使用 **MEASURE** 关键字，您可以定义 Qlik Sense 在“元级别”识别的字段。在数据加载过程中度量字段的实际数据只驻留在数据库中，并且通过在可视化中使用的图表表达式推动进行临时检索。

通常，具有用作维度离散值的字段应使用 **DIMENSION** 关键字加载，而只用于聚合的数字应使用 **MEASURE** 关键字来选择。

DETAIL 字段提供用户可能希望在“钻取至详细信息”表窗格中显示的信息或详细信息，如注释字段。**DETAIL** 字段不能用于图表表达式中。

按照设计，**DIRECT QUERY** 语句是数据源的中立数据源，用于支持 SQL。由于此原因，可以将同一 **DIRECT QUERY** 语句用于不同的 SQL 数据库，不需要进行任何更改。Direct Discovery 根据需要生成相应的数据库查询。

如果用户知道要查询的数据库并希望利用 SQL 的数据库特定扩展名，可以使用本地数据源语法。支持本地数据源语法：

- 作为 **DIMENSION** 和 **MEASURE** 子句中的字段表达式
- 作为 **WHERE** 子句的内容

示例：

```
DIRECT QUERY
      DIMENSION Dim1, Dim2

      MEASURE
            NATIVE ('X % Y') AS X_MOD_Y
FROM TableName

DIRECT QUERY
```

```
DIMENSION Dim1, Dim2

MEASURE X, Y

FROM TableName

WHERE NATIVE ('EMAIL MATCHES "\*.EDU"')
```

下列术语可用作关键字，因此不能用作列或字段名称，且没有引号：and, as, detach, detail, dimension, distinct, from, in, is, like, measure, native, not, or, where

参数：

参数	说明
fieldlist	字段规范的逗号分隔列表，*fieldname {, fieldname}*. 字段规范可以是字段名称，在此情况下，相同名称用于数据库列名称和 Qlik Sense 字段名称。字段规范也可以是“字段别名”，在此情况下，数据库表达式或列名称被给予 Qlik Sense 字段名称。
tablelist	数据库中将从其中加载数据的表格或视图的名称列表。通常是包含在数据库中执行的联接的视图。

where_clause	此处没有定义数据库 **WHERE** 子句的完整语法，但允许使用大部分 SQL“关系表达式”，包括使用函数调用、字符串的 **LIKE** 运算符、**IS NULL** 和 **IS NOT NULL**，不包括 **IN. BETWEEN**。
	NOT 是一元运算符，而非某些关键字的修饰符。
	示例：
	`WHERE x > 100 AND "Region Code" IN ('south', 'west')`
	`WHERE Code IS NOT NULL and Code LIKE '%prospect'`
	`WHERE NOT X in (1,2,3)`
	不能将最后一个示例写成如下所示：
	`WHERE X NOT in (1,2,3)`

示例：

在本例中，使用了称为 TableName 的数据库表，包含字段 Dim1、Dim2、Num1、Num2 和 Num3。Dim1 和 Dim2 将被载入 Qlik Sense 数据集。

```
DIRECT QUERY DIMENSTION Dim1, Dim2 MEASURE Num1, Num2, Num3 FROM TableName ;
```

可将 Dim1 和 Dim2 用作维度。Num1、Num2 和 Num3 均可用于聚合。Dim1 和 Dim2 也可用于聚合。可以用于 Dim1 和 Dim2 的聚合类型取决于其数据类型。例如，在很多情况下，**DIMENSION** 字段包含字符串数据，如名称或帐号。这些字段无法求和，但是可以对其进行计数：`count(Dim1)`。

DIRECT QUERY 语句直接在脚本编辑器中编写。要简化 DIRECT QUERY 语句的结构，您可以从数据连接生成 SELECT 语句，然后编辑生成的脚本以将其更改成 DIRECT QUERY 语句。
例如 SELECT 语句：

```
SQL SELECT
 SalesOrderID,
 RevisionNumber,
 OrderDate,
 SubTotal,
 TaxAmt
FROM MyDB.Sales.SalesOrderHeader;
```

可更改为以下 DIRECT QUERY 语句：

```
DIRECT QUERY
 DIMENSION
 SalesOrderID,
 RevisionNumber

MEASURE
 SubTotal,
 TaxAmt

DETAIL
OrderDate

FROM MyDB.Sales.SalesOrderHeader;
```

Direct Discovery 字段列表

字段列表是以逗号分隔的字段规范列表，如 *fieldname {, fieldname}*。字段规范可以是字段名称，在此情况下，相同名称用于数据库列名称和字段名称。字段规范也可以是“字段别名”，在此情况下，数据库表达式或列名称被给予 Qlik Sense 字段名称。

字段名称既可以是简单名称也可以是引用名称。简单名称以字母 Unicode 字符为开头，后跟字母或数字字符或下划线的任意组合。引用名称以双引号为开头，并包含任意字符序列。如果引用名称包含双引号，则这些引号使用两个相邻的双引号表示。

Qlik Sense 字段名称区分大小写。数据库字段名称可能会或可能不会区分大小写，具体取决于数据库。Direct Discovery 查询保留所有字段标识符和别名的情况。在下例中，内部使用别名 "MyState" 存储数据库列 "STATEID" 的数据。

```
DIRECT QUERY Dimension STATEID as MyState Measure AMOUNT from SALES_TABLE;
```

这不同于使用别名的 **SQL Select** 语句的结果。如果没有明确引用别名，则结果包含由目标数据库返回的列的默认情况。在下例中，**SQL Select** 语句用于在 Oracle 数据库中创建 "MYSTATE,"，且所有字母均为大写，这就像内部 Qlik Sense 别名一样，即使指定别名为混合大小写也是如此。**SQL Select** 语句使用数据库返回的列名称，在这种情况下 Oracle 为全部大写。

```
SQL Select STATEID as MyState, STATENAME from STATE_TABLE;
```

要避免这种行为，可使用 LOAD 语句指定别名。

```
Load STATEID as MyState, STATENAME;
SQL Select STATEID, STATEMENT from STATE_TABLE;
```

在本例中，Qlik Sense 在内部将 "STATEID" 列存储作为 "MyState"。

可以将大部分数据库的标量表达式作为字段规范。在字段规范中也可以使用函数调用。表达式包含的常量可以是布尔值、数字或用单引号括起来的字符串(嵌入式单引号用相邻的单引号表示)。

示例：

```
DIRECT QUERY

   DIMENSION

      SalesOrderID, RevisionNumber

   MEASURE

      SubTotal AS "Sub Total"

FROM AdventureWorks.Sales.SalesOrderHeader;


DIRECT QUERY

   DIMENSION

      "SalesOrderID" AS "Sales Order ID"

   MEASURE

      SubTotal,TaxAmt,(SubTotal-TaxAmt) AS "Net Total"

FROM AdventureWorks.Sales.SalesOrderHeader;


DIRECT QUERY

   DIMENSION

      (2*Radius*3.14159) AS Circumference,

      Molecules/6.02e23 AS Moles

   MEASURE
```

```
            Num1 AS numA

      FROM TableName;


DIRECT QUERY
   DIMENSION
      concat(region, 'code') AS region_code
   MEASURE
      Num1 AS NumA
FROM TableName;
```

Direct Discovery 不支持在 **LOAD** 语句中使用聚合。如果使用聚合，则结果将不可预测。不得使用类似如下的 **LOAD** 语句：

```
DIRECT QUERY DIMENSION stateid, SUM(amount*7) AS MultiFirst MEASURE amount FROM sales_table;
```

在 **LOAD** 语句中不得使用 **SUM**。

Direct Discovery 也不支持在 **Direct Query** 语句中使用 Qlik Sense 函数。例如，当将 "Mth" 字段用作可视化的维度时，**DIMENSION** 字段的以下规范将无法使用：

```
month(ModifiedDate) as Mth
```

Directory

Directory 语句用于定义在后续 **LOAD** 语句中查找数据文件的目录，直到出现新的 **Directory** 语句。

语法：

```
Directory[path]
```

如果 **Directory** 语句在没有 **path** 或将其忽略的情况下发布，Qlik Sense 将会查找 Qlik Sense 工作目录。

参数：

参数

参数	说明
path	可解释为 data 文件的路径的文本。 该路径是文件的路径，即： • 绝对 **示例：*c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：*data*** • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：*http://www.qlik.com***

示例：

```
DIRECTORY C:\userfiles\data; // OR -> DIRECTORY data\

LOAD * FROM
[data1.csv] // ONLY THE FILE NAME CAN BE SPECIFIED HERE (WITHOUT THE FULL PATH)
(ansi, txt, delimiter is ',', embedded labels);

LOAD * FROM
[data2.txt] // ONLY THE FILE NAME CAN BE SPECIFIED HERE UNTIL A NEW DIRECTORY STATEMENT IS
MADE
(ansi, txt, delimiter is '\t', embedded labels);
```

Disconnect

Disconnect 语句用于终止当前 ODBC/OLE DB/自定义连接。此语句为可选。

语法：

```
Disconnect
```

当执行新 **connect** 语句或完成脚本执行时该连接将自动终止。

示例：

```
Disconnect;
```

Drop

Drop脚本关键字可用于从数据库删除表格或字段。

Drop field

在执行脚本期间，可以使用 **drop field** 语句随时从数据模型和内存中删除一个或多个 Qlik Sense 字段。

> ***drop field*** *和* ***drop fields*** *都是允许的格式，效果完全一样。如果未指定任何表格，字段将从全部表格中删除。*

语法：

```
Drop field fieldname { , fieldname2 ...} [from tablename1 { , tablename2 ...}]
Drop fields fieldname { , fieldname2 ...} [from tablename1 { , tablename2 ...}]
```

示例：

```
Drop field A;
Drop fields A,B;
Drop field A from X;
Drop fields A,B from X,Y;
```

Drop table

在执行脚本期间，可以使用 **drop table** 语句随时从数据模型和内存中删除一个或多个 Qlik Sense 内部表格。

语法：

```
drop table  tablename {, tablename2 ...}
drop tables tablename {, tablename2 ...}
```

> *可以同时接受* ***drop table*** *和* ***drop tables*** *格式。*

使用此语句将导致以下项目丢失：

- 真实表格。
- 不属于剩余表格部分的全部字段。
- 剩余字段中的字段值，独立于已删除表格。

示例和结果：

示例	结果
`drop table Orders, Salesmen, T456a;`	这一行将产生从内存中删除的三个表格。

示例	结果
Tab1: Load * Inline [Customer, Items, UnitPrice Bob, 5, 1.50]; Tab2: LOAD Customer, Sum(Items * UnitPrice) as Sales resident Tab1 group by Customer; drop table Tab1;	创建表格 *Tab2* 后，已删除表格 *Tab1*。

Drop table

在执行脚本期间，可以使用 **drop table** 语句随时从数据模型和内存中删除一个或多个 Qlik Sense 内部表格。

语法：

```
drop table  tablename {, tablename2 ...}
drop tables tablename {, tablename2 ...}
```

*可以同时接受 **drop table** 和 **drop tables** 格式。*

使用此语句将导致以下项目丢失：

- 真实表格。
- 不属于剩余表格部分的全部字段。
- 剩余字段中的字段值，独立于已删除表格。

示例和结果：

示例	结果
drop table Orders, Salesmen, T456a;	这一行将产生从内存中删除的三个表格。
Tab1: Load * Inline [Customer, Items, UnitPrice Bob, 5, 1.50]; Tab2: LOAD Customer, Sum(Items * UnitPrice) as Sales resident Tab1 group by Customer; drop table Tab1;	创建表格 *Tab2* 后，已删除表格 *Tab1*。

Execute

Execute 语句用于在 Qlik Sense 加载数据的同时运行其他程序。例如，需要执行转换。

该功能在 Qlik Sense SaaS 中不可用。

在标准模式下不支持此语句。

语法：

```
execute commandline
```

参数：

参数

参数	说明
commandline	可以通过操作系统解释为命令行的文本。您可以引用文件的绝对文件路径或者 lib:// 文件夹路径。

如果您想要使用 **Execute**，则需要满足以下条件：

- 您必须在旧模式下运行（适用于 Qlik Sense 和 Qlik Sense Desktop）。
- 您需要在 *Settings.ini* 中将OverrideScriptSecurity 设置为 1（适用于 Qlik Sense）。*Settings.ini* 位于 *C:\ProgramData\Qlik\Sense\Engine*，一般都是空文件。

*如果您设置 OverrideScriptSecurity 为启用 **Execute**，则任何用户都可以在服务器上执行文件。例如，用户可将可执行文件附加到应用程序，然后再数据加载脚本中执行该文件。*

执行以下操作：

1. 复制 *Settings.ini* 并在文本编辑器中打开。
2. 检查文件的第一行是否包含 *[Settings 7]*。
3. 插入新行并键入 *OverrideScriptSecurity=1*。
4. 在文件的末尾插入新行。
5. 保存文件。
6. 使用编辑后的文件替换 *Settings.ini*。
7. 重新启动 Qlik Sense Engine Service (QES)。

如果将 Qlik Sense 作为服务运行，则有些命令可能无法正常运行。

示例：

```
Execute C:\Program Files\Office12\Excel.exe;

Execute lib://win\notepad.exe // win is a folder connection referring to c:\windows
```

Field/Fields

Field 和 **Fields** 脚本关键字用于 **Declare**、**Derive**、**Drop**、**Comment**、**Rename** 和 **Tag/Untag** 语句。

FlushLog

FlushLog 语句用于强制 Qlik Sense 将脚本缓冲区的内容写入脚本日志文件。

语法：

```
FlushLog
```

缓冲区的内容写入日志文件。该命令对于调试非常有用，因为您可能接收已经在错误的脚本执行中丢失的数据。

示例：

```
FlushLog;
```

Force

force 语句用于强制 Qlik Sense 将后面的 **LOAD** 和 **SELECT** 语句的字段名称和字段值写入方式解释为仅限大写字母，仅限小写字母，总是首字母大写或它们的原初显示形式 (大小写混合)。此语句可以根据不同的惯例关联表格的字段值。

语法：

```
Force ( capitalization | case upper | case lower | case mixed )
```

如果未指定任何一个，将假设为强制大小写混合。force 语句会一直有效，直到新的 force 语句出现。

force 语句对存取部分无任何影响：全部加载的字段值都不区分大小写。

示例和结果

示例	结果

本示例说明了如何强制转换为首字母大写。
 	
`FORCE Capitalization;`
`Capitalization:`
`LOAD * Inline [`
`ab`
`Cd`
`eF`
`GH`
`];`
	Capitalization 表格包含以下值：
	
	`Ab`
	`Cd`
	`Ef`
	`Gh`
	所有值均为首字母大写。

本示例说明了如何强制转换为大写。
 	
`FORCE Case Upper;`
`CaseUpper:`
`LOAD * Inline [`
`ab`
`Cd`
`eF`
`GH`
`];`
	CaseUpper 表格包含以下值：
	
	`AB`
	`CD`
	`EF`
	`GH`
	所有值都采用大写。

本示例说明了如何强制转换为小写。
 	
`FORCE Case Lower;`
`CaseLower:`
`LOAD * Inline [`
`ab`
`Cd`
`eF`
`GH`
`];`
	CaseLower 表格包含以下值：
	
	`ab`
	`cd`
	`ef`
	`gh`
	所有值都采用小写。

本示例说明了如何强制转换为大小写混合。
 	
`FORCE Case Mixed;`
`CaseMixed:`
`LOAD * Inline [`
`ab`
`Cd`
`eF`
`GH`
`];`
	CaseMixed 表格包含以下值：
	
	`ab`
	`Cd`
	`eF`
	`GH`
	所有的显示与脚本中相同。

另请参见：

From

From 脚本关键字在 **Load** 语句中用于引用文件，在 **Select** 语句中用于引用数据库表格或视图。

Load

LOAD 语句可以加载以下来源的字段：文件、脚本中定义的数据、预先载入的输入表格、网页、后续 **SELECT** 语句产生的结果或自动生成的数据。还可从分析连接加载数据。

语法：

```
LOAD [ distinct ] fieldlist
[( from file [ format-spec ] |
from_field fieldassource [format-spec]|
inline data [ format-spec ] |
resident table-label |
autogenerate size ) |extension pluginname.functionname([script]
tabledescription)]
[ where criterion | while criterion ]
[ group by groupbyfieldlist ]
[order by orderbyfieldlist ]
```

参数：

参数

参数	说明
distinct	如果您只希望加载唯一的记录，您可将 **distinct** 用作谓语。如果存在重复的记录，则将加载第一个实例。 如果您使用前置加载，需要将 **distinct** 置于第一加载语句，因为 **distinct** 仅影响目标表格。

参数	说明
fieldlist	*fieldlist ::= (* | field {, * | field })* 要加载的字段列表。使用 * 作为字段列表，表示表格中全部字段。 *field ::= (fieldref | expression) [***as** *aliasname]* 字段定义必须总是包含精确的对现存字段或表达式的引用。 *fieldref ::= (fieldname |@fieldnumber |@startpos:endpos [**I** | **U** | **R** | **B** | **T**])* *fieldname* 是指与表格中的字段名完全相同的文本。注意，如果包含空格，则字段名必须使用双引号或方括号括起来。有时字段名不会显示可用。这时使用另外的符号： *@fieldnumber* 表示字段数字在带分隔符的表格文件中。它必须是前面带“@”的正整数。编号通常从 1 开始至字段数字。 *@startpos*:*endpos* 代表在拥有固定长度记录的文件中字段的开始和结束位置。这两个位置必须都是正整数。这两个数字前都必须加上“@”并用冒号隔开。编号通常从 1 开始至位置数字。在最后一个字段中，将 **n** 用作结束位置。 • 如果 *@startpos*:*endpos* 后紧随字符 **I** 或 **U**，字节读取将分别解释为二进制带符号 (**I**) 或不带符号 (**U**) 整数（Intel 字节序）。数字读取位置必须是 1、2 或 4。 • 如果 *@startpos*:*endpos* 后紧随字符 **R**，字节读取将解释成二进制实数（IEEE 32-bit 或 64 位浮点）。数字读取位置必须是 4 或 8。 • 如果 *@startpos*:*endpos* 后紧随字符 **B**，字节读取将根据 COMP-3 标准解释成 BCD (Binary Coded Decimal) 数字。任何字节数可以是指定的。 *expression* 可以是数学函数或基于同一表格中一个或多个其他字段的字符串函数。有关详细信息，请参阅表达式的语法。 **as** 用于为字段设定新名称。

参数	说明
from	如果使用文件夹或 Web 文件数据连接从文件加载数据，可使用 **from** *file ::= [path] filename* **示例：** ***'lib://Table Files/'*** 如果路径被省略，Qlik Sense 则在由 **Directory** 语句指定的目录中搜索文件。如果没有 **Directory** 语句，那么 Qlik Sense 将在工作目录 *C:\Users\{user}\Documents\Qlik\Sense\Apps* 中搜索。 *在 Qlik Sense 服务器安装中，工作目录是在 Qlik Sense 存储库服务 中指定，默认情况下是 C:\ProgramData\Qlik\Sense\Apps。* *filename* 可能包含标准 DOS 通配符字符（* 和 ?）。这将导致指定目录中的所有匹配文件被加载。 *format-spec ::= (fspec-item { , fspec-item })* 格式规格包含数个格式规格项目的列表（在括号中）。 **传统脚本模式** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** ***c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** ***data*** • URL 地址（HTTP 或 FTP），指向一个互联网或内联网的位置。 **示例：** ***http://www.qlik.com***
from_field	如果需要从之前加载的字段加载数据，则使用 **from_field** 语句。 *fieldassource::=(tablename, fieldname)* 该字段是之前加载过的 *tablename* 和 *fieldname* 的名称。 *format-spec ::= (fspec-item {, fspec-item })* 格式规格包含数个格式规格项目的列表（在括号中）。

参数	说明
inline	**inline** 当数据需要输入脚本而不是从文件中加载时使用该符号。 *data ::= [text]* 通过 **inline** 子句的输入的数据由双引号或方括号括起来。括号之间的文本以同一方式被解释为文件的内容。因此，当您需要在文本文件中插入新的一行时，您应该在 **inline** 子句文本中重复该操作，即，键入脚本时按压输入键。列数通过第一行来定义。 *format-spec ::= (fspec-item {, fspec-item })* 格式规格包含数个格式规格项目的列表(在括号中)。
resident	如果需要从之前加载的表格加载数据，则使用 **resident** 语句。 *table label* 是加在创建于原始表格的 **LOAD** 或 **SELECT** 语句之前的标签。该标签需要在末尾加上冒号。
autogenerate	**autogenerate** 是数据需要 Qlik Sense 自动生成时使用。 *size ::= number* *Number* 是用来指示生成记录数字的整数。 字段列表不得包含需要外部数据源或之前加载表格中数据的表达式，除非您引用之前使用 **Peek** 函数加载的表格中的单个字段值。

参数	说明
extension	您可从分析连接加载数据。您需要使用 **extension** 子句来调用在服务器端扩展 (SSE) 插件中定义的函数或计算脚本。 您可将单个表格发送至 SSE 插件，并返回单个数据表。如果插件没有指定返回的字段的名称，字段将被命名为 Field1, Field2，并且以此类推。 `Extension pluginname.functionname( tabledescription );` • 使用 SSE 插件中的函数加载数据 *tabledescription ::= (table { ,tablefield})* 如果您没有声明表格字段，将按加载顺序使用这些字段。 • 通过在 SSE 插件中运算脚本来加载数据 *tabledescription ::= (script, table { ,tablefield})* **表格字段定义中的数据类型处理** 将在分析连接中自动检测数据类型。如果数据没有数值以及至少一个非 NULL 文本字符串，则字段被视为文本。在其他任何情况下中，其将被视为数字。 您可通过用 **String()**或 **Mixed()** 围住字段名称来强制规定数据类型。 • **String()** 将强制规定字段为文本。如果字段是数字，则会提取双重值的文本部分，不会执行转换。 • **Mixed()** 强制规定字段为双重值。 **String()** 或 **Mixed()** 不能在 **extension** 表格字段定义之外使用，并且您无法在表格字段定义中使用其他 Qlik Sense 函数。 **有关分析连接的更多信息** 您需要先配置分析连接方可使用它们。
where	**where** 是一个子句，用于陈述一个记录是否应该包括在选择项内。如果 *criterion* 为 True，则将其包括在选择项内。 *criterion* 是一个逻辑表达式。
while	**while** 是用于显示记录是否应该反复读取的子句。只要 *criterion* 为 True，则同一记录被读取。为了能发挥作用，**while** 子句通常应包含 **IterNo( )** 函数。 *criterion* 是一个逻辑表达式。
group by	**group by** 是用于定义应聚合(组合)的字段的子句。聚合字段应该以某种方式包含在加载表达式中。除了聚合字段没有其他字段可被用于加载表达式中的聚合函数之外。 *groupbyfieldlist ::= (fieldname { ,fieldname })*

参数	说明
order by	**order by** 是用于被加于 **load** 语句之前的驻留表记录排序的子句。驻留表可以被一个或多个字段以升序或降序的顺序排序。排序主要由数值决定，其次由国家校对顺序决定。此子句仅在数据源为驻留表时可用。 排序字段用于指定驻留表按哪些字段排序。驻留表中的字段可以由其名称或其数字指定(第一个字段为 1)。 orderbyfieldlist ::= fieldname [sortorder] { , fieldname [sortorder] } *sortorder* 要么是 *asc*(对于升序)，要么是 *desc*(对于降序)。如果未指定任何 *sortorder*，将假设 *asc*。 *fieldname*、*path*、*filename* 和 *aliasname* 都是表示他们各自名称的字符串。源表格中的任何字段都可用作 *fieldname*。但是，通过子句 (*aliasname*) 创建的字段不在此范围之内，且不能用于相同的 **load** 语句内。

如果 **from**、**inline**、**resident**、**from_field, extension** 或 **autogenerate** 子句都无法给出数据源，则数据将从 **SELECT** 或 **LOAD** 语句随后得出的结果中加载。接下来的语句不应包含前缀。

示例：

加载不同文件格式
使用默认选项加载分隔符分隔的数据文件：

```
LOAD * from data1.csv;
```

通过库连接 (DataFiles) 加载分隔符分隔的数据文件：

```
LOAD * from 'lib://DataFiles/data1.csv';
```

通过库连接 (DataFiles) 加载所有分隔符分隔的数据文件：

```
LOAD * from 'lib://DataFiles/*.csv';
```

加载以逗号为分隔符且包含嵌入标签的分隔文件：

```
LOAD * from 'c:\userfiles\data1.csv' (ansi, txt, delimiter is ',', embedded labels);
```

加载以制表符为分隔符且包含嵌入标签的分隔文件：

```
LOAD * from 'c:\userfiles\data2.txt' (ansi, txt, delimiter is '\t', embedded labels);
```

加载包含嵌入标题的 dif 文件：

```
LOAD * from file2.dif (ansi, dif, embedded labels);
```

从没有标题的固定记录文件加载三个字段：

```
LOAD @1:2 as ID, @3:25 as Name, @57:80 as City from data4.fix (ansi, fix, no labels, header is
0, record is 80);
```

加载指定绝对文件路径的 QVX 文件：

```
LOAD * from C:\qdssamples\xyz.qvx (qvx);
```

正在加载 Web 文件
从 Web 文件数据连接中设置的默认 URL 加载：

```
LOAD * from [lib://MyWebFile];
```

从特定 URL 加载，并覆盖 Web 文件数据连接中设置的 URL：

```
LOAD * from [lib://MyWebFile] (URL is 'http://localhost:8000/foo.bar');
```

使用货币符号扩展从变量中设置的特定 URL 加载：

```
SET dynamicURL = 'http://localhost/foo.bar';
LOAD * from [lib://MyWebFile] (URL is '$(dynamicURL)');
```

选择特定字段，重命名和已计算字段
仅从分隔符分隔的文件加载三个特定字段：

```
LOAD FirstName, LastName, Number from data1.csv;
```

加载没有标签的文件时，将第一个字段重命名为 A，将第二个字段重命名为 B：

```
LOAD @1 as A, @2 as B from data3.txt (ansi, txt, delimiter is '\t', no labels);
```

以 FirstName、空格字符和 LastName 的串联形式加载 Name：

```
LOAD FirstName&' '&LastName as Name from data1.csv;
```

加载 Quantity、Price 和 Value(有 Quantity 和 Price 的产品) ：

```
LOAD Quantity, Price, Quantity*Price as Value from data1.csv;
```

选择特定记录
仅加载唯一记录，重复记录会被丢弃：

```
LOAD distinct FirstName, LastName, Number from data1.csv;
```

仅加载字段 Litres 值大于零的记录：

```
LOAD * from Consumption.csv where Litres>0;
```

加载不在文件中的数据和自动生成的数据
加载一个包含内联数据的表格、两个名为 CatID 和 Category 的字段：

```
LOAD * Inline
[CatID, Category
0,Regular
1,Occasional
2,Permanent];
```

加载一个包含内联数据的表格、三个名为 UserID、Password 和 Access 的字段：

```
LOAD * Inline [UserID, Password, Access
A, ABC456, User
B, VIP789, Admin];
```

加载一个有 10000 行的表格。字段 A 将包含读取记录 (1,2,3,4,5...) 的数量，字段 B 将包含一个介于 0 和 1 之间的随机数字：

```
LOAD RecNo( ) as A, rand( ) as B autogenerate(10000);
```

autogenerate 后的括号虽允许使用，但不是必须的。

从之前加载的表格中加载数据
首先，加载一个分隔符分隔的表格文件，并将其命名为 tab1：

```
tab1:
SELECT A,B,C,D from 'lib://DataFiles/data1.csv';
```

从已加载的 tab1 表格中加载字段作为 tab2：

```
tab2:
LOAD A,B,month(C),A*B+D as E resident tab1;
```

从已加载的 tab1 表格中加载字段，但仅加载 A 大于 B 的记录：

```
tab3:
LOAD A,A+B+C resident tab1 where A>B;
```

从已加载的 tab1 表格中加载按 A 排序的字段：

```
LOAD A,B*C as E resident tab1 order by A;
```

从已加载的 tab1 表格中加载先按第一个字段，然后按第二个字段排序的字段：

```
LOAD A,B*C as E resident tab1 order by 1,2;
```

从已加载的 tab1 表格中加载依次按 C 降序，B 升序，第一个字段降序排序的字段：

```
LOAD A,B*C as E resident tab1 order by C desc, B asc, 1 desc;
```

从之前加载的字段中加载数据
从之前加载的表格 Characters 中加载字段 Types 作为 A：

```
LOAD A from_field (Characters, Types);
```

从随后的表格中加载数据(前置加载)
使用随后的 **SELECT** 语句从已加载的 Table1 中加载 A、B 以及已计算字段 X 和 Y：

```
LOAD A, B, if(C>0,'positive','negative') as X, weekday(D) as Y;
SELECT A,B,C,D from Table1;
```

组合数据
加载按 ArtNo 分组(聚合)的字段：

```
LOAD ArtNo, round(Sum(TransAmount),0.05) as ArtNoTotal from table.csv group by ArtNo;
```

加载按 Week 和 ArtNo 分组(聚合)的字段：

```
LOAD Week, ArtNo, round(Avg(TransAmount),0.05) as WeekArtNoAverages from table.csv group by
Week, ArtNo;
```

重复读取一个记录

在本例中，输入文件 Grades.csv 包含合并在一个字段中的每个学生的分数：

```
Student,Grades
Mike,5234
John,3345
Pete,1234
Paul,3352
```

分数（1-5 分）分别代表科目 Math、English、Science 和 History。通过使用 **while** 子句（使用 **IterNo( )** 函数作为一个计数器）多次读取每一条记录，我们可以将分数划分为单独的值。在每一次读取中，分数使用 **Mid** 函数进行提取，使用 Grade 进行存储，而科目则使用 **pick** 函数进行选择，使用 Subject 进行存储。最后一个 **while** 子句包含检查是否所有分数均已读取的测试（本例中每个学生四个分数），这表示应该读取下一个学生记录。

```
MyTab:
LOAD Student,
mid(Grades,IterNo( ),1) as Grade,
pick(IterNo( ), 'Math', 'English', 'Science', 'History') as Subject from Grades.csv
while IsNum(mid(Grades,IterNo(),1));
```

结果是包含以下数据的表格：

Student	Subject	Grade
John	English	3
John	History	5
John	Math	3
John	Science	4
Mike	English	2
Mike	History	4
Mike	Math	5
Mike	Science	3
Paul	English	3
Paul	History	2
Paul	Math	3
Paul	Science	5
Pete	English	2
Pete	History	4
Pete	Math	1
Pete	Science	3

从分析连接加载

使用了以下示例数据。

```
Values:
Load
 Rand() as A,
 Rand() as B,
 Rand() as C
AutoGenerate(50);
```

使用函数加载数据

在这些实例中，我们假设具有名为 *P* 的分析连接插件，该插件包含自定义函数 *Calculate (Parameter1, Parameter2)*。函数返回表格 *Results*，该表格包含字段 *Field1* 和 *Field2*。

```
Load * Extension P.Calculate( Values{A, C} );
```
加载将字段 A 和 C 发送至函数时返回的所有字段。

```
Load Field1 Extension P.Calculate( Values{A, C} );
```
当把字段 A 和 C 发送至函数时仅加载 Field1 字段。

```
Load * Extension P.Calculate( Values );
```
加载将字段 A 和 B 发送至函数时返回的所有字段。由于未指定字段，在表格中以第一顺序使用 A 和 B。

```
Load * Extension P.Calculate( Values {C, C});
```
加载将字段 C 发送至函数的两个参数时返回的所有字段。

```
Load * Extension P.Calculate( Values {String(A), Mixed(B)});
```
加载将强制规定为字符串的字段 A 和强制规定为数字的 B 发送至函数时返回的所有字段。

通过运算脚本来加载数据

```
Load A as A_echo, B as B_echo Extension R.ScriptEval( 'q;', Values{A, B} );
```
加载发送 A 和 B 的值时由脚本 q 返回的表格。

```
Load * Extension R.ScriptEval( '$(My_R_Script)', Values{A, B} );
```
加载发送 A 和 B 的值时由 My_R_Script 变量中存储的脚本返回的表格。

```
Load * Extension R.ScriptEval( '$(My_R_Script)', Values{B as D, *} );
```
加载发送重命名为 D、A、C 的 B 的值时由 My_R_Script 变量中存储的脚本返回的表格。使用 * 发送剩余未参考的字段。

DataFiles 连接的文件扩展要区分大小写。例如：.qvd。

格式规格项目

每种格式规格项目定义表格文件的特定属性：

fspec-item ::= [**ansi | oem | mac | UTF-8 | Unicode | txt | fix | dif | biff | ooxml | html | xml | kml | qvd | qvx | delimiter is** `char` | **no eof | embedded labels | explicit labels | no labels | table is [**`tablename`**] | header is** `n` | **header is** `line` | **header is** `n` **lines** | **comment is** `string` | **record is** `n` | **record is** `line` | **record is** `n` **lines** | **no quotes |msq** | **URL is** `string` | **userAgent is** `string`]

字符集

字符集是定义文件所用字符集的 **LOAD** 语句中的一个文件说明符。

ansi、**oem** 和 **mac** 说明符用于 QlikView 中，目前仍可使用。但是，当使用最新版 Qlik Sense 创建 **LOAD** 语句时，不会生成这几个说明符。

语法：

```
utf8 | unicode | ansi | oem | mac | codepage is
```

参数：

参数

参数	说明
utf8	UTF-8 字符集
unicode	Unicode 字符集
ansi	Windows、代码页 1252
oem	DOS、OS/2、AS400 等
mac	代码页 10000
codepage is	通过 **codepage** 说明符，可以将任何 Windows 代码页用作 *N*。

限制：

从 **oem** 字符集进行的转换在 MacOS 中未实现。如果未指定任何一项，将假设在 Windows 下使用代码页 1252。

示例：

```
LOAD * from a.txt (utf8, txt, delimiter is ',' , embedded labels)
LOAD * from a.txt (unicode, txt, delimiter is ',' , embedded labels)
LOAD * from a.txt (codepage is 10000, txt, delimiter is ',' , no labels)
```

另请参见：

Load (page 91)

表格格式

表格格式是用于定义文件类型的 **LOAD** 语句的文件说明符。如果未指定任何一个，将假设为 *.txt* 文件。

表格式类型

类型	说明
txt	在分隔符分隔的文本文件中，表格各列都用分隔符分隔。

类型	说明
fix	在固定记录文件中，每个字段实际上是一定的字符数。 通常，许多固定记录长度文件都包含以换行符分隔的记录，但有更高级的选项可指定记录的字节大小，或使用 **Record is** 跨越多行。 *如果数据包含多字节字符，则字段断开不整齐，因为格式基于固定长度(以字节为单位)。*
dif	在 *.dif* 文件中，将使用一种特殊的格式(Data Interchange Format) 定义表格。
biff	Qlik Sense 还可以通过使用 Excel 格式(*biff*) 解释标准 Binary Interchange File Format 文件中的数据。
ooxml	Excel 2007 和更高版本使用 ooxml *.xslx* 格式。
html	如果表格是 html 页面或文件的一部分，则应使用 html。
xml	xml(可扩展标记语言) 是一种通用标记语言，用于以文本格式表示数据结构。
qvd	*qvd* 格式是 QVD 文件的专用格式，可从 Qlik Sense 应用程序导出。
qvx	*qvx* 是一种用于 Qlik Sense 高性能输出的文件/数据流格式。

Delimiter is

对于分隔的表格文件，可以通过 **delimiter is** 说明符指定任意分隔符。该说明符仅适用于分隔的 .txt 文件。

语法：

```
delimiter is char
```

参数：

参数

参数	说明
char	从 127 ASCII 字符指定单个字符。

此外，可以使用以下值：

可选值

值	说明
'\t'	代表标签符号，可以有或无引号。
'\\'	代表反斜线 (\) 字符。
'spaces'	代表有一个或多个空格的全部组合。ASCII 值小于 32 的非打印字符(CR 和 LF 除外) 将被解释为空格。

如果未指定任何一个，将假设为 **delimiter is ','**。

示例：

```
LOAD * from a.txt (utf8, txt, delimiter is ',' , embedded labels);
```

另请参见：

- *Load (page 91)*

No eof

no eof 说明符用于在加载分隔的 **.txt** 文件时忽略文件结束字符。

语法：

```
no eof
```

如果使用 **no eof** 说明符，可以忽略代码点为 26 的字符，并将其作为字段值的一部分，否则表示文件结束。

该说明符仅与分隔符分隔的文本文件相关。

示例：

```
LOAD * from a.txt (txt, utf8, embedded labels, delimiter is ' ', no eof);
```

另请参见：

- *Load (page 91)*

Labels

Labels 是 **LOAD** 语句的一个文件说明符，该语句定义可在文件中找到字段名的位置。

语法：

```
embedded labels|explicit labels|no labels
```

字段名可以显示在文件的不同位置。如果第一条记录包含字段名，应使用 **embedded labels**。如果没有字段名显示，应使用 **no labels**。在 *dif* 文件中，有时可以使用显式字段名的单独页眉部分。在这种情况，应使用 **explicit labels**。如果未指定任何一项，将假设使用 **embedded labels**，同样用于 *dif* 文件。

Example 1:

```
LOAD * from a.txt (unicode, txt, delimiter is ',' , embedded labels
```

Example 2:

```
LOAD * from a.txt (codePage is 1252, txt, delimiter is ',' , no labels)
```

另请参见：

- *Load (page 91)*

Header is

在表格文件中指定标题大小。可以通过 **header is** 说明符指定任意标题长度。标题是 Qlik Sense 不会用到的文本部分。

语法：

```
header is n
header is line
header is n lines
```

可以字节为单位提供标题长度（**header is n**）或按行（**header is line**或**header is n lines**）提供。**n** 必须是正整数，表示标题长度。如果未指定，将假设 **header is 0**。**header is** 说明符仅用于表格文件。

示例：

这是包含标题文本行的数据源表格的示例，Qlik Sense 不能将该标题文本行解释为数据。

```
*Header line
Col1,Col2
a,B
c,D
```

使用 **header is 1 lines** 说明符，第一行不会作为数据加载。在本示例中，**embedded labels** 说明符告诉 Qlik Sense 将第一个非排除的行解释为包含字段标签。

```
LOAD Col1, Col2
FROM 'lib://files/header.txt'
(txt, embedded labels, delimiter is ',', msq, header is 1 lines);
```

结果是包含两个字段的表格：Col1 和 Col2。

另请参见：

- *Load (page 91)*

Record is

对于固定记录长度文件，必须通过 **record is** 说明符来指定记录长度。

语法：

```
Record is n
Record is line
Record is n lines
```

参数：

参数

参数	说明
n	指定记录长度(以字节为单位)。
line	指定记录长度(以一行为单位)。
n lines	指定记录长度(以行为单位)，其中 n 是一个正整数，表示记录长度。

限制：

record is 说明符仅用于 **fix** 文件。

另请参见：

Load (page 91)

Quotes

Quotes 是 **LOAD** 语句(定义引号是否可以使用以及引号与分隔符之间的优先级)的文件说明符。仅限文本文件

语法：

```
no quotes
msq
```

如果省略说明符，将使用标准引用，即可以使用引号 "" 或 ''，但只有当其是字段值的首个或最后一个非空白字符时才行。

参数：

参数

参数	说明
no quotes	只有在文本文件中无法使用引号时才使用。
msq	用于指定新样式引用，并允许字段包括多行内容。包含行尾结束字符的字段必须用双引号括起来。 msq 选项有一个限制，即单个双引号 (") 字符作为字段内容的第一个或最后一个字符出现时，将被解释为多行内容的开始或结尾，这可能会导致加载的数据集出现不可预知的后果。在这种情况下，应改为使用标准引用并省略说明符。

XML

当加载 xml 文件时使用此脚本说明符。在语法中列出了适用于 **XML** 说明符的有效选项。

您不能在 Qlik Sense 中加载 DTD 文件。

语法：

```
xmlsimple
```

另请参见：

- *Load (page 91)*

KML

当加载用于图形可视化的 KML 文件时，使用此脚本说明符。

语法：

```
kml
```

KML 文件可以表示区域数据（例如，国家或地区，由多边形表示）、线路数据（例如轨道或道路）或数据点（例如，城市或地点，以 [经度，纬度] 格式的点来表示）。

URL is

该脚本说明符用于设置加载 Web 文件时 Web 文件数据连接的 URL。

语法：

```
URL is string
```

参数：

参数

参数	说明
string	指定以加载的文件的 URL。这将覆盖在使用的 Web 文件连接中设置的 URL。

限制：

URL is 说明符仅用于 Web 文件。您需要使用现有 Web 文件数据连接。

另请参见：

- *Load (page 91)*

userAgent is

该脚本说明符用于在加载 Web 文件时设置浏览器用户代理。

语法：

```
userAgent is string
```

参数：

参数

参数	说明
string	指定浏览器用户代理字符串。这将覆盖默认浏览器用户代理 "Mozilla/5.0"。

限制：

userAgent is 说明符仅用于 Web 文件。

另请参见：

Load (page 91)

Let

let 语句是 **set** 语句的补充，用于定义脚本变量。相对于 **set** 语句，**let** 语句在其被分配到变量之前可以在脚本运行时计算等号“=”右边的表达式。

语法：

```
Let variablename=expression
```

示例和结果：

示例	结果
Set x=3+4; Let y=3+4; z=$(y)+1;	$(x) 将求值为“3+4” $(y) 将求值为“ 7” $(z) 将求值为“ 8” 请注意 **Set** 和 **Let** 语句之间的差异。**Set** 语句将字符串 '3+4' 赋值给变量，而 **Let** 语句对字符串求值并将 7 赋值给变量。
Let T=now( );	$(T) 将给定当前时间的值。

Loosen Table

在使用 **Loosen Table** 语句执行脚本期间，一个或多个 Qlik Sense 内部数据表格可以明确声明松散耦合。当表格是松散耦合时，表格中字段值之间的所有关联都会被移除。通过独立加载松散耦合表格（未相互连接的表格）的每个字段可以达到类似的效果。在对数据结构临时独立的不同部分进行测试时，松散耦合会非常有用。松散耦合表格在表格查看器中将以虚线进行标识。执行脚本之前，在脚本中使用一个或多个 **Loosen Table** 语句将使 Qlik Sense 忽略任何松散耦合表设置。

语法：

```
Loosen Tabletablename [ , tablename2 ...]
Loosen Tablestablename [ , tablename2 ...]
```

可使用以下语法之一：**Loosen Table** 或 **Loosen Tables**。

> *如果 Qlik Sense 在数据结构中发现循环引用，且此引用在脚本中不会被明确表现出交互式或显式松散组合的表格所中断，此时将强制松散组合一个或多个其他表格，直到无循环引用存在。出现这种情况时，**循环警告**对话框会发出警告。*

示例：

```
Tab1:
SELECT * from Trans;
Loosen Table Tab1;
```

Map

map ... using 语句用于将某些字段值或表达式映射为特定映射表的值。映射表可通过 **Mapping** 语句创建。

语法：

```
Map fieldlist Using  mapname
```

自动映射可用于在 **Map … Using** 语句之后，脚本末尾或 **Unmap** 语句之前加载的字段。

在生成由 Qlik Sense 内部表格存储的字段的事件链中，映射是最后环节。这意味着，并非每次遇到作为表达式组成部分的字段名时都会执行映射，而是在当值存储在内部表格中的字段名之下时才执行映射。如果要求执行表达式级映射，则必须使用 **Applymap()** 函数。

参数：

参数

参数	说明
fieldlist	用逗号分隔的字段列表，该列表应从脚本中的此点进行映射。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。
mapname	先前在 **mapping load** 或 **mapping select** 语句中读取的映射表的名称。

示例和结果：

示例	结果
Map Country Using Cmap;	使用映射 Cmap 启用字段 Country 的映射。
Map A, B, C Using X;	使用映射 X启用字段 A、B 和 C 的映射。
Map * Using GenMap;	使用 GenMap 启用所有字段的映射。

NullAsNull

NullAsNull 语句用于关闭 NULL 语句之前设置的从 **NullAsValue** 值到字符串值的转换。

语法：

```
NullAsNull *fieldlist
```

NullAsValue 语句可像开关一样运作，无论是使用 **NullAsValue** 还是 **NullAsNull** 语句，均可在脚本中开启或关闭数次。

参数：

参数

参数	说明
*fieldlist	用逗号分隔的字段列表，应针对该列表启用 **NullAsNull**。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。

示例：

```
NullAsNull A,B;
LOAD A,B from x.csv;
```

NullAsValue

NullAsValue 语句用于指定应将 NULL 值转换为值的字段。

语法：

```
NullAsValue *fieldlist
```

Qlik Sense 默认将 NULL 值视为缺失或未定义实体。但是，某些数据库上下文暗示可将 NULL 值视为特殊值，而不是缺失值。通常不允许将 NULL 值链接至可通过 NULL 语句挂起的其他 **NullAsValue** 值。

NullAsValue 语句可像开关一样运作，且可在后续的加载语句中运作。您可借助 **NullAsNull** 语句再次切换关闭该语句。

参数：

参数

参数	说明
*fieldlist	用逗号分隔的字段列表，应针对该列表启用 **NullAsValue**。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。

示例：

```
NullAsValue A,B;
Set NullValue = 'NULL';
LOAD A,B from x.csv;
```

Qualify

Qualify 语句用于打开字段名限制条件，即字段名将表格名作为前缀。

语法：

```
Qualify *fieldlist
```

使用 **qualify** 语句可以暂时中止不同表格内具有相同名称的字段之间的自动关联，同时该语句可以使用表格名限定字段名。如果限定，则会在表格中找到时重命名字段名。新名称的格式为 *tablename.fieldname*。*Tablename* 等同于当前表格的标签，或者如果标签不存在，则等同于显示在 **LOAD** 和 **SELECT** 语句中 **from** 之后的名称。

qualify 语句将对在其后加载的所有字段将进行限定。

脚本执行开始时始终默认打开限制条件。字段名称限定可随时使用限定 **qualify** 语句激活。使用 **Unqualify** 语句可随时关闭限制条件。

> ***qualify** 语句不得与部分重新加载结合使用。*

参数：

参数

参数	说明
*fieldlist	用逗号分隔的字段列表，为此限定应开户。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。

Example 1:

```
Qualify B;
LOAD A,B from x.csv;
LOAD A,B from y.csv;
```

只能通过 **A** 关联两个表格 **x.csv** 和 **y.csv**。将会生成三个字段：A、x.B、y.B。

Example 2:

在不熟悉的数据库中，首先确保仅一个或少数字段关联，这样做通常有用，如以下示例所示：

```
qualify *;
unqualify TransID;
SQL SELECT * from tab1;
SQL SELECT * from tab2;
```

```
SQL SELECT * from tab3;
```
在表格 *tab1*、*tab2* 和 *tab3* 之间只能使用 **TransID** 进行关联。

Rem

rem 语句用于插入备注或注释到脚本，或暂时关闭脚本语句而无需移除脚本。

语法：

```
Rem string
```

rem 和下一个分号 **;** 之间的一切内容都视为注释。

在脚本中注释有两种替代性方法：

1. 通过将有问题的部分放置在 **/*** 和 ***/** 之间可在脚本的任何位置创建注释（两个引号之间除外）。
2. 当在脚本中输入 **//** 时，同一行右方的所有文本都将成为注释。（请注意，*//*: 的例外情况是可以用作网址的一部分。）

参数：

参数

参数	说明
string	任意文本。

示例：

```
Rem ** This is a comment **;
/* This is also a comment */
// This is a comment as well
```

Rename

Rename 脚本关键字可用于重命名已经加载的表格或字段。

Rename field

此脚本函数用于在加载一个或多个现有 Qlik Sense 字段后对其进行重命名。

建议不对 Qlik Sense 中的字段和函数将变量命名为相同的名称。

可使用以下语法之一：**rename field** 或 **rename fields**。

语法：

```
Rename Field (using mapname | oldname to newname{ , oldname to newname })
Rename Fields (using mapname | oldname to newname{ , oldname to newname })
```

参数：

参数	说明
mapname	先前加载的映射表的名称，其中包含一对或多对旧的和新的字段名。
oldname	旧的字段名称。
newname	新的字段名称。

限制：

您不能将两个字段重命名成同样的名称。

Example 1:

```
Rename Field XAZ0007 to Sales;
```

Example 2:

```
FieldMap:
Mapping SQL SELECT oldnames, newnames from datadictionary;
Rename Fields using FieldMap;
```

Rename table

此脚本函数用于在加载一个或多个现有 Qlik Sense 内部表格后对其进行重命名。

可使用以下语法之一：**rename table** 或 **rename tables**。

语法：

```
Rename Table (using mapname | oldname to newname{ , oldname to newname })
Rename Tables (using mapname | oldname to newname{ , oldname to newname })
```

参数：

参数

参数	说明
mapname	先前加载的映射表的名称，其中包含一对或多对旧的和新的表格名。
oldname	旧的表格名称。
newname	新的表格名称。

限制：

两个不同名称的表格不能重命名成相同的名称。如果尝试将表格重命名为与现有表格相同的名称，则脚本将生成错误。

Example 1:

```
Tab1:
SELECT * from Trans;
Rename Table Tab1 to Xyz;
```

Example 2:

```
TabMap:
Mapping LOAD oldnames, newnames from tabnames.csv;
Rename Tables using TabMap;
```

Search

Search 语句用于在智能搜索中包含或排除字段。

语法：

```
Search Include *fieldlist
Search Exclude *fieldlist
```

可以使用多个I Search 语句来优化要选择包含在内的字段。语句自上而下求值。

参数：

参数

参数	说明
*fieldlist	要在智能搜索中包含或排除搜索的字段的逗号分隔列表。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。

示例：

搜索示例

语句	说明
`Search Include *;`	在智能搜索中包含搜索的所有字段。
`Search Exclude [*ID];`	在智能搜索中排除搜索以 ID 结尾的所有字段。
`Search Exclude '*ID';`	在智能搜索中排除搜索以 ID 结尾的所有字段。
`Search Include ProductID;`	在智能搜索中包含搜索的 ProductID 字段。

按照此顺序，这三个语句的合并结果是从智能搜索中排除搜索以 ID 结尾但不含 ProductID 的所有字段。

Section

使用 **section** 语句，可以定义随后的 **LOAD** 和 **SELECT** 语句应视为数据还是访问权限定义。

语法：

```
Section (access | application)
```

如果未指定任何一个，将假设为 **section application**。**section** 定义会一直有效，直到新的 **section** 语句出现。

示例：

```
Section access;
Section application;
```

Select

可通过标准 SQL **SELECT** 语句从 ODBC 数据源或 OLE DB 提供商选择字段。然而，是否接受 **SELECT** 语句取决于所使用的 ODBC 驱动程序或 OLE DB 提供程序。使用 **SELECT** 语句，需要指向源的开放数据连接。

语法：

```
Select [all | distinct | distinctrow | top n [percent] ] fieldlist

From tablelist

[where criterion ]

[group by fieldlist [having criterion ] ]

[order by fieldlist [asc | desc] ]

[ (Inner | Left | Right | Full) join tablename on fieldref = fieldref ]
```

而且，几个 **SELECT** 语句可通过使用 **union** 运算符结合成一个整体：

```
selectstatement Union selectstatement
```

SELECT 语句由 ODBC 驱动程序或 OLE DB 提供者解释，因此可能会发生一般的 SQL 语法偏差，具体取决于 ODBC 驱动程序的功能或 OLE DB 提供者，例如：

- 有时，不允许使用 **as**，即 *aliasname* 必须紧跟在 *fieldname* 之后。
- 如果使用 **as**，有时会强制使用 *aliasname*。
- 有时，不支持 **distinct**、**as**、**where**、**group by**、**order by** 或 **union**。
- 有时，ODBC 驱动程序不支持所有以上列出的不同引号。

*这不是完整的 SQL **SELECT** 语句！例如，**SELECT** 语句可以嵌套，可在一个 **SELECT** 语句中创建几个连接，表达式中允许的函数个数有时非常大等。*

参数:

参数

参数	说明
distinct	**distinct** 是一个在所选字段中的值的重复组合只应加载一次时使用的谓词。
distinctrow	**distinctrow** 是一个在源表格中的重复记录只应加载一次时使用的谓词。
fieldlist	**fieldlist ::= (*| field) {, field }** 要选择的字段列表。使用 * 作为字段列表,表示表格中全部字段。 **fieldlist ::= field {, field }** 一个或多个字段的列表,用逗号分开。 **field ::= (fieldref| expression) [as aliasname]** 例如表达式可以为一个基于一个或几个其他字段的数字或字符串函数。通常接受的一些运算符和函数为:+、-、*、/、&(字符串串联运算) sum(fieldname)、count (fieldname)、avg(fieldname)(average)、month(fieldname) 等。有关详细信息,请参阅 ODBC 驱动程序的文档。 **fieldref ::= [tablename.] fieldname** **tablename** 和 **fieldname** 是它们表示的意思相似的文本字符串。如果他们包含空格则它们必须包括在直双引号内。 **as** 子句用于为字段分配一个新名。
from	**tablelist ::= table {, table }** 要从其中选择字段表格列表。 **table ::= tablename [[as] aliasname]** **tablename**可以也可以不放在引号内。
where	**where** 是一个子句,用于陈述一个记录是否应该包括在选择项内。 **criterion** 是一个逻辑表达式,有时可能会非常复杂。以下是可以接受的一些运算符:数值运算符和函数、**=**、**<>** 或 **#(**不等于) 、**>**、**>=**、**<**、**<=**、**and**、**or**、**not**、**exists**、**some**、**all**、**in** 和新的 **SELECT** 语句。有关详细信息,请参阅文档 ODBC 驱动程序或 OLE DB 提供者。
group by	**group by** 是一个子句,用于将几个记录聚合(组成) 为一个整体。对于某些字段来说,在一个组中,所有记录要么拥有一个相同的值,要么字段只能用于一个表达式内,如作为合计或平均值。基于一个或几个字段的表达式在字段符号的表达式中定义。
having	**having** 是一个子句,用于以一种与 **where** 子句在限定记录时相同的使用方式限定组。
order by	**order by** 是一个用于表述 **SELECT** 语句的结果表排序顺序的子句。
join	**join** 是一个限定符,用于表述几个表格是否应联接为一个整体。字段名及表格名如果包含空格或来自国际字符集的字母则必须被放进引号内。脚本由 Qlik Sense 自动生成时,使用的引号应为在 ODBC 语句的数据源定义中指定的 OLE DB 驱动程序或 **Connect** 提供者偏好的引号。

Example 1:

```
SELECT * FROM `Categories`;
```

Example 2:

```
SELECT `Category ID`, `Category Name` FROM `Categories`;
```

Example 3:

```
SELECT `Order ID`, `Product ID`,

`Unit Price` * Quantity * (1-Discount) as NetSales

FROM `Order Details`;
```

Example 4:

```
SELECT `Order Details`.`Order ID`,

Sum(`Order Details`.`Unit Price` * `Order Details`.Quantity) as `Result`

FROM `Order Details`, Orders

where Orders.`Order ID` = `Order Details`.`Order ID`

group by `Order Details`.`Order ID`;
```

Set

set 语句用于定义脚本变量。这些变量可用来替代字符串，路径和驱动程序等。

语法：

```
Set variablename=string
```

Example 1:

```
Set FileToUse=Data1.csv;
```

Example 2:

```
Set Constant="My string";
```

Example 3:

```
Set BudgetYear=2012;
```

Sleep

sleep 语句用于在指定的时间暂停脚本执行。

语法：

```
Sleep n
```

参数：

参数	说明
n	以毫秒为单位表示，其中 *n* 是一个正整数，且不得大于 3600000(即 1 小时)。该值也可以是一个表达式。

Example 1:

```
Sleep 10000;
```

Example 2:

```
Sleep t*1000;
```

SQL

SQL 语句可通过 ODBC 或 OLE DB 连接发送任意 SQL 命令。

语法：

```
SQL sql_command
```

如果 Qlik Sense 已经以只读模式打开 ODBC 连接，发送更新数据库的 SQL 语句将会返回错误。

相应语法为：

```
SQL SELECT * from tab1;
```

允许使用，并且考虑到一致性，将作为 **SELECT** 的首选语法。但是，SQL 前缀仍然是 **SELECT** 语句的可选项。

参数：

参数	说明
sql_command	有效的 SQL 命令。

Example 1:

```
SQL leave;
```

Example 2:

```
SQL Execute <storedProc>;
```

SQLColumns

sqlcolumns 语句用于返回描述 ODBC 或 OLE DB 数据源的列以建立 **connect** 的字段集。

语法：

```
SQLcolumns
```

这些字段可以与 **sqltables** 和 **sqltypes** 命令生成的字段组合，以概述给定的数据库。这十二个标准字段为：

TABLE_QUALIFIER

TABLE_OWNER

TABLE_NAME

COLUMN_NAME

DATA_TYPE

TYPE_NAME

PRECISION

LENGTH

SCALE

RADIX

NULLABLE

REMARKS

关于这些字段的详细说明，请参阅 ODBC 参考手册。

示例：

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\DataSrc\QWT.mbd';
SQLcolumns;
```

某些 ODBC 驱动程序可能不支持此命令。某些 ODBC 驱动程序可能不会产生额外字段。

SQLTables

sqltables 语句用于返回描述 ODBC 或 OLE DB 数据源的表格以建立 **connect** 的字段集。

语法：

```
SQLTables
```

这些字段可以与 **sqlcolumns** 和 **sqltypes** 命令生成的字段组合，以概述给定的数据库。这五个标准字段为：

TABLE_QUALIFIER

TABLE_OWNER

TABLE_NAME

TABLE_TYPE

REMARKS

关于这些字段的详细说明，请参阅 ODBC 参考手册。

示例：

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\DataSrc\QWT.mbd';
SQLTables;
```

某些 ODBC 驱动程序可能不支持此命令。某些 ODBC 驱动程序可能不会产生额外字段。

SQLTypes

sqltypes 语句用于返回描述 ODBC 或 OLE DB 数据源的类型以建立 **connect** 的字段集。

语法：

```
SQLTypes
```

这些字段可以与 **sqlcolumns** 和 **sqltables** 命令生成的字段组合，以概述给定的数据库。这十五个标准字段为：

TYPE_NAME

DATA_TYPE

PRECISION

LITERAL_PREFIX

LITERAL_SUFFIX

CREATE_PARAMS

NULLABLE

CASE_SENSITIVE

SEARCHABLE

UNSIGNED_ATTRIBUTE

MONEY

AUTO_INCREMENT

LOCAL_TYPE_NAME

MINIMUM_SCALE

MAXIMUM_SCALE

关于这些字段的详细说明，请参阅 ODBC 参考手册。

示例：

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\DataSrc\QWT.mbd';
SQLTypes;
```

某些 ODBC 驱动程序可能不支持此命令。某些 ODBC 驱动程序可能不会产生额外字段。

Star

该字符串用于呈现数据库中字段的全部值设置，可以通过 **star** 语句设置。星号可以影响随后的**LOAD** 和 **SELECT** 语句。

语法：

```
Star is[ string ]
```

参数：

参数

参数	说明
string	任意文本。请注意，如果字符串包含空串，则必须用引号引起来。 如果未指定任何一项，将假设为 **star is;**，即无星号可用，除非明确指定。此定义会一直有效，直到新的 **star** 语句出现。

如果使用了区域权限，不建议将 **Star is** 语句用于脚本的数据部分（在 **Section Application** 下）。但在脚本的**区域权限**部分，对于受保护字段完全支持星号字符。在该情况下，您不需要使用显式的 **Star is** 语句，因为这在区域权限中始终是隐式的。

限制

- 您无法将星号字符结合关键字段使用；即链接表格的字段。
- 您无法将星号字符结合受 **Unqualify** 语句影响的任何字段使用，因为这可能影响链接表格的字段。
- 您无法将星号字符结合非逻辑表格使用，例如信息加载表格或映射加载表格。
- 如果在区域权限中的减少字段（与数据链接的字段）中使用星号字符，则其表示在区域权限的该字段中列出的值。它不代表可能存在于数据中但未在区域权限中列出的其他值。
- 您无法将星号字符结合受**区域权限**以外任何形式的数据减少影响的字段使用。

示例

以下示例是采用区域权限提取数据加载脚本。

```
Star is *;


Section Access;

LOAD * INLINE [

ACCESS, USERID, OMIT

ADMIN, ADMIN,

USER, USER1, SALES

USER, USER2, WAREHOUSE

USER, USER3, EMPLOYEES

USER, USER4, SALES

USER, USER4, WAREHOUSE

USER, USER5, *

];


Section Application;

LOAD * INLINE [

SALES, WAREHOUSE, EMPLOYEES, ORDERS

1, 2, 3, 4

];
```

适用以下条件：

- *Star* 号为 *。
- 用户 *ADMIN* 看到所有字段。没有被省略的内容。
- 用户 *USER1* 看不到字段 *SALES*。
- 用户 *USER2* 看不到字段 *WAREHOUSE* 。
- 用户 *USER3* 看不到字段 *EMPLOYEES*。
- 将用户 *USER4* 添加两次以解决此用户的两个 OMIT 字段：*SALES* 和 *WAREHOUSE*。

- *USER5*添加了一个"*", 这意味着 OMIT 中列出的所有字段都不可用, 即用户 *USER5*看不见字段 *SALES*、*WAREHOUSE* 和 *EMPLOYEES* , 但该用户可看见字段 *ORDERS*。

Store

Store 语句创建 QVD、CSV 或 text 文件。

语法:

```
Store [ fieldlist from] table into filename [ format-spec ];
```

该语句将创建一个明确命名的 QVD、CSV 或 TXT 文件。

该语句仅会从一个数据表格中导出字段。如果要从多个表格中导出字段, 必须明确命名之前在脚本中生成的 join 以创建应导出的数据表。

文本值将以 UTF-8 格式导出至 CSV 文件。可以指定一个分隔符, 请参阅 **LOAD**。**store** 语句不支持将 CSV 导出至 BIFF 文件。

参数:

存储命令参数

参数	说明
*fieldlist::= (* | field) { , field })*	要选择的字段列表。使用 * 作为字段列表, 则其表示全部字段。 *field::= fieldname [***as** *aliasname]* *fieldname*是指与*table*中的字段名完全相同的文本。(请注意, 如果字段名包含空格或其他非标准字符, 则必须使用双引号或方括号括起来。) *aliasname* 是指生成的 QVD 或 CSV 文件中所用字段的替代名称。
table	脚本标签表示要用作数据源的已加载表格。

参数	说明
filename	目标文件的名称，包括现有文件夹数据连接的有效路径。 **示例：** ***'lib://Table Files/target.qvd'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** ***c:\data\sales.qvd*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** ***data\sales.qvd*** 如果路径被省略，Qlik Sense 则在由 **Directory** 语句指定的目录中存储文件。如果没有 **Directory** 语句，那么 Qlik Sense 将在工作目录 *C:\Users\{user}\Documents\Qlik\Sense\Apps* 中存储文件。
*format-spec ::=***(** (**txt** | **qvd**) **)**	格式规范包含文本 **txt**(对于文本文件) 或文本 **qvd**(对于 qvd 文件) 。如果省略格式规范，则假定为 **qvd**。

示例：

```
Store mytable into xyz.qvd (qvd);

Store * from mytable into 'lib://FolderConnection/myfile.qvd';

Store Name, RegNo from mytable into xyz.qvd;

Store Name as a, RegNo as b from mytable into 'lib://FolderConnection/myfile.qvd';

Store mytable into myfile.txt (txt);

Store * from mytable into 'lib://FolderConnection/myfile.qvd';
```

DataFiles 连接的文件扩展要区分大小写。例如：.qvd。

Table/Tables

Table 和 **Tables** 脚本关键字用于 **Drop**、**Comment** 和 **Rename** 语句，以及 **Load** 语句中的格式说明符。

Tag

此脚本语句提供了一种将标记分配给一个或多个字段或表格的方法。如果试图标记应用程序中不存在的字段或表格，则将忽略该标记。如果出现字段名和标记名冲突的情况，将使用最后出现的值。

语法：

```
Tag [field|fields] fieldlist with tagname
```

```
Tag [field|fields] fieldlist using mapname
```

```
Tag table tablelist with tagname
```

参数

参数	说明
fieldlist	在逗号分隔的列表中，应标记的一个或多个字段。
mapname	先前在 **mapping Load** 或 **mapping Select** 语句中加载的映射表的名称。
tablelist	应标记的表的逗号分隔列表。
tagname	要应用到字段的标记名称。

Example 1:

```
tagmap:
mapping LOAD * inline [
a,b
Alpha,MyTag
Num,MyTag
];
tag fields using tagmap;
```

Example 2:

```
tag field Alpha with 'MyTag2';
```

Trace

trace 语句用于将字符串写入**脚本执行进度**窗口和脚本日志文件中。这对调试过程很有用。使用在 **trace** 语句之前计算的 $ 变量扩展可以自定义消息。

语法：

```
Trace string
```

Example 1:

下面的语句可以在加载“主”表的 Load 语句之后使用。

```
Trace Main table loaded;
```

这将在脚本执行对话框和日志文件中显示文本“主表已加载”。

Example 2:

下面的语句可以在加载“主”表的 Load 语句之后使用。

```
Let MyMessage = NoOfRows('Main') & ' rows in Main table';

Trace $(MyMessage);
```

这将显示一个文本，显示脚本执行对话框和日志文件中的行数，例如，“主表中的 265,391 行”。

Unmap

Unmap 语句可禁用由之前的 **Map … Using** 语句为后来加载的字段指定的字段值映射。

语法：

```
Unmap *fieldlist
```

参数：

参数

参数	说明
*fieldlist	用逗号分隔的字段列表，该列表不再从脚本中的此点进行映射。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。

示例和结果：

示例	结果
Unmap Country;	禁止映射字段 Country。
Unmap A, B, C;	禁止映射字段 A、B 和 C。
Unmap * ;	禁止映射全部字段。

Unqualify

Unqualify 语句用于断开前面由 **Qualify** 语句打开的字段名限制条件。

语法：

```
Unqualify *fieldlist
```

参数：

参数

参数	说明
*fieldlist	用逗号分隔的字段列表，为此限定应开户。使用 * 作为字段列表，则其表示全部字段。在字段名称中允许使用通配符 * 和 ?。使用通配符时可能需要将字段名引起来。 有关详细信息，请参阅文档 **Qualify** 语句。

Example 1:

在不熟悉的数据库中，首先确保仅一个或少数字段关联，这样做通常有用，如以下示例所示：

```
qualify *;
unqualify TransID;
SQL SELECT * from tab1;
SQL SELECT * from tab2;
SQL SELECT * from tab3;
```

首先，为所有字段启用限定。
然后为 **TransID** 关闭限定。
在表格 *tab1*、*tab2* 和 *tab3* 之间只能使用 **TransID** 进行关联。所有其他字段都将使用表名进行限定。

Untag

此脚本语句提供了一种从字段或表中删除标记的方法。如果试图取消标记应用程序中不存在的字段或表格，则将忽略该取消标记。

语法：

```
Untag [field|fields] fieldlist with tagname

Untag [field|fields] fieldlist using mapname

Untag table tablelist with tagname
```

参数：

参数

参数	说明
fieldlist	在逗号分隔的列表中，应移除标记的一个或多个字段。
mapname	先前在映射 **LOAD** 或映射 **SELECT** 语句中加载的映射表的名称。
tablelist	应取消标记的表的逗号分隔列表。
tagname	应从字段移除的标记名称。

Example 1:

```
tagmap:
mapping LOAD * inline [
a,b
Alpha,MyTag
Num,MyTag
];
Untag fields using tagmap;
```

Example 2:

```
Untag field Alpha with MyTag2;
```

2.6 工作目录

如果在脚本语句中引用文件，并且省略路径，则 Qlik Sense 会按以下顺序搜索该文件：

1. **Directory** 语句指定的目录(仅传统脚本模式支持)。
2. 如果没有 **Directory** 语句，则 Qlik Sense 将在工作目录中搜索。

Qlik Sense Desktop 工作目录

在 Qlik Sense Desktop 中，工作目录为 *C:\Users\{user}\Documents\Qlik\Sense\Apps*。

Qlik Sense 工作目录

在 Qlik Sense 服务器安装中，工作目录是在 Qlik Sense 存储库服务 中指定，默认情况下是 *C:\ProgramData\Qlik\Sense\Apps*。有关详细信息，请参阅 Qlik Management Console 帮助。

2 在数据加载编辑器中使用变量

Qlik Sense 中的变量是存储静态值或计算(例如数字或字母数字值)的容器。在应用程序中使用此变量时,对此变量做出的任何更改将应用于使用此变量的任何位置。您可在变量概述中定义变量,或利用数据加载编辑器在脚本中定义。您可在数据加载脚本中使用 **Let** 或 **Set** 语句设置变量的值。

编辑表格时,您也可以从变量概述使用 Qlik Sense 变量。

2.7 概述

如果变量值的第一个字符为等于符号"=", Qlik Sense 将尝试以公式(Qlik Sense 表达式)评估该值,然后显示或返回结果而不是实际公式文本。

使用时,此变量用其值取代。脚本中的变量可用于货币符号扩展脚本和各种控制语句。如果同一字符串(如路径)在脚本中重复多次,则其将非常有用。

部分特别的系统变量将由 Qlik Sense 在开始执行脚本时设置,不管之前为何值。

2.8 定义变量

变量提供了存储静态值或计算结果的能力。定义变量时,使用以下语法:

```
set variablename = string
```

或

```
let variable = expression
```

该 **Set** 语句用于字符串赋值。它将等号右侧的文本赋值给变量。**Let** 语句在脚本运行时对等号右侧的表达式求值,并将表达式的结果赋给该变量。

变量区分大小写。

建议不对 Qlik Sense 中的字段和函数将变量命名为相同的名称。

示例:

```
set x = 3 + 4; // 该变量将获得字符串"3 + 4"作为值。

let x = 3 + 4; // 返回 7 作为值。

set x = Today(); // 返回"Today()"作为值。

let x = Today(); // 返回今天的日期作为值,例如,"9/27/2021"。
```

2.9 删除变量

如果您从脚本移除变量并重新加载数据，变量将留在应用程序中。如果您想要从应用程序中完全移除变量，必须也从变量对话框删除变量。

2.10 将变量值加载为字段值

如果要在 **LOAD** 语句中加载变量值作为字段值且货币符号扩展的结果为文本(而非数字或表达式)，则需要用单引号将扩展变量引起来。

示例：

本例将包含脚本错误列表的系统变量加载到表格中。您会发现 **If** 子句中的 ScriptErrorCount 扩展变量不需要使用引号，而 ScriptErrorList 扩展变量需要使用引号。

```
IF $(ScriptErrorCount) >= 1 THEN

   LOAD '$(ScriptErrorList)' AS Error AutoGenerate 1; END IF
```

2.11 变量计算

可以通过多种方法在 Qlik Sense 中使用变量计算值，结果取决于定义变量以及在表达式中调用变量的方式。

在本例中，我们加载一些内联数据：

```
LOAD * INLINE [
    Dim, Sales
    A, 150
    A, 200
    B, 240
    B, 230
    C, 410
    C, 330
];
```

首先需要定义两个变量：

```
Let vSales  = 'Sum(Sales)' ;
Let vSales2  = '=Sum(Sales)' ;
```

在第二个变量中，在表达式前面添加一个等号。这可以使得在扩展变量和计算表达式之前计算变量。

如果按此方法使用 vSales 变量(如在度量中)，则结果将为字符串 Sum(Sales)，即没有执行任何计算。

如果在表达式中添加货币符号扩展和调用 $(vSales)，则该变量已扩展且显示Sales的总和。

最后，如果调用 $(vSales2)，则将会在扩展变量之前计算其值。这意味着所显示的结果是Sales的总和。使用 =$(vSales) 和 =$(vSales2) 作为度量表达式之间的区别如此图表显示的结果所示：

结果

Dim	$(vSales)	$(vSales2)
A	350	1560
B	470	1560
C	740	1560

如图表所示，$(vSales) 生成维度值的部分和，而 $(vSales2) 却生成总和。

以下脚本变量可用：

- *错误变量 (page 150)*
- *数字解释变量 (page 137)*
- *系统变量 (page 130)*
- *值处理变量 (page 135)*

2.12 系统变量

一部分系统变量由系统所定义，用于提供有关系统和 Qlik Sense 应用程序的信息。

系统变量概述

一部分函数在概述后面进行了详细描述。对于这些函数，可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Floppy

用于返回找到的第一个软盘驱动器的驱动器号，通常是 *a:*。这是系统定义的变量。

Floppy

在标准模式下不支持此变量。

CD

用于返回找到的第一个 CD-ROM 驱动器的驱动器号。如果未找到任何 CD-ROM，随后会返回 *c:*。这是系统定义的变量。

CD

在标准模式下不支持此变量。

Include

Include/Must_Include 变量用于指定包含应包括在脚本中并作为脚本代码计算值的文本的文件。它不用于添加数据。您可以将部分脚本代码存储在单独的文本文件中，并可以在多个应用程序中重复使用它。这是用户定义的变量。

```
$(Include=filename)
$(Must_Include=filename)
```

HidePrefix

所有以此文本字符串开始的字段名会按与系统字段相同的方式隐藏。这是用户定义的变量。

```
HidePrefix
```

HideSuffix

所有以此文本字符串结束的字段名会按与系统字段相同的方式隐藏。这是用户定义的变量。

```
HideSuffix
```

QvPath

用于返回浏览字符串到可执行的 Qlik Sense 文件。这是系统定义的变量。

```
QvPath
```

在标准模式下不支持此变量。

QvRoot

用于返回可执行的 Qlik Sense 的根目录。这是系统定义的变量。

```
QvRoot
```

在标准模式下不支持此变量。

QvWorkPath

用于返回浏览字符串到当前 Qlik Sense 应用程序。这是系统定义的变量。

```
QvWorkPath
```

在标准模式下不支持此变量。

QvWorkRoot

用于返回当前 Qlik Sense 应用程序的根目录。这是系统定义的变量。

```
QvWorkRoot
```

在标准模式下不支持此变量。

StripComments

如果此变量设置为 0，则禁止剥离脚本中的 /*..*/ 和 // 注释。如果未定义此变量，则会始终执行注释剥离。

```
StripComments
```

Verbatim

全部字段值通常会自动剥离前导和尾部空格 (ASCII 32)，然后再加载到 Qlik Sense 数据库。将此变量设置为 1 会暂停剥离空格。Tab (ASCII 9) 和硬空格 (ANSI 160) 字符永远都不会剥离。

```
Verbatim
```

OpenUrlTimeout

此变量用于定义 Qlik Sense 应在从 URL 源（如HTML 页面）获取数据时考虑的超时（秒）。如果省略，则超时约为 20 分钟。

```
OpenUrlTimeout
```

WinPath

用于返回浏览字符串到 Windows。这是系统定义的变量。

```
WinPath
```

在标准模式下不支持此变量。

WinRoot

返回 Windows 的根目录。这是系统定义的变量。

```
WinRoot
```

在标准模式下不支持此变量。

CollationLocale

指定要用于排序顺序和搜索匹配的区域设置。该值是区域设置的区域性名称，如“zh-CN”。这是系统定义的变量。

```
CollationLocale
```

CreateSearchIndexOnReload

此变量用于定义是否应在数据重新加载期间创建搜索索引文件。

```
CreateSearchIndexOnReload
```

CreateSearchIndexOnReload

此变量用于定义是否应在数据重新加载期间创建搜索索引文件。

语法：

```
CreateSearchIndexOnReload
```

您可以定义是否在数据重新加载期间创建搜索索引文件，或是否在用户首次发出搜索请求后创建搜索索引文件。在数据重新加载期间创建搜索索引文件的好处是可以避免用户首次进行搜索时的等待时间。这需要根据搜索索引创建所需的较长数据重新加载时间进行加权。

如果省略此变量，则不会在数据重新加载期间创建搜索索引文件。

对于会话应用程序，无论此变量的设置为何，都不会在数据重新加载期间创建搜索索引文件。

Example 1: 数据重新加载期间创建搜索索引字段

```
set CreateSearchIndexOnReload=1;
```

Example 2: 首次搜索请求后创建搜索索引字段

```
set CreateSearchIndexOnReload=0;
```

HidePrefix

所有以此文本字符串开始的字段名会按与系统字段相同的方式隐藏。这是用户定义的变量。

语法：

```
HidePrefix
```

示例：

```
set HidePrefix='_' ;
```

如果使用此语句，当系统字段隐藏时，始于下划线的字段名不会显示在字段名称列表中。

HideSuffix

所有以此文本字符串结束的字段名会按与系统字段相同的方式隐藏。这是用户定义的变量。

语法：

```
HideSuffix
```

示例：

```
set HideSuffix='%';
```

如果使用此语句，当系统字段隐藏时，以百分比符号结束的字段名不会显示在字段名称列表中。

Include

Include/Must_Include 变量用于指定包含应包括在脚本中并作为脚本代码计算值的文本的文件。它不用于添加数据。您可以将部分脚本代码存储在单独的文本文件中，并可以在多个应用程序中重复使用它。这是用户定义的变量。

此变量仅在标准模式下支持文件夹数据连接。

语法：

```
$(Include=filename)
```

```
$(Must_Include=filename)
```

变量有两个版本：

- **Include** 在找不到文件的情况下不会生成错误，而会静默失败。
- **Must_Include** 在找不到文件的情况下会生成错误。

如果不指定路径，文件名将相对于 Qlik Sense 应用程序工作目录。也可以指定绝对文件路径，或者 lib:// 文件夹连接的路径。请勿在等号前后放置空格字符。

*构造函数 **set Include =**filename 不适用。*

示例：

```
$(Include=abc.txt);
```

```
$(Must_Include=lib://DataFiles/abc.txt);
```

限制

Windows 和 Linux 下 UTF-8 编码文件之间的交叉兼容性有限。

可以选择将 UTF-8 与 BOM(字节顺序标记) 一起使用。BOM 可能会干扰 UTF-8 在软件中的使用，该软件不希望在文件的开头使用非 ASCII 字节，但可以处理文本流。

- Windows 系统使用 UTF-8 中的 BOM 来标识文件是 UTF-8 编码的，尽管字节存储中没有歧义。
- Unix/Linux 对 Unicode 使用 UTF-8，但不使用 BOM，因为这会干扰命令文件的语法。

这对 Qlik Sense 有一些意义。

- 在 Windows 中，任何以 UTF-8 BOM 开头的文件都被视为 UTF-8 脚本文件。否则会假设 ANSI 编码。
- 在 Linux 中，系统默认的 8 位代码页是 UTF-8。这就是为什么 UTF-8 可以工作的原因，尽管它不包含 BOM。

因此，无法保证便携性。在 Windows 上创建一个可以被 Linux 解释的文件并非总是可能，反之亦然。由于对 BOM 的不同处理，两个系统之间对于 UTF-8 编码文件没有交叉兼容性。

OpenUrlTimeout

此变量用于定义 Qlik Sense 应在从 URL 源(如HTML 页面) 获取数据时考虑的超时 (秒) 。如果省略，则超时约为 20 分钟。

语法：

```
OpenUrlTimeout
```

示例：

```
set OpenUrlTimeout=10;
```

StripComments

如果此变量设置为 0，则禁止剥离脚本中的 /*..*/ 和 // 注释。如果未定义此变量，则会始终执行注释剥离。

语法：

```
StripComments
```

某些数据库驱动程序使用 /*..*/ 作为 **SELECT** 语句中的优化提示。如果出现这种情况，在将 **SELECT** 语句发送到数据库驱动程序之前不会去除注释。

我们强烈建议此变量在所需语句执行之后立即重置为 1。

示例：

```
set StripComments=0;
SQL SELECT * /* <optimization directive> */ FROM Table ;
set StripComments=1;
```

Verbatim

全部字段值通常会自动剥离前导和尾部空格 (ASCII 32)，然后再加载到 Qlik Sense 数据库。将此变量设置为 1 会暂停剥离空格。Tab (ASCII 9) 和硬空格 (ANSI 160) 字符永远都不会剥离。

语法：

```
Verbatim
```

示例：

```
set Verbatim = 1;
```

2.13 值处理变量

本节介绍用于处理 NULL 值和其他值的变量。

值处理变量概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

NullDisplay
定义的符号可以替代数据最低级别上 ODBC 的全部 NULL 值和连接器。这是用户定义的变量。

```
NullDisplay
```

NullInterpret
当定义的符号出现在文本文件，Excel 文件或内联语句中时将被解释为 NULL 值。这是用户定义的变量。

```
NullInterpret
```

NullValue
如果使用 **NullAsValue** 语句，定义的符号可以替代 NULL 指定包含指定字符串的字段中的全部 **NullAsValue** 值。

```
NullValue
```

OtherSymbol
在 **LOAD/SELECT** 语句前定义需要用作“全部其他值”的符号。这是用户定义的变量。

```
OtherSymbol
```

NullDisplay

定义的符号可以替代数据最低级别上 ODBC 的全部 NULL 值和连接器。这是用户定义的变量。

语法：

```
NullDisplay
```

示例：

```
set NullDisplay='<NULL>';
```

NullInterpret

当定义的符号出现在文本文件，Excel 文件或内联语句中时将被解释为 NULL 值。这是用户定义的变量。

语法：

```
NullInterpret
```

示例：

```
set NullInterpret=' ';
set NullInterpret =;
```

> 对于 Excel 中的空白值不会返回 NULL 值，但对于 CSV 文本文件的空白值将返回该值。

```
set NullInterpret ='';
```

对于 Excel 中的空白值会返回 NULL 值。

NullValue

如果使用 **NullAsValue** 语句，定义的符号可以替代 NULL 指定包含指定字符串的字段中的全部 **NullAsValue** 值。

语法：

```
NullValue
```

示例：

```
NullAsValue Field1, Field2;
set NullValue='<NULL>';
```

OtherSymbol

在 **LOAD/SELECT** 语句前定义需要用作“全部其他值”的符号。这是用户定义的变量。

语法：

```
OtherSymbol
```

示例：

```
set OtherSymbol='+';
LOAD * inline
[X, Y
a, a
b, b];
LOAD * inline
[X, Z
a, a
+, c];
```

字段值 Y='b' 现已通过其他字符链接到 Z='c'。

2.14 数字解释变量

数字解释变量是系统定义的变量，即这些变量是在创建新的应用程序时根据当前操作系统区域设置自动生成的。在 Qlik Sense Desktop 中，这取决于计算机操作系统的设置，在 Qlik Sense 中，它取决于已安装 Qlik Sense 的服务器的操作系统。

包括的变量位于 Qlik Sense 的新应用程序脚本顶部，并且可以在执行脚本时替换操作系统默认设置为某种数字格式设置。您可以随意删除、编辑或复制这些变量。

如果您想要为某些区域创建应用程序，最简单的方法可能是在操作系统中在包含所需区域设置的计算机上使用 Qlik Sense Desktop 创建应用程序。然后，应用程序包含该区域的适当地区设置，您可以将其移动到所选择的 Qlik Sense 服务器以便进一步开发。

数字解释变量概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

货币格式

MoneyDecimalSep

定义的小数位分隔符会替换操作系统(地区设置)的货币小数位符号。

```
MoneyDecimalSep
```

MoneyFormat

定义的符号会替换操作系统(地区设置)的货币符号。

```
MoneyFormat
```

MoneyThousandSep

定义的千位分隔符会替换操作系统(地区设置)的货币数字分组符号。

```
MoneyThousandSep
```

数字格式

DecimalSep

定义的小数位分隔符会替换操作系统(地区设置)的小数位符号。

```
DecimalSep
```

ThousandSep

定义的千位分隔符会替换操作系统(地区设置)的数字分组符号。

```
ThousandSep
```

NumericalAbbreviation

数字缩写设置将哪个缩写用于数字的量级前缀，例如将 M 用于百万 (10^6)，将 μ 用于微小 (10^{-6}).

```
NumericalAbbreviation
```

时间格式

DateFormat

此环境变量定义应用程序中用作默认值的日期格式。该格式用于解释和格式化日期。如果未定义变量，则在脚本运行时将获取操作系统区域设置的日期格式。

```
DateFormat
```

TimeFormat

定义的格式会替换操作系统(地区设置)的时间格式。

```
TimeFormat
```

TimestampFormat

定义的格式会替换操作系统(地区设置)的日期和时间格式。

```
TimestampFormat
```

MonthNames

定义的格式会替换操作系统(地区设置)的普通月名称惯例。

```
MonthNames
```

LongMonthNames

定义的格式会替换操作系统(地区设置)的长月名称惯例。

```
LongMonthNames
```

DayNames

定义的格式会替换操作系统(地区设置)的普通日名称惯例。

```
DayNames
```

LongDayNames

定义的格式会替换操作系统(地区设置)的长普通日名称惯例。

```
LongDayNames
```

FirstWeekDay

整数用于定义将哪一天用作一周的第一天。

```
FirstWeekDay
```

BrokenWeeks

该设置用于定义周是否已中断。

```
BrokenWeeks
```

ReferenceDay

此设置用于定义将一月的哪一天设置为定义第 1 周的参考日。

```
ReferenceDay
```

FirstMonthOfYear

该设置定义要用作某一年的第一个月的月份，可以用来定义使用每月偏移的财政年度，如从 4 月 1 日开始。

此设置目前未使用，但保留以供未来使用。

有效设置为 1(一月)到 12(十二月)。默认设置为 1。

语法：

```
FirstMonthOfYear
```

示例：

```
Set FirstMonthOfYear=4; //Sets the year to start in April
```

BrokenWeeks

该设置用于定义周是否已中断。

语法：

BrokenWeeks

默认情况下，Qlik Sense 函数使用连续的周。这意味着：

- 在某些年份中，第 1 周在 12 月开始，而在其他年份中，第 52 或 53 周延续到 1 月。
- 在 1 月中，第 1 周始终至少有 4 天。

替代方法是使用不连续的周：

- 第 52 或 53 周不延续到 1 月。
- 第 1 周在 1 月 1 日开始，因此在大部分情况下不是完整的一周。

可以使用以下值：

- 0（表示使用连续周）
- 1（表示使用不连续周）

示例：

```
Set BrokenWeeks=0; //(use unbroken weeks)

Set BrokenWeeks=1; //(use broken weeks)
```

DateFormat

此环境变量定义应用程序中用作默认值的日期格式。该格式用于解释和格式化日期。如果未定义变量，则在脚本运行时将获取操作系统区域设置的日期格式。

语法：

DateFormat

示例：

```
Set DateFormat='M/D/YY'; //(US format)

Set DateFormat='DD/MM/YY'; //(UK date format)

Set DateFormat='YYYY-MM-DD'; //(ISO date format)
```

DayNames

定义的格式会替换操作系统（地区设置）的普通日名称惯例。

语法：

DayNames

示例：

```
Set DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
```

DecimalSep

定义的小数位分隔符会替换操作系统(地区设置)的小数位符号。

语法：

DecimalSep

示例：

```
Set DecimalSep='.';

Set DecimalSep=',';
```

FirstWeekDay

整数用于定义将哪一天用作一周的第一天。

语法：

FirstWeekDay

默认情况下，Qlik Sense 系统变量定义 `FirstWeekDay=6`。这意味着周日是一周的第一天。

可以为 FirstWeekDay 设置值

值	日
0	星期一
1	星期二
2	星期三
3	星期四
4	星期五
5	星期六
6	星期日

LongDayNames

定义的格式会替换操作系统(地区设置)的长普通日名称惯例。

语法：

LongDayNames

示例：

```
Set LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
```

LongMonthNames

定义的格式会替换操作系统(地区设置)的长月名称惯例。

语法：

LongMonthNames

示例：

```
Set
LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;D
ecember';
```

MoneyDecimalSep

定义的小数位分隔符会替换操作系统(地区设置)的货币小数位符号。

语法：

MoneyDecimalSep

示例：

```
Set MoneyDecimalSep='.';
```

MoneyFormat

定义的符号会替换操作系统(地区设置)的货币符号。

语法：

MoneyFormat

示例：

```
Set MoneyFormat='$ #,##0.00; ($ #,##0.00)';
```

MoneyThousandSep

定义的千位分隔符会替换操作系统(地区设置)的货币数字分组符号。

语法：

MoneyThousandSep

示例：

```
Set MoneyThousandSep=',';
```

MonthNames

定义的格式会替换操作系统(地区设置)的普通月名称惯例。

语法:

```
MonthNames
```

示例:

```
Set MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
```

NumericalAbbreviation

数字缩写设置将哪个缩写用于数字的量级前缀,例如将 M 用于百万 (10^6),将 μ 用于微小 (10^{-6}).

语法:

```
NumericalAbbreviation
```

您将 NumericalAbbreviation 变量设置为包含一系列缩写定义对的字符串,这些缩写定义对以分号分隔。每个缩写定义对应包含量度(十进制指数)并且缩写以冒号分隔,例如 6:M 表示一百万。

默认设置为 '3:k;6:M;9:G;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y'。

示例:

该设置将把一千的前缀更改为 t 并将一亿的前缀更改为 B。这将可用于金融应用,在其中您可能需要 t$、M$ 和 B$ 等缩写。

```
Set NumericalAbbreviation='3:t;6:M;9:B;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y';
```

ReferenceDay

此设置用于定义将一月的哪一天设置为定义第 1 周的参考日。

语法:

```
ReferenceDay
```

默认情况下,Qlik Sense 函数使用 4 作为参考日。这意味着第 1 周必须包含 1 月 4 日,换句话说,第 1 周始终至少具有 1 月份的前 4 天。

以下值可用于设置不同参考日:

- 1(表示 1 月 1 日)
- 2(表示 2 月 1 日)
- 3(表示 3 月 1 日)
- 4(表示 4 月 1 日)
- 5(表示 5 月 1 日)
- 6(表示 6 月 1 日)
- 7(表示 7 月 1 日)

示例：

```
Set ReferenceDay=3; //(set January 3 as the reference day)
```

ThousandSep

定义的千位分隔符会替换操作系统(地区设置)的数字分组符号。

语法：

ThousandSep

示例：

```
Set ThousandSep=','; //(for example, seven billion must be specified as: 7,000,000,000)

Set ThousandSep=' ';
```

TimeFormat

定义的格式会替换操作系统(地区设置)的时间格式。

语法：

TimeFormat

示例：

```
Set TimeFormat='hh:mm:ss';
```

TimestampFormat

定义的格式会替换操作系统(地区设置)的日期和时间格式。

语法：

TimestampFormat

示例：

以下示例使用 *1983-12-14T13:15:30Z* 作为时间戳数据来显示不同 **SET TimestampFormat** 语句的结果。所用的日期格式为 **YYYYMMDD** 并且时间格式为 **h:mm:ss TT**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定，并且时间格式在 **SET TimeFormat** 语句中指定。

结果

示例	结果
`SET TimestampFormat='YYYYMMDD';`	19831214
`SET TimestampFormat='M/D/YY hh:mm:ss[.fff]';`	12/14/83 13:15:30
`SET TimestampFormat='DD/MM/YYYY hh:mm:ss[.fff]';`	14/12/1983 13:15:30
`SET TimestampFormat='DD/MM/YYYY hh:mm:ss[.fff] TT';`	14/12/1983 1:15:30 PM
`SET TimestampFormat='YYYY-MM-DD hh:mm:ss[.fff] TT';`	1983-12-14 01:15:30

示例:加载脚本

示例:加载脚本

在第一个加载脚本中使用了 *SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT'*。在第二个加载脚本中,时间戳格式更改为 *SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]'*。不同的结果示出 **SET TimeFormat** 语句如何用于不同的时间数据格式。

下面的表格示出用在所遵照的加载脚本中的数据集。表格的第二列示出数据集中每个时间戳的格式。前五个时间戳遵照 ISO 8601 规则但是第六个没有。

数据集

表格示出数据集中使用的时间数据以及每个时间戳的格式。

transaction_timestamp	time data format
2018-08-30	YYYY-MM-DD
20180830T193614.857	YYYYMMDDhhmmss.sss
20180830T193614.857+0200	YYYYMMDDhhmmss.sss±hhmm
2018-09-16T12:30-02:00	YYYY-MM-DDhh:mm±hh:mm
2018-09-16T13:15:30Z	YYYY-MM-DDhh:mmZ
9/30/18 19:36:14	M/D/YY hh:mm:ss

在**数据加载编辑器**中,创建新的部分,然后添加示例脚本并运行它。然后,至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

加载脚本

```
SET FirstWeekDay=0; SET BrokenWeeks=1; SET ReferenceDay=0; SET
DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun'; SET
LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday'; SET
DateFormat='YYYYMMDD'; SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT';   Transactions: Load
*, Timestamp(transaction_timestamp, 'YYYY-MM-DD hh:mm:ss[.fff]') as LogTimestamp ;   Load *
Inline [ transaction_id, transaction_timestamp, transaction_amount, transaction_quantity,
discount, customer_id, size, color_code 3750, 2018-08-30, 12423.56, 23, 0,2038593, L, Red
3751, 20180830T193614.857, 5356.31, 6, 0.1, 203521, m, orange 3752, 20180830T193614.857+0200,
15.75, 1, 0.22, 5646471, S, blue 3753, 2018-09-16T12:30-02:00, 1251, 7, 0, 3036491, l, Black
3754, 2018-09-16T13:15:30Z, 21484.21, 1356, 75, 049681, xs, Red 3755, 9/30/18 19:36:14, -
59.18, 2, 0.333333333333333, 2038593, M, Blue ];
```

结果

Qlik Sense 表格示出在加载脚本中使用的 TimestampFormat 解释变量的结果。数据集中最后的时间戳不返回正确的日期。

transaction_id	transaction_timestamp	LogTimeStamp
3750	2018-08-30	2018-08-30 00:00:00

transaction_id	transaction_timestamp	LogTimeStamp
3751	20180830T193614.857	2018-08-30 19:36:14
3752	20180830T193614.857+0200	2018-08-30 17:36:14
3753	2018-09-16T12:30-02:00	2018-09-16 14:30:00
3754	2018-09-16T13:15:30Z	2018-09-16 13:15:30
3755	9/30/18 19:36:14	-

下个加载脚本使用相同的数据集。然而，它使用 *SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]'* 来匹配第六个时间戳的非 ISO 8601 格式。

在**数据加载编辑器**中，将之前的示例脚本替换为下面的一个并运行它。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

加载脚本

```
SET FirstWeekDay=0; SET BrokenWeeks=1; SET ReferenceDay=0; SET
DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun'; SET
LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday'; SET
DateFormat='YYYYMMDD'; SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]';   Transactions: Load
*, Timestamp(transaction_timestamp, 'YYYY-MM-DD hh:mm:ss[.fff]') as LogTimestamp ;   Load *
Inline [ transaction_id, transaction_timestamp, transaction_amount, transaction_quantity,
discount, customer_id, size, color_code 3750, 2018-08-30, 12423.56, 23, 0,2038593, L, Red
3751, 20180830T193614.857, 5356.31, 6, 0.1, 203521, m, orange 3752, 20180830T193614.857+0200,
15.75, 1, 0.22, 5646471, S, blue 3753, 2018-09-16T12:30-02:00, 1251, 7, 0, 3036491, l, Black
3754, 2018-09-16T13:15:30Z, 21484.21, 1356, 75, 049681, xs, Red 3755, 9/30/18 19:36:14, -
59.18, 2, 0.333333333333333, 2038593, M, Blue ];
```

结果

Qlik Sense 表格示出在加载脚本中使用的 TimestampFormat 解释变量的结果。

transaction_id	transaction_timestamp	LogTimeStamp
3750	2018-08-30	2018-08-30 00:00:00
3751	20180830T193614.857	2018-08-30 19:36:14
3752	20180830T193614.857+0200	2018-08-30 17:36:14
3753	2018-09-16T12:30-02:00	2018-09-16 14:30:00
3754	2018-09-16T13:15:30Z	2018-09-16 13:15:30
3755	9/30/18 19:36:14	2018-09-16 19:36:14

2.15 Direct Discovery 变量

Direct Discovery 系统变量

DirectCacheSeconds

您可以为 Direct Discovery 可视化查询结果设置缓存限制。在达到此时间限制后，Qlik Sense 会在执行新的 Direct Discovery 查询时清除缓存。Qlik Sense 将查询选择项的源数据并将为指定的时间限制再次创建缓存。对于各选择项组合，将独立缓存其结果。即为每个选择项独立刷新缓存，对于一个选择项，仅对所选字段刷新缓存，对于另一个选择项，对其相关字段刷新缓存。如果第二个选择项包含在第一个选择项中刷新的字段，则在尚未达到缓存限制的情况下不会在缓存中再次更新这些字段。

Direct Discovery 缓存不适用于**表格**可视化。对于表格选择项，每次都会查询数据源。

限值设置必须使用秒为单位。默认缓存限制是 1800 秒(30 分) 。

用于 **DirectCacheSeconds** 的值是在执行 **DIRECT QUERY** 语句时设置的值。在运行时不能更改此值。

示例：

```
SET DirectCacheSeconds=1800;
```

DirectConnectionMax

您可使用连接池功能进行数据库异步、并行调用。设置连接池功能的加载脚本语法如下所示：

```
SET DirectConnectionMax=10;
```

数字设置指定更新表格时 Direct Discovery 代码应使用的最大数据库连接数量。默认设置为 1。

> *此变量应谨慎使用。如果将其设置为大于 1 的数，会导致在连接到 Microsoft SQL Server 时出现问题。*

DirectUnicodeStrings

Direct Discovery 可以通过使用一些数据库(尤其是 SQL Server) 所要求的扩展字符串文字 (N'<扩展字符串>') 的 SQL 标准格式来支持扩展 Unicode 数据的选择。带有脚本变量 **DirectUnicodeStrings** 的 Direct Discovery 允许使用这种语法。

将该变量设置为“真”将允许在字符串文字之前使用 ANSI 标宽字符标记“N”。并非所有数据库都支持此标准。默认设置为“假”。

DirectDistinctSupport

如果在 Qlik Sense 对象中选择 **DIMENSION** 字段值，则会为源数据库生成查询。当查询要求分组时，Direct Discovery 会使用 **DISTINCT** 关键字仅选择唯一的值。但是，某些数据库要求使用 **GROUP BY** 关键字。将 **DirectDistinctSupport** 设置为 'false' 会在唯一值的查询中生成 **GROUP BY**，而非 **DISTINCT**。

```
SET DirectDistinctSupport='false';
```

如果将 DirectDistinctSupport 设置为“真”，则使用 **DISTINCT**。否则，默认行为是使用 **DISTINCT**。

DirectEnableSubquery
在高基数多表格情形中，可能会在 SQL 查询中生成子查询，而不是生成很大的 IN 子句。这可以通过将 **DirectEnableSubquery** 设置为 'true' 来激活。默认值为 'false'。

*启用 **DirectEnableSubquery**，无法加载不是处于 Direct Discovery 模式下的表格。*

```
SET DirectEnableSubquery='true';
```

Teradata 查询分级变量

Teradata 查询分级是一个函数，可让企业应用程序与基础 Teradata 数据库一起协作，以便提供更好的会计、确定优先顺序和工作量管理。使用查询分级，可以围绕查询限制元数据，例如用户凭据。

两个变量都可用，都是发送到数据库的评估字符串。

SQLSessionPrefix
在创建数据库连接时，发送此字符串。

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & Chr(39) & 'Who=' & OSuser() & ';' & Chr(39) & '
FOR SESSION;';
```

例如，如果 **OSuser()** 返回 *WA\sbt*，将针对 `SET QUERY_BAND = 'Who=WA\sbt;' FOR SESSION;` 评估此结果，此字符串会在创建连接时发送到数据库。

SQLQueryPrefix
每一次查询都发送此字符串。

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & Chr(39) & 'Who=' & OSuser() & ';' & Chr(39) & '
FOR TRANSACTION;';
```

Direct Discovery 字符变量

DirectFieldColumnDelimiter
您可以在 **Direct Query** 数据库语句中将使用的字符设置为字段分隔符，字段分隔符必须是非逗号字符。在 **SET** 语句中，必须对指定字符使用单引号。

```
SET DirectFieldColumnDelimiter= '|'
```

DirectStringQuoteChar
可以指定要在生成的查询中用于引用字符串的字符。默认值是单引号。在 **SET** 语句中，必须对指定字符使用单引号。

```
SET DirectStringQuoteChar= '"';
```

DirectIdentifierQuoteStyle
可以指定在生成的查询中使用的非 ANSI 引用的标识符。此时，唯一可用的非ANSI 引用是 GoogleBQ。默认值为 ANSI。可以使用大写、小写和混合大小写格式 (ANSI, ansi, Ansi)。

```
SET DirectIdentifierQuoteStyle="GoogleBQ";
```

例如，在以下 **SELECT** 语句中使用 ANSI 引用：

```
SELECT [Quarter] FROM [qvTest].[sales] GROUP BY [Quarter]
```

当 **DirectIdentifierQuoteStyle** 设置为 "GoogleBQ" 时，**SELECT** 语句将使用如下引用：

```
SELECT [Quarter] FROM [qvTest.sales] GROUP BY [Quarter]
```

DirectIdentifierQuoteChar

可以指定要在生成的查询中控制标识符引用的字符。此字符可以设置为一个字符(例如双引号)或两个字符(例如方括号对)。默认值是双引号。

```
SET DirectIdentifierQuoteChar='[]';
SET DirectIdentifierQuoteChar='``';
SET DirectIdentifierQuoteChar=' ';
SET DirectIdentifierQuoteChar='""';
```

DirectTableBoxListThreshold

在**表格**可视化中使用 Direct Discovery 字段时，可设置阈值来限制显示的行数。默认阈值为 1000 个记录。默认阈值设置可以更改，只需在加载脚本中设置 **DirectTableBoxListThreshold** 变量。例如：

```
SET DirectTableBoxListThreshold=5000;
```

阈值设置仅适用于包含 Direct Discovery 字段的**表格**可视化。仅包含内存中字段的**表格**可视化不受 **DirectTableBoxListThreshold** 设置限制。

在选择项的记录少于阈值限制之前，不会在**表格**可视化中显示任何字段。

Direct Discovery 数字解释变量

DirectMoneyDecimalSep

定义的小数位分隔符会替代使用 Direct Discovery 加载数据生成的 SQL 语句中的货币小数位符号。此字符必须与 **DirectMoneyFormat** 中使用的字符一致。

默认值为 '.'

示例：

```
Set DirectMoneyDecimalSep='.';
```

DirectMoneyFormat

定义的符号会替代使用 Direct Discovery 加载数据生成的 SQL 语句中的货币格式。不应包含千分位分隔符的货币符号。

默认值为 '#.0000'

示例：

```
Set DirectMoneyFormat='#.0000';
```

DirectTimeFormat

定义的时间格式会替代使用 Direct Discovery 加载数据生成的 SQL 语句中的时间格式。

示例：

```
Set DirectTimeFormat='hh:mm:ss';
```

DirectDateFormat

定义的日期格式会替代使用 Direct Discovery 加载数据生成的 SQL 语句中的日期格式。

示例：

```
Set DirectDateFormat='MM/DD/YYYY';
```

DirectTimeStampFormat

定义的格式会替代使用 Direct Discovery 加载数据生成的 SQL 语句中的日期和时间格式。

示例：

```
Set DirectTimestampFormat='M/D/YY hh:mm:ss[.fff]';
```

2.16 错误变量

所有错误变量的值在脚本执行之后依然保留。第一个变量 ErrorMode 由用户输入，最后三个变量是 Qlik Sense 的输出(包括脚本中错误的信息)。

错误变量概述

概述后将进一步描述每个变量。也可以单击语法中的变量名称即时访问有关该特定变量的更多信息。

有关这些变量的详细信息，请参阅 Qlik Sense 在线帮助。

ErrorMode

此错误变量可确定在脚本执行期间遇到错误时 Qlik Sense 将采取什么操作。

```
ErrorMode
```

ScriptError

此错误变量用于返回上次执行的脚本语句的错误代码。

```
ScriptError
```

ScriptErrorCount

此错误变量用于返回在当前脚本执行期间引起错误的语句总数。此变量在脚本开始执行时总是重置为 0。

```
ScriptErrorCount
```

ScriptErrorList

此错误变量包含上次脚本执行期间发生的所有脚本错误的串联列表。每个错误均以换行方式隔开。

```
ScriptErrorList
```

ErrorMode

此错误变量可确定在脚本执行期间遇到错误时 Qlik Sense 将采取什么操作。

语法：

```
ErrorMode
```

参数：

参数

参数	说明
ErrorMode=1	默认设置。脚本执行会暂停，并且会提示用户进行操作(非批量模式)。
ErrorMode =0	Qlik Sense 只需忽略故障，并继续在下一个脚本语句上执行脚本。
ErrorMode =2	一旦出现错误，Qlik Sense 会立即触发“脚本执行故障...”错误信息，但不会提示用户预先进行操作。

示例：

```
set ErrorMode=0;
```

ScriptError

此错误变量用于返回上次执行的脚本语句的错误代码。

语法：

```
ScriptError
```

每次成功执行脚本语句之后，此变量将重置为 0。如果发生错误，则其会设置为 Qlik Sense 内部错误代码。错误代码为带有数值和文本组件的双重值。以下错误代码存在：

脚本错误代码

错误代码	说明
0	无错误。双值文本为空。
1	一般错误。
2	语法错误。
3	一般 ODBC 错误。
4	一般 OLE DB 错误。
5	一般自定义数据库错误。
6	一般 XML 错误。
7	一般 HTML 错误。
8	文件未找到。

错误代码	说明
9	数据库未找到。
10	未找到表格。
11	字段未找到。
12	文件格式错误。
16	语义错误。

示例：

```
set ErrorMode=0;

LOAD * from abc.qvf;

if ScriptError=8 then

exit script;

//no file;

end if
```

ScriptErrorCount

此错误变量用于返回在当前脚本执行期间引起错误的语句总数。此变量在脚本开始执行时总是重置为 0。

语法：

```
ScriptErrorCount
```

ScriptErrorList

此错误变量包含上次脚本执行期间发生的所有脚本错误的串联列表。每个错误均以换行方式隔开。

语法：

```
ScriptErrorList
```

2 脚本表达式

表达式可用于 **LOAD** 语句和 **SELECT** 语句。此处所述的语法和函数适用于 **LOAD** 语句，不适用于 **SELECT** 语句，因为后者由 ODBC 驱动程序（而非 Qlik Sense）进行解释。然而，大多数 ODBC 驱动程序往往能够解释以下函数。

表达式包含在语法中组合使用的函数、字段和运算符。

Qlik Sense 脚本中的全部表达式会返回数字及/或字符串，不论哪个适当。逻辑函数和运算符对于 False 返回 0，对于 True 返回 -1。数字和字符串的转换是隐式的。逻辑运算符和函数将 0 解释为 False，将所有其他结果解释为 True。

表达式的一般语法为：

一般语法

表达式	字段	运算符
expression ::= (constant	constant	|
expression ::= (constant	fieldref	|
expression ::= (constant	operator1 expression	|
expression ::= (constant	expression operator2 expression	|
expression ::= (constant	function	|
expression ::= (constant	(expression)	)

其中：

- **constant** 是由单引号括起来的字符串（文本，日期或时间）或数字。写入的常数没有千分位分隔符，但使用小数点作为小数位分隔符。
- **fieldref** 是加载表格的字段名。
- **operator1** 是一元运算符（作用于一个表达式，位于右边）。
- **operator2** 是二元运算符（作用于两个表达式，每边一个）。
- **function ::= functionname(parameters)**
- **parameters ::= expression { , expression }**

参数的数字和类型不是任意的。它们取决于所使用的函数。

表达式和函数还可自由嵌套，并且只要表达式返回可解释的值，Qlik Sense 就不会显示任何错误信息。

3 图表表达式

图表(可视化)表达式是函数、字段和数学运算符 (+ * / =) 及其他度量的组合。表达式用于处理应用程序中的数据,以便生成可以在可视化中看到的结果。在度量中,不限制使用表达式。您可以创建更有活力更强大的可视化,只需使用标题、副标题、脚注和维度的表达式。

这表示(例如)可视化标题不是静态文本,而是可以使用表达式获取的内容,其结果将根据做出的选择改变。

有关脚本函数和图表函数的详细参考,请参阅 脚本语法和图表函数。

3.1 定义聚合范围

通常,结合两个因子可以确定用于定义表达式中聚合值的记录。当在可视化中使用时,这些因子为:

- 维度值(图表表达式中的聚合)
- 选择项

总之,这些因子可定义聚合的范围。您可能会遇到希望计算忽略选择项、维度或同时忽略这二者的情况。在图表函数中,为此可以使用 TOTAL 限定符、集合分析或两者的组合。

聚合:方法和描述

方法	说明
TOTAL 限定符	在聚合函数内使用合计限定符忽略维度值。 将对所有可能的字段值执行聚合。 **TOTAL** 限定符后可能紧跟着一系列由尖括号括起来的一个或多个字段名。这些字段名应该是图表维度变量的子集。此时,计算会忽略所有图表维度变量,但会计算已列出的变量,即列出的维度字段内字段值的各组合均会返回一个值。此外,当前并非为图表内维度的字段也可能会包括在列表之中。这对于组维度可能极为有用,其中未固定维度字段。在组中列出全部变量会导致函数在钻取级变化时生效。
集合分析	在聚合内使用集合分析将覆盖选择项。将对在维度之间拆分的所有值执行聚合。
TOTAL 限定符和集合分析	在聚合内使用 **TOTAL** 限定符将覆盖选择项并忽略维度。

ALL 限定符	在聚合内使用 **ALL** 限定符将忽略选择项和维度。通过 {1} 设置分析语句和 **TOTAL** 限定符可以获得同等效果:
	`=sum(All Sales)`
	`=sum({1} Total Sales)`

示例：TOTAL 限定符

以下示例显示了如何使用 TOTAL 计算相对共享。假定已选择 Q2，使用 TOTAL 计算全部值的总和，同时忽略维度。

示例：TOTAL 限定符

Year	Quarter	Sum(Amount)	Sum(TOTAL Amount)	Sum(Amount)/Sum(TOTAL Amount)
		3000	3000	100%
2012	Q2	1700	3000	56,7%
2013	Q2	1300	3000	43,3%

要将数字显示为百分比，请针对要显示为百分比值的度量，在属性面板中的 ***Number formatting*** *下，选择* ***Number****，然后从* ***Formatting*** *中选择* ***Simple*** *和其中一种百分比格式。*

示例：集合分析

以下示例显示了如何在做出任何选择之前使用集合分析比较不同数据集。假定已选择 Q2，使用集合定义 {1} 的集合分析计算全部值的总和，同时忽略所有选择项，但按维度拆分。

示例：集合分析

Year	Quarter	Sum(Amount)	Sum({1} Amount)	Sum(Amount)/Sum({1} Amount)
		3000	10800	27,8%
2012	Q1	0	1100	0%
2012	Q3	0	1400	0%
2012	Q4	0	1800	0%
2012	Q2	1700	1700	100%
2013	Q1	0	1000	0%
2013	Q3	0	1100	0%
2013	Q4	0	1400	0%
2013	Q2	1300	1300	100%

示例：TOTAL 限定符和集合分析

以下示例显示了如何在所有维度之间做出任何选择之前组合集合分析和 TOTAL 限定符来比较不同数据集。假定已选择 Q2，使用集合定义 {1} 的集合分析和 TOTAL 限定符计算全部值的总和，同时忽略所有选择项，并忽略维度。

示例:TOTAL 限定符和集合分析

Year	Quarter	Sum (Amount)	Sum({1} TOTAL Amount)	Sum(Amount)/Sum({1} TOTAL Amount)
		3000	10800	27,8%
2012	Q2	1700	10800	15,7%
2013	Q2	1300	10800	12%

示例中所使用的数据:

```
AggregationScope: LOAD * inline [ Year Quarter Amount 2012 Q1 1100 2012 Q2 1700 2012 Q3 1400
2012 Q4 1800 2013 Q1 1000 2013 Q2 1300 2013 Q3 1100 2013 Q4 1400] (delimiter is ' ');
```

3.2 集合分析

在应用程序中进行选择时,将在数据中定义记录的子集。聚合函数,诸如 Sum()、Max()、Min()、Avg() 和 Count() 都基于该子集计算。

换句话说,您的选择定义了聚合的范围;它定义了进行计算的记录集。

集合分析提供了一种定义范围的方法,该范围不同于当前选择所定义的记录集。该新范围也可被视为一种替代选择。

如果要将当前选择与特定值(例如去年的值或全球市场份额)进行比较,此选项非常有用。

集合表达式

集合表达式在聚合函数中使用,并用花括号括起来。例如:

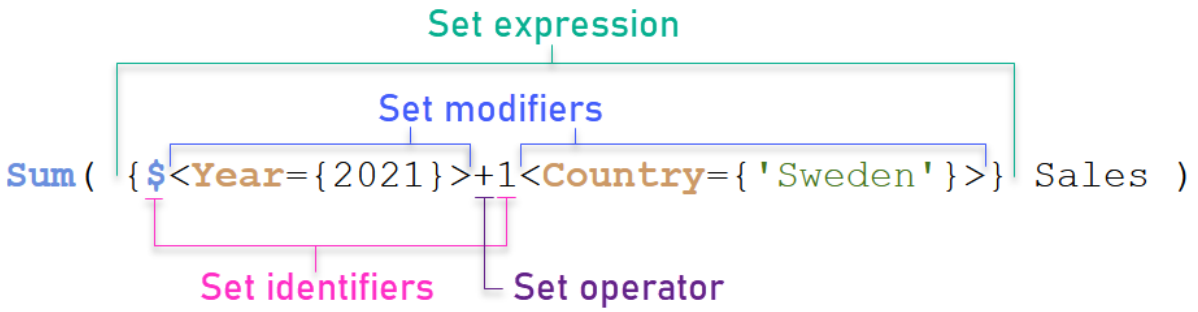
```
Sum( {$<Year={2021}>} Sales )
```

集合表达式包含以下元件的组合:

- **标识符**。集合标识符表示在别处定义的选择。它还表示数据中的一组特定记录。它可以是当前选择、书签选择或备用状态选择。简单的集合表达式包含一个单一的标识符(如美元符号 {$}),这意味着当前选择项中的所有记录。
  示例:$、1、BookMark1、State2
- **运算符**。集合运算符可用于创建不同集合标识符之间的并集、差异或交点。通过这种方式,您可以创建由集合标识符定义的选择的子集或超集。
  示例:+、-、*、/
- **修饰符**。可以将集合修饰符添加到集合标识符以更改其选择。修饰符也可以单独使用,然后修改默认标识符。修饰符必须用尖括号 <…> 括起来。
  示例:<Year={2020}>、<Supplier={ACME}>

这些元素被组合成集合表达式。

集合表达式中的元素



例如，以上集合表达式是从聚合 Sum(Sales) 生成的。

第一个操作数返回当前选择的年份 2021 的销售额，由 $ 集合标识符和包含年份 2021 选择的修饰符指示。第二个操作数返回 Sweden 的 Sales，并忽略由 1 集合标识符指示的当前选择。

最后，表达式返回一个集合，该集合由属于两个集合操作数中任何一个的记录组成，如 + 集合运算符所示。

示例

以下主题中提供了组合上述集合表达式元素的示例：

自然集

通常，集合表达式表示数据模型中的一组记录和定义此数据子集的选择。在这种情况下，该集合称为自然集合。

集合标识符(带或不带集合修饰符)始终表示自然集合。

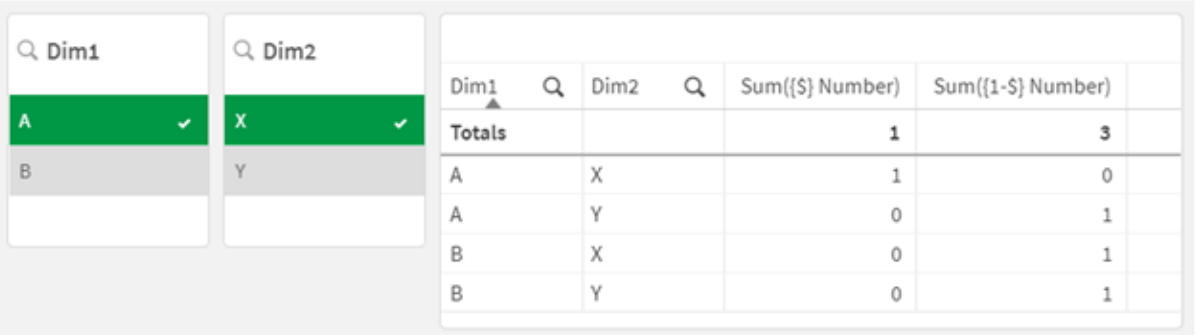
然而，使用集合运算符的集合表达式也表示记录的子集，但通常仍然不能使用字段值的选择来描述。这样的表达式是非自然的集合。

例如，{1-$} 给定的集合不能总是由选择定义。因此，它不是一个自然的集合。这可以通过加载以下数据、将其添加到表中，然后使用筛选窗格进行选择来显示。

```
Load * Inline [Dim1, Dim2, Number A, X, 1 A, Y, 1 B, X, 1 B, Y, 1];
```

通过对 Dim1 和 Dim2 进行选择，可以获得下表所示的视图。

具有自然集合和非自然集合的表



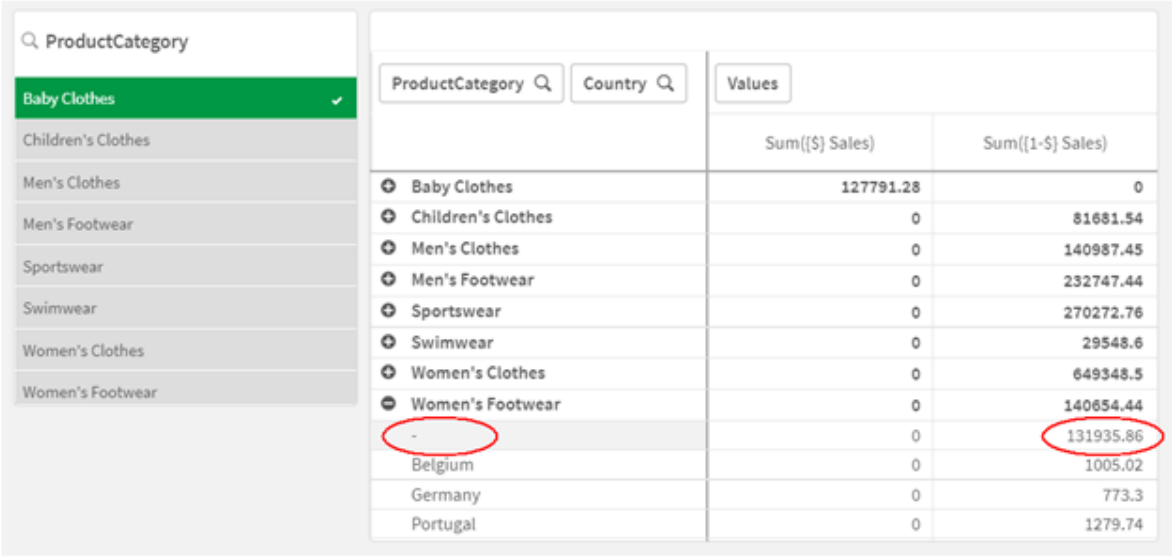
Dim1	Dim2	Sum({$} Number)	Sum({1-$} Number)
Totals		1	3
A	X	1	0
A	Y	0	1
B	X	0	1
B	Y	0	1

第一个度量中的集合表达式使用自然集合:它对应于所做的选择 {$}。

第二度量是不同的。它使用 {1-$}。不可能做出与此集合对应的选择,因此它是非自然集合。

这种区别有许多后果:

- 集合修饰符只能应用于集合标识符。它们不能应用于任何集合表达式。例如,不可能使用以下集合表达式:
  { (BM01 * BM02) <Field={x,y}> }
  在这里,正常(圆形)括号表示应在应用"设置"修饰符之前计算 BM01 和 BM02 之间的交点。原因是没有可以修改的元素集。
- 不能在 P() 和 E() 元素函数中使用非自然集。这些函数返回一个元素集,但无法从非自然集推断元素集。
- 如果数据模型有多个表,则使用非自然集的度量值不能始终归因于正确的维度值。例如,在下表中,一些排除在外的销售数字归因于正确的 Country,而另一些将 NULL 作为 Country。
  具有非自然集的图表



  分配是否正确取决于数据模型。在这种情况下,如果号码属于被选择排除在外的国家,则无法分配该号码。

集合标识符

集合标识符表示数据中的一组记录;所有数据或数据的子集。它是由选择定义的记录集。它可以是当前选择、所有数据(无选择)、书签选择或备用状态选择。

在示例 Sum({$<Year = {2009}>} Sales) 中,标识符为美元符号:$。这表示当前选择。它还表示所有可能的记录。然后,可以通过集合表达式的修饰符部分更改此集合:添加 Year 中的选择 2009。

在更复杂的集合表达式中,两个标识符可与运算符一起使用,以形成两个记录集的并集、差集或交集。

下表显示了一些常用的修饰符。

带共同标识符的示例

标识符	说明
1	表示应用程序中所有记录的完整集合，而不考虑选择的任何选择项。
$	表示默认状态下的当前选择项的记录。因此，集合表达式 {$} 通常与不陈述集合表达式的意义等同。
$1	表示默认状态下的上一个选择。$2 表示上一个“唯一选择”，依此类推。
$_1	表示下一个（向前）选择。$_2 表示下一个“唯一选择”，以此类推。
BM01	您可以使用任何书签 ID 或书签名称。
AltState	您可以使用状态名称引用备用状态。
AltState::BM01	书签包含所有状态的选择，您可以通过限定书签名称来引用特定书签。

下表显示了一些不同的标识符示例。

带不同标识符的示例

示例	结果
Sum ({1} Sales)	返回应用程序的总销售额，忽略选择项而不是维度。
Sum ({$} Sales)	返回当前选择项的销售额，也就是说效果与 Sum(Sales) 相同。
Sum ({$1} Sales)	返回上一个选择项的销售额。
Sum ({BM01} Sales)	返回书签名为 BM01 的销售额。

集合运算符

集合运算符用于包含、排除或交汇数据集。所有运算符都将集合用作操作数，并返回集协作为结果。

可以在两种不同的情况下使用集合运算符：

- 对集合标识符执行集合操作，表示数据中的记录集合。
- 对元素集、字段值或集合修饰符内部执行集合操作。

下表显示了可用于集合表达式的运算符。

运算符

运算符	说明
+	并集运算符。此二元运算返回两个集合操作数中所有记录或元素构成的集合。
-	异或运算符。此二元运算返回由属于第一个集合操作数但不属于另一个集合操作数的记录或元素构成的集合。如用于一元运算，则结果是补集。

运算符	说明
*	交集运算符。此二元运算返回两个集合操作数中所有记录或元素构成的集合。
/	对称差集 (XOR)。此二元运算返回两个集合操作数中所有记录或元素构成的集合。

下表显示了一些运算符示例。

运算符示例

示例	结果
`Sum ({1-$} Sales)`	用于返回除当前选择项以外的所有销售额。
`Sum ({$*BM01} Sales)`	用于返回选择项和书签 BM01 之间交集的销售额。
`Sum ({-($+BM01)} Sales)`	用于返回除选择项和书签 BM01 以外的销售额。
`Sum ({$<Year= {2009}>+1<Country= {'Sweden'}>} Sales)`	用于返回与当前选择项相关联的 2009 年的销售额总和，并添加所有年度中与国家 Sweden 相关联的整个数据集。
`Sum ({$<Country={"S*"}+ {"*land"}>} Sales)`	返回以 S 开头或以 land 结尾的国家/地区的销售额。

集合修饰符

集合表达式用于定义计算范围。集合表达式的中心部分是指定选择的集合修饰符。这用于修改用户选择或集合标识符中的选择，结果定义了新的计算范围。

集合修饰符由一个或多个字段名组成，每个字段名后面都有一个应在该字段上进行的选择。修饰符由尖括号括起：< >

例如：

- `Sum ( {$<Year = {2015}>} Sales )`
- `Count ( {1<Country = {Germany}>} distinct OrderID )`
- `Sum ( {$<Year = {2015}, Country = {Germany}>} Sales )`

元素集

可以使用以下内容定义元素集：

- 值列表
- 搜索
- 另一字段的引用
- 集合函数

如果省略元素集定义，则集合修饰符将清除此字段中的任何选择。例如：

```
Sum( {$<Year = >} Sales )
```

示例:基于元素集的修饰符集合的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入,以创建以下图表表达式示例。

```
MyTable: Load * Inline [ Country, Year, Sales Argentina, 2014, 66295.03 Argentina, 2015, 140037.89 Austria, 2014, 54166.09 Austria, 2015, 182739.87 Belgium, 2014, 182766.87 Belgium, 2015, 178042.33 Brazil, 2014, 174492.67 Brazil, 2015, 2104.22 Canada, 2014, 101801.33 Canada, 2015, 40288.25 Denmark, 2014, 45273.25 Denmark, 2015, 106938.41 Finland, 2014, 107565.55 Finland, 2015, 30583.44 France, 2014, 115644.26 France, 2015, 30696.98 Germany, 2014, 8775.18 Germany, 2015, 77185.68 ];
```

图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

表格 - 基于元素集的集合修饰符

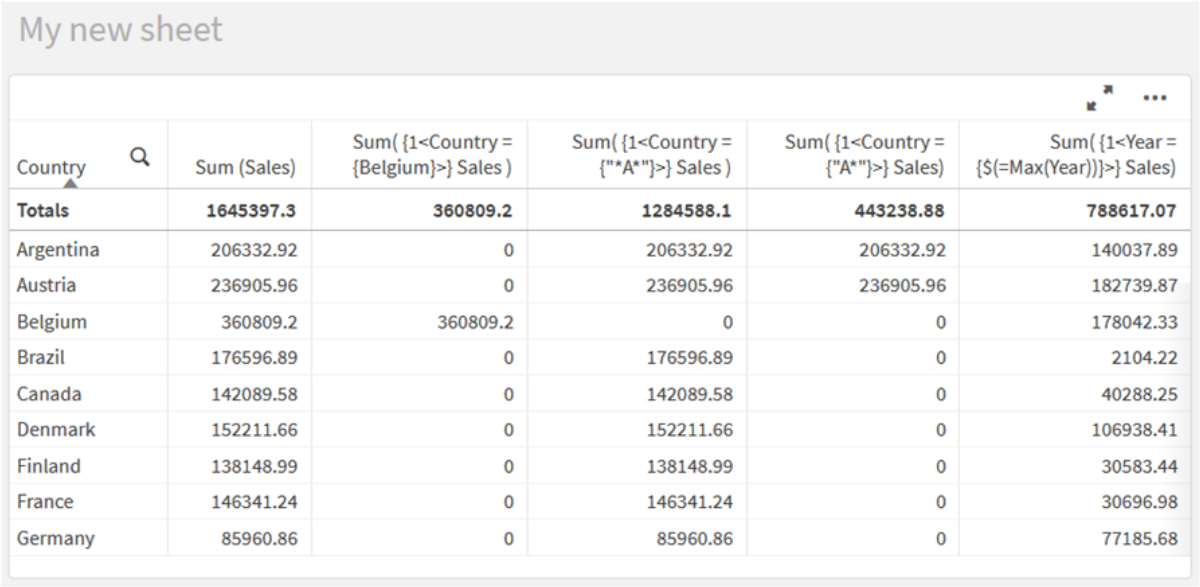
国家	Sum(Sales)	Sum ({1<Country= {Belgium}>} Sales)	Sum ({1<Country= {"*A*"}>} Sales)	Sum ({1<Country= {"A*"}>} Sales)	Sum({1<Year= {$(=Max (Year))}>} Sales)
总计	1645397.3	360809.2	1284588.1	443238.88	788617.07
阿根廷	206332.92	0	206332.92	206332.92	140037.89
奥地利	236905.96	0	236905.96	236905.96	182739.87
比利时	360809.2	360809.2	0	0	178042.33
巴西	176596.89	0	176596.89	0	2104.22
加拿大	142089.58	0	142089.58	0	40288.25
丹麦	152211.66	0	152211.66	0	106938.41
芬兰	138148.99	0	138148.99	0	30583.44
法国	146341.24	0	146341.24	0	30696.98
Germany	85960.86	0	85960.86	0	77185.68

解释

- 维度:
  - `Country`
- 度量:
  - `Sum(Sales)`
    没有集合表达式的总和 `Sales`。

- Sum({1<Country={Belgium}>}Sales)
  选择 Belgium，然后选择相应的 Sales。
- Sum({1<Country={"*A*"}>}Sales)
  选择具有 A 的所有国家，然后将对应的 Sales 求总和。
- Sum({1<Country={"A*"}>}Sales)
  选择以 A 开头的所有国家，然后将对应的 Sales 求总和。
- Sum({1<Year={$(=Max(Year))}>}Sales)
  计算 Max(Year)，其为 2015,，然后将对应的 Sales 求总和。

设置基于元素集的修饰符


My new sheet

Country	Sum (Sales)	Sum({1<Country = {Belgium}>} Sales)	Sum({1<Country = {"*A*"}>} Sales)	Sum({1<Country = {"A*"}>} Sales)	Sum({1<Year = {$(=Max(Year))}>} Sales)
Totals	**1645397.3**	**360809.2**	**1284588.1**	**443238.88**	**788617.07**
Argentina	206332.92	0	206332.92	206332.92	140037.89
Austria	236905.96	0	236905.96	236905.96	182739.87
Belgium	360809.2	360809.2	0	0	178042.33
Brazil	176596.89	0	176596.89	0	2104.22
Canada	142089.58	0	142089.58	0	40288.25
Denmark	152211.66	0	152211.66	0	106938.41
Finland	138148.99	0	138148.99	0	30583.44
France	146341.24	0	146341.24	0	30696.98
Germany	85960.86	0	85960.86	0	77185.68

列出的值

元素集的最常见的示例是基于包含在波浪括号中的字段值的列表的集合表达式。例如：

- {$<Country = {Canada, Germany, Singapore}>}
- {$<Year = {2015, 2016}>}

内部花括号定义元素集。各个值之间用逗号分隔。

引号和大小写区分

如果值包含空格或特殊字符，则需将值加引号。单引号将与单个字段值进行文本、区分大小写的匹配。双引号表示与一个或多个字段值不区分大小写的匹配。例如：

- <Country = {'New Zealand'}>
  仅匹配 New Zealand。
- <Country = {"New Zealand"}>
  匹配 New Zealand、NEW ZEALAND 以及 new zealand.

日期必须用引号括起来，并使用相关字段的日期格式。例如：

- <ISO_Date = {'2021-12-31'}>
- <US_Date = {'12/31/2021'}>
- <UK_Date = {'31/12/2021'}>

双引号可以用方括号或重音符代替。

搜索

也可以通过搜索创建元素集。例如:

- <Country = {"C*"}>
- <Ingredient = {"*garlic*"}>
- <Year = {">2015"}>
- <Date = {">12/31/2015"}>

通配符可用于文本搜索:星号 (*) 表示任意数量的字符,问号 (?) 表示单个字符。关系运算符可用于定义数值搜索。

搜索时应始终使用双引号。搜索要区分大小写。

美元符号扩展

如果要在元素集中使用计算,则需要美元符号扩展。例如,如果只想查看最后一年,请使用:

```
<Year = {$(=Max(Year))}>
```

其他字段中所选择的值

修饰符可基于另一个字段的所选值。例如:

```
<OrderDate = DeliveryDate>
```

此修饰符将获得 DeliveryDate 的所选值,并将这些值作为选择项应用于 OrderDate。如果字段包括很多不同特殊值(数百个),则该操作是 CPU 密集型的,应避免此操作。

元素集函数

元素集也可以基于 P() 集合函数(可能值)和 E()(排除值)。

例如,如果要选择已销售产品 'Cap' 的国家/地区,可以使用:

```
<Country = P({1<Product={Cap}>} Country)>
```

类似地,如果要选择产品 Cap 尚未销售的国家,可以使用:

```
<Country = E({1<Product={Cap}>} Country)>
```

集合修饰符和高级搜索

可以通过使用集合修饰符进行搜索来创建元素集合。

例如:

- <Country = {"C*"}>
- <Year = {">2015"}>
- <Ingredient = {"*garlic*"}>

搜索应始终用双引号、方括号或严肃的重音符号括起来。您可以使用混合了文字字符串(单引号)和搜索(双引号)的列表。例如:

```
<Product = {'Nut', "*Bolt", Washer}>
```

文本搜索

通配符和其他符号可用于文本搜索:

- 星号 (*) 表示任意数量的字符。
- 问号 (?) 表示单个字符。
- 扬抑重音 (^) 将标记单词的开头。

例如:

- <Country = {"C*", "*land"}>
  匹配以 C 开头或以 land 技术的所有国家。
- <Country = {"*^z*"}>
  这将匹配有以 z 开头的单次的所有国家,例如 New Zealand。

数字搜索

您可以使用以下关系运算符进行数字搜索:>、>=、<、<=

数值搜索始终以这些运算符之一开始。例如:

- <Year = {">2015"}>
  匹配 2016 年及以后年份。
- <Date = {">=1/1/2015<1/1/2016"}>
  匹配 2015 年期间的所有日期。请注意用于描述两个日期之间的时间范围的语法。日期格式需要与相关字段的日期格式匹配。

表达式搜索

可以使用表达式搜索进行更高级的搜索。然后为搜索字段中的每个字段值计算聚合。将选择搜索表达式返回 true 的所有值。

表达式搜索始终以等号开头:=

例如:

```
<Customer = {"=Sum(Sales)>1000"}>
```

这将返回销售额大于1000的所有客户。Sum(Sales) 根据当前选择计算。这意味着,如果您在另一个字段(如 Product 字段)中进行了选择,则您将仅获得满足所选产品销售条件的客户。

如果希望条件独立于选择,则需要在搜索字符串中使用集合分析。例如:

```
<Customer = {"=Sum({1} Sales)>1000"}>
```

等号后面的表达式将被解释为布尔值。这意味着，如果它的计算结果为其他值，任何非零数字都将被解释为 true，而零和字符串则被解释为 false。

引号

当搜索字符串包含空格或特殊字符时，请使用引号。单引号将与单个字段值进行文本、区分大小写的匹配。双引号表示不区分大小写的搜索可能匹配多个字段值。

例如：

- `<Country = {'New Zealand'}>`
  仅匹配 `New Zealand`。
- `<Country = {"New Zealand"}>`
  匹配 `New Zealand`、`NEW ZEALAND` 以及 `new zealand`

双引号可以用方括号或重音符代替。

> *在之前的 Qlik Sense 版本中，在单引号和双引号之间不存在区别，并且将所有引用的字符串作为搜索处理。要保持向后兼容性，使用较旧版本 Qlik Sense 创建的应用程序将继续和在之前版本中那样工作。使用 Qlik Sense November 2017 或更高版本创建的应用程序将存在两种类型引号之间的差异。*

示例：带有搜索的集合修饰符的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
MyTable: Load  Year(Date) as Year, Date#(Date,'YYYY-MM-DD') as ISO_Date,  Date(Date#
(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,  Country, Product, Amount  Inline [Date, Country,
Product, Amount 2018-02-20, Canada, Washer, 6 2018-07-08, Germany, Anchor bolt, 10 2018-07-14,
Germany, Anchor bolt, 3 2018-08-31, France, Nut, 2 2018-09-02, Czech Republic, Bolt, 1 2019-
02-11, Czech Republic, Bolt, 3 2019-07-31, Czech Republic, Washer, 6 2020-03-13, France,
Anchor bolt, 1 2020-07-12, Canada, Anchor bolt, 8 2020-09-16, France, Washer, 1];
```

示例 1：带有文本搜索的图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

表格 - 带文本搜索的集合修饰符

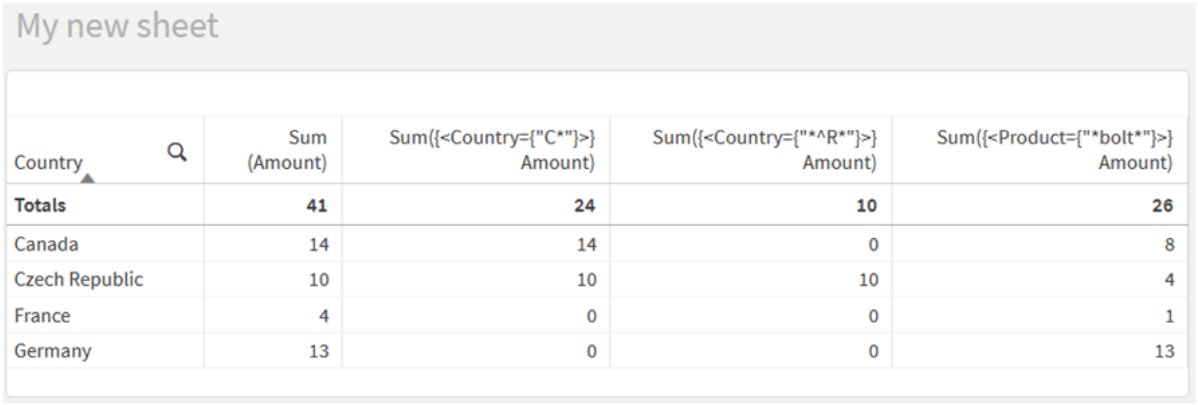
国家	Sum (Amount)	Sum({<Country= {"C*"}>} Amount)	Sum({<Country= {"*^R*"}>} Amount)	Sum({<Product= {"*bolt*"}>} Amount)
总计	**41**	**24**	**10**	**26**
加拿大	14	14	0	8
捷克共和	10	10	10	4

国家	Sum (Amount)	Sum({<Country= {"C*"}>} Amount)	Sum({<Country= {"*^R*"}>} Amount)	Sum({<Product= {"*bolt*"}>} Amount)
总计	**41**	**24**	**10**	**26**
国				
法国	4	0	0	1
Germany	13	0	0	13

解释

- 维度：
  - `Country`
- 度量：
  - `Sum(Amount)`
    没有集合表达式的总和 `Amount`。
  - `Sum({<Country={"C*"}>}Amount)`
    总和 `Amount`，针对以 `C` 开头的所有国家，例如 `Canada` 以及 `Czech Republic`。
  - `Sum({<Country={"*^R*"}>}Amount)`
    总和 `Amount`，针对以 `R` 开头的所有国家，例如 `Czech Republic`。
  - `Sum({<Product={"*bolt*"}>}Amount)`
    总和 `Amount`，针对包含字符串 `bolt` 的所有产品，例如 `Bolt` 以及 `Anchor bolt`。

带文本高级搜索的集合修饰符



示例 2: 带有数字搜索的图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

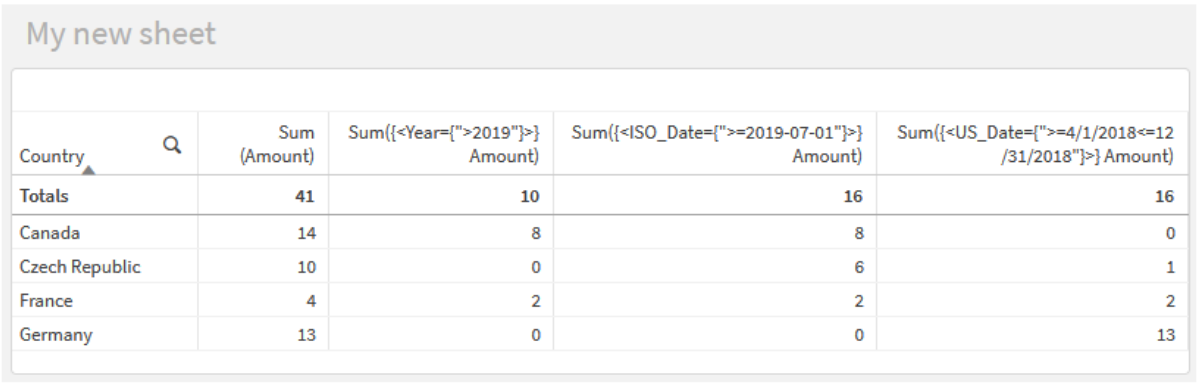
表 - 带有数字搜索的集合表达式

国家	Sum (Amount)	Sum({<Year={">2019"}>} Amount)	Sum({<ISO_Date={">=2019-07-01"}>} Amount)	Sum({<US_Date={">=4/1/2018<=12/31/2018"}>} Amount)
总计	**41**	**10**	**16**	**16**
加拿大	14	8	8	0
捷克共和国	10	0	6	1
法国	4	2	2	2
Germany	13	0	0	13

解释

- 维度：
  - `Country`
- 度量：
  - `Sum(Amount)`
    没有集合表达式的总和 `Amount`。
  - `Sum({<Year={">2019"}>}Amount)`
    总和 `Amount`，针对 2019 之后的所有年份。
  - `Sum({<ISO_Date={">=2019-07-01"}>}Amount)`
    总和 `Amount`，针对 2019-07-01 或之后的所有日期。搜索中日期的格式必须与字段的格式匹配。
  - `Sum({<US_Date={">=4/1/2018<=12/31/2018"}>}Amount)`
    总和 `Amount`，适用于从 4/1/2018 到 12/31/2018 的所有日期，包括开始和结束日期。搜索中日期的格式必须与字段的格式匹配。

带有数字搜索的集合表达式



示例 3: 带有表达式搜索的图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

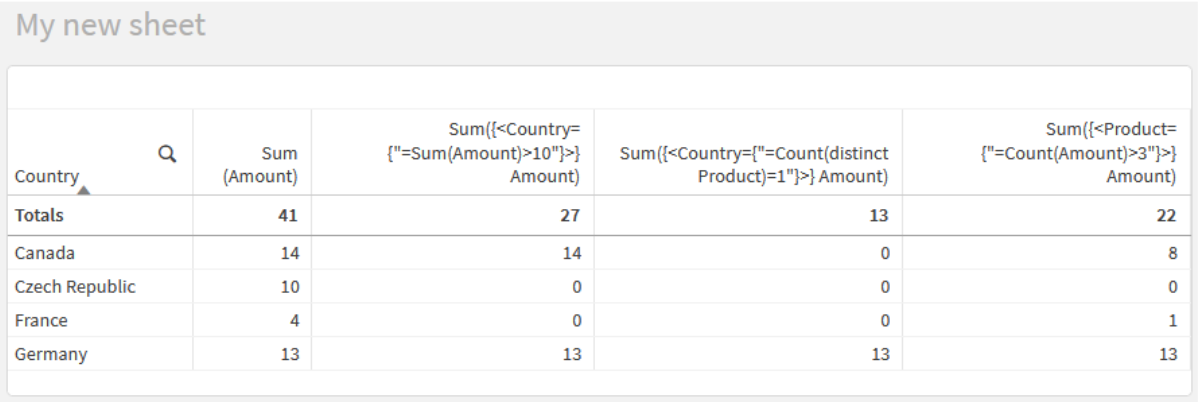
Table - Set modifiers with expression searches

Country	Sum (Amount)	Sum({<Country={"=Sum (Amount)>10"}>} Amount)	Sum({<Country={"=Count(distinct Product)=1"}>} Amount)	Sum({<Product={"=Count (Amount)>3"}>} Amount)
Totals	**41**	**27**	**13**	**22**
Canada	14	14	0	8
Czech Republic	10	0	0	0
France	4	0	0	1
Germany	13	13	13	13

解释

- 维度：
  - `Country`
- 度量：
  - `Sum(Amount)`
    没有集合表达式的总和 `Amount`。
  - `Sum({<Country={"=Sum(Amount)>10"}>}Amount)`
    总和 `Amount`，针对 `Amount` 的聚合总和大于 10 的所有国家。
  - `Sum({<Country={"=Count(distinct Product)=1"}>}Amount)`
    总和 `Amount`，针对准确地与一个不同的产品关联的所有国家。
  - `Sum({<Product={"=Count(Amount)>3"}>}Amount)`
    总和 `Amount`，针对数据中有三个以上交易的所有国家。

带有表达式搜索的集合表达式



集合修饰符和美元符号扩展

美元符号扩展是在解析和计算表达式之前计算的结构。然后将结果注入到表达式中，而不是注入 `$(...)`。然后使用美元符号扩展的结果计算表达式。

表达式编辑器显示美元符号扩展预览，以便您可以验证美元符号扩展的计算结果。

表达式编辑器中美元符号扩展预览

如果要在元素集中使用计算，请使用美元符号扩展。

例如，如果只想查看上一个可能的年份，可以使用以下构造：

```
<Year = {$(=Max(Year))}>
```

Max(Year) 首先被计算，结果将注入表达式中，而不是 $(…)。

货币符号扩展后的结果将是如下表达式：

```
<Year = {2021}>
```

美元符号扩展中的表达式是基于当前选择计算的。这意味着，如果在另一个字段中进行了选择，则表达式的结果将受到影响。

如果希望计算独立于选择，则在美元符号扩展中使用集合分析。例如：

```
<Year = {$(=Max({1} Year))}>
```

字符串

当您希望美元符号扩展生成字符串时，将应用正常的报价规则。例如：

```
<Country = {'$(=FirstSortedValue(Country,Date)'}>
```

货币符号扩展后的结果将是如下表达式：

```
<Country = {'New Zealand'}>
```

如果不使用引号，将出现语法错误。

数字

如果希望美元符号扩展生成数字，请确保扩展的格式与字段的格式相同。这意味着您有时需要将表达式包含到格式化函数中。

例如：

```
<Amount = {$(=Num(Max(Amount), '###0.00'))}>
```

货币符号扩展后的结果将是如下表达式：

```
<Amount = {12362.00}>
```

使用散列强制扩展始终使用小数点，并且不使用千位分隔符。例如：

```
<Amount = {$(#=Max(Amount))}>
```

日期

如果希望美元符号扩展生成日期，请确保扩展设定正确格式。这意味着您有时需要将表达式包含到格式化函数中。

例如：

```
<Date = {'$(=Date(Max(Date)))'}>
```

货币符号扩展后的结果将是如下表达式：

```
<Date = {'12/31/2015'}>
```

与字符串一样，您需要使用正确的引号。

一个常见的用例是，您希望将计算限制在最后一个月(或一年)。然后可以将数字搜索与 AddMonths() 函数结合使用。

例如：

```
<Date = {">=$(=AddMonths(Today(),-1))"}>
```

货币符号扩展后的结果将是如下表达式：

```
<Date = {">=9/31/2021"}>
```

这将选出上个月发生的所有事件。

示例:带美元符号扩展的集合修饰符的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
Let vToday = Today(); MyTable: Load  Year(Date) as Year, Date#(Date,'YYYY-MM-DD') as ISO_Date,
 Date(Date#(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,  Country, Product, Amount  Inline
[Date, Country, Product, Amount 2018-02-20, Canada, Washer, 6 2018-07-08, Germany, Anchor
bolt, 10 2018-07-14, Germany, Anchor bolt, 3 2018-08-31, France, Nut, 2 2018-09-02, Czech
Republic, Bolt, 1 2019-02-11, Czech Republic, Bolt, 3 2019-07-31, Czech Republic, Washer, 6
2020-03-13, France, Anchor bolt, 1 2020-07-12, Canada, Anchor bolt, 8 2021-10-15, France,
Washer, 1];
```

带美元符号扩展的图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

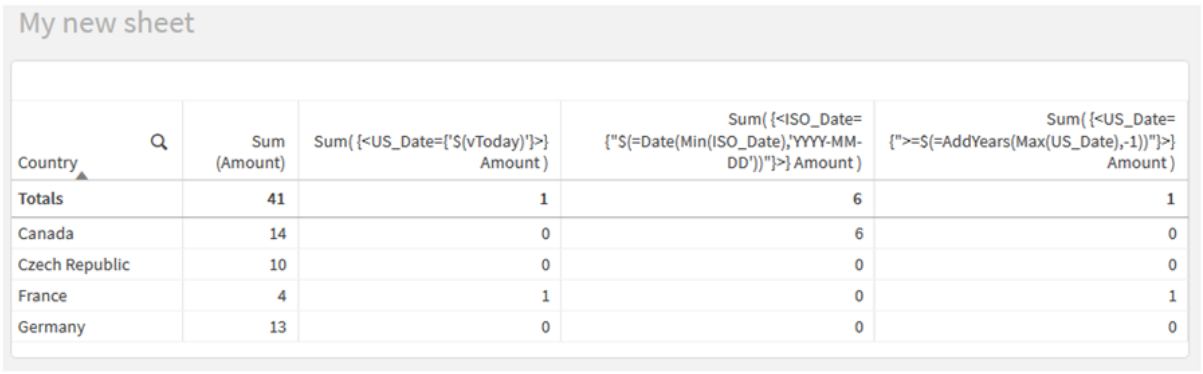
表 - 集合修饰符和美元符号扩展

国家	Sum (Amount)	Sum({<US_ Date= {'$(vToday)'}>} Amount)	Sum({<ISO_Date= {"$(=Date(Min(ISO_ Date),'YYYY-MM-DD'))"}>} Amount)	Sum({<US_Date= {">=$(=AddYears(Max (US_Date),-1))"}>} Amount)
总计	**41**	**1**	**6**	**1**
加拿大	14	0	6	0
捷克共和国	10	0	0	0
法国	4	1	0	1
Germany	13	0	0	0

解释

- 维度：
  - `Country`
- 度量：
  - `Sum(Amount)`
    和 没有集合表达式的 `Amount`。
  - `Sum({<US_Date={'$(vToday)'}>}Amount)`
    总和 `Amount`，针对 `US_Date` 和变量 `vToday` 中的一样的所有记录。
  - `Sum({<ISO_Date={"$(=Date(Min(ISO_Date),'YYYY-MM-DD'))"}>}Amount)`
    总和 `Amount`，针对 `ISO_Date` 和第一个（最小）可能的 `ISO_Date` 相同的所有记录。`Date()` 函数用于确保日期的格式与字段的格式匹配。
  - `Sum({<US_Date={">=$(=AddYears(Max(US_Date),-1))"}>}Amount)`
    总和 `Amount`，针对 `US_Date` 在最新（最大）可能 `US_Date` 的之前的年份的日期之后或之时的所有记录。`AddYears()` 函数将返回日期，其格式由变量 `DateFormat` 指定，并且这需要匹配字段 `US_Date` 的格式。

集合修饰符和美元符号扩展



集合修饰符和集合运算符

集合运算符用于包含、排除或交汇不同的图元集。它们结合了不同的方法来定义元素集。

运算符与用于集合标识符的运算符相同。

运算符

运算符	说明
+	并集运算符。此二元运算返回两个集合操作数中所有记录或元素构成的集合。
-	异或运算符。此二元运算返回由属于第一个集合操作数但不属于另一个集合操作数的记录或元素构成的集合。如用于一元运算，则结果是补集。
*	交集运算符。此二元运算返回两个集合操作数中所有记录或元素构成的集合。
/	对称差集 (XOR)。此二元运算返回两个集合操作数中所有记录或元素构成的集合。

例如，以下两个修饰符定义相同的字段值集：

- <Year = {1997, "20*"}>
- <Year = {1997} + {"20*"}>

这两个表达式都选择 1997 和以 20 开头的年份。换句话说，是这两个条件的结合。

集合运算符还允许更复杂的定义。例如：

```
<Year = {1997, "20*"} - {2000}>
```

此表达式将选择与上述年份相同的年份，但不包括年份 2000。

。

示例：带有集合运算符的集合修饰符的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
MyTable: Load  Year(Date) as Year, Date#(Date,'YYYY-MM-DD') as ISO_Date,  Date(Date#
(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,  Country, Product, Amount  Inline [Date, Country,
Product, Amount 2018-02-20, Canada, Washer, 6 2018-07-08, Germany, Anchor bolt, 10 2018-07-14,
Germany, Anchor bolt, 3 2018-08-31, France, Nut, 2 2018-09-02, Czech Republic, Bolt, 1 2019-
02-11, Czech Republic, Bolt, 3 2019-07-31, Czech Republic, Washer, 6 2020-03-13, France,
Anchor bolt, 1 2020-07-12, Canada, Anchor bolt, 8 2020-09-16, France, Washer, 1];
```

图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

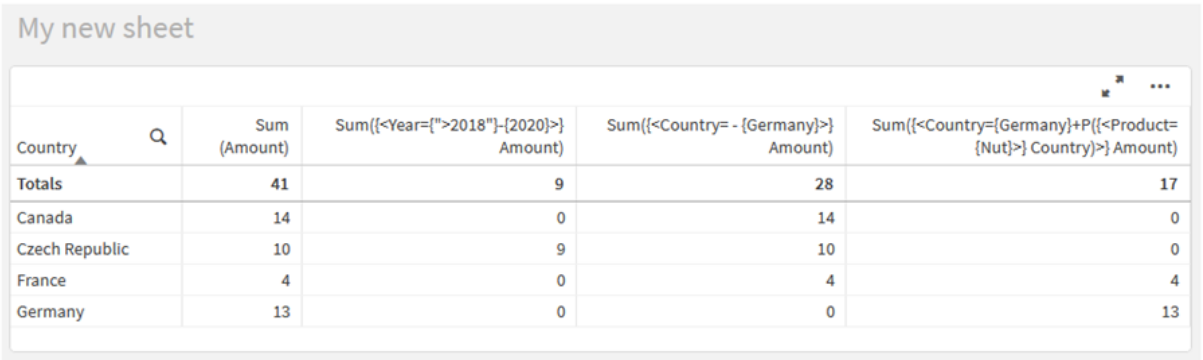
表格 - 集合修饰符和集合运算符

国家	Sum (Amount)	Sum({<Year={">2018"}-{2020}>} Amount)	Sum ({<Country=-{Germany}>} Amount)	Sum({<Country={Germany}+P({<Product={Nut}>}Country)>} Amount)
总计	**41**	**9**	**28**	**17**
加拿大	14	0	14	0
捷克共和国	10	9	10	0
法国	4	0	4	4
Germany	13	0	0	13

解释

- 维度：
  - `Country`
- 度量：
  - `Sum(Amount)`
    没有集合表达式的总和 `Amount`。
  - `Sum({<Year={">2018"}-{2020}>}Amount)`
    总和 `Amount`，针对 2018 之后的所有年份，不包括 2020。
  - `Sum({<Country=-{Germany}>}Amount)`
    总和 `Amount`，针对 `Germany` 之外的所有国家/地区。请注意一元排除运算符。
  - `Sum({<Country={Germany}+P({<Product={Nut}>}Country)>}Amount)`
    总和 `Amount`，针对 `Germany` 以及与产品 `Nut` 关联的所有国家。

集合修饰符和集合运算符



带隐式集合运算符的集合修饰符

在集合修饰符中写入选择的标准方法是使用等号。例如:

```
Year = {">2015"}
```

集合修改器中等号右侧的表达式称为元素集合。它定义了一组不同的字段值,换句话说就是一个选择。

此表示法定义了新选择项,忽略了字段中的当前选择项。因此,如果集合标识符包含此字段中的选择,则旧的选择将替换为元素集合中的选择。

如果要基于字段中的当前选择进行选择,则需要使用不同的表达式

例如,如果您希望尊重旧的选择,并添加年份在 2015 年之后的要求,您可以编写以下内容:

```
Year = Year * {">2015"}
```

星号是定义交集的集合运算符,因此您将获得 Year 中的当前选择与 2015 后的附加要求之间的交集。另一种编写方法如下:

```
Year *= {">2015"}
```

也就是说,赋值操作符 (*=) 隐式定义了一个交集。

类似地,可以使用以下公式定义隐式并集、排除和对称差集:+=、-=、/=

示例:带有隐式集合运算符的集合修饰符的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入,以创建以下图表表达式示例。

```
 MyTable: Load  Year(Date) as Year, Date#(Date,'YYYY-MM-DD') as ISO_Date,  Date(Date#
(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,  Country, Product, Amount  Inline [Date, Country,
Product, Amount 2018-02-20, Canada, Washer, 6 2018-07-08, Germany, Anchor bolt, 10 2018-07-14,
Germany, Anchor bolt, 3 2018-08-31, France, Nut, 2 2018-09-02, Czech Republic, Bolt, 1 2019-
02-11, Czech Republic, Bolt, 3 2019-07-31, Czech Republic, Washer, 6 2020-03-13, France,
Anchor bolt, 1 2020-07-12, Canada, Anchor bolt, 8 2020-09-16, France, Washer, 1];
```

带有隐式集合运算符的图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

从国家列表中选择 Canada 和 Czech Republic。

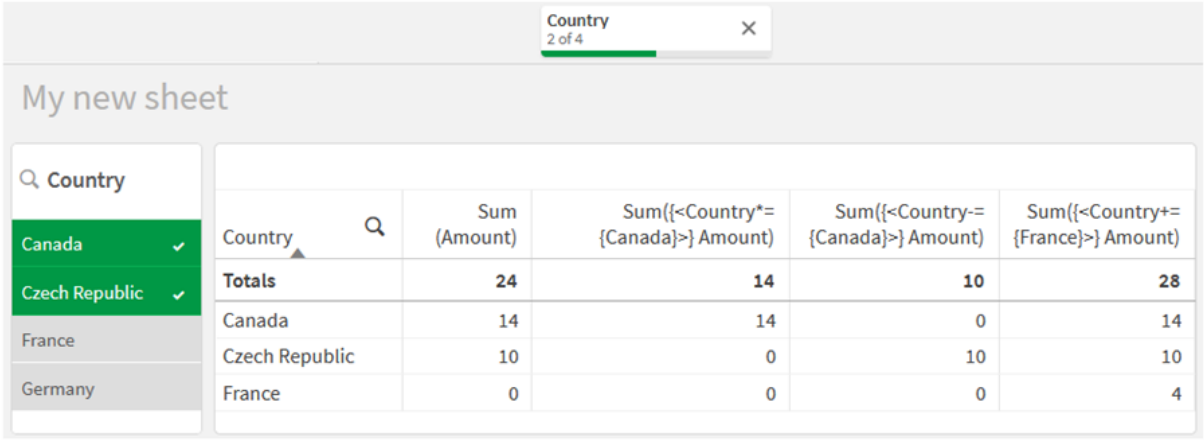
表格 - 带有隐式集合运算符的图表表达式

国家	Sum (Amount)	Sum({<Country*= {Canada}>} Amount)	Sum({<Country-= {Canada}>} Amount)	Sum({<Country+= {France}>} Amount)
总计	**24**	**14**	**10**	**28**
加拿大	14	14	0	14
捷克共和国	10	0	10	10
法国	0	0	0	4

解释

- 维度：
  - Country
- 度量：
  - Sum(Amount)
    总和 Amount，针对当前选择项。注意，仅 Canada 和 Czech Republic 具有非零值。
  - Sum({<Country*={Canada}>}Amount)
    总和 Amount，针对当前选择项，与 Country 为 Canada 的要求相交。如果 Canada 不是用户选择的一部分，则集合表达式将返回一个空集，并且该列的所有行都将为 0。
  - Sum({<Country-={Canada}>}Amount)
    总和 Amount，针对当前选择项，但首先从 Country 选择项排除 Canada。如果 Canada 不是用户选择的一部分，则集合表达式不会更改任何数字。
  - Sum({<Country+={France}>}Amount)
    总和 Amount，针对当前选择项，但首先将 France 添加至 Country 选择项 。如果 France 不是用户选择的一部分，则集合表达式不会更改任何数字。

带隐式集合运算符的集合修饰符



使用集合函数的集合修饰符

有时需要使用嵌套的集合定义定义一组字段值。例如，您可能希望选择购买了特定产品的所有客户，而不选择该产品。

在这种情况下，请使用元素集函数 P() 和 E()。它们分别返回字段的可能值和排除值的元素集。在括号内，可以指定相关字段和定义范围的集合表达式。例如：

```
P({1<Year = {2021}>} Customer)
```

这将返回 2021 年有交易的客户集。然后可以在集合修饰符中使用此选项。例如：

```
Sum({<Customer = P({1<Year = {2021}>} Customer)>} Amount)
```

此集合表达式将选择这些客户，但不会将选择限制在 2021 年。

这些函数不能用于其他表达式。

此外，在元素集函数中只能使用自然集合。即，可通过简单选择项定义的记录集合。

例如，通过 {1-$} 指定的集合无法始终通过选择项进行定义，因此该集合不是自然集合。在非自然集合中使用这些函数将返回意外结果。

示例：使用集合函数的集合修饰符的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
MyTable: Load  Year(Date) as Year, Date#(Date,'YYYY-MM-DD') as ISO_Date,  Date(Date#
(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,  Country, Product, Amount  Inline [Date, Country,
Product, Amount 2018-02-20, Canada, Washer, 6 2018-07-08, Germany, Anchor bolt, 10 2018-07-14,
Germany, Anchor bolt, 3 2018-08-31, France, Nut, 2 2018-09-02, Czech Republic, Bolt, 1 2019-
02-11, Czech Republic, Bolt, 3 2019-07-31, Czech Republic, Washer, 6 2020-03-13, France,
Anchor bolt, 1 2020-07-12, Canada, Anchor bolt, 8 2020-09-16, France, Washer, 1];
```

图表表达式

使用以下图表表达式在 Qlik Sense 工作表中创建表。

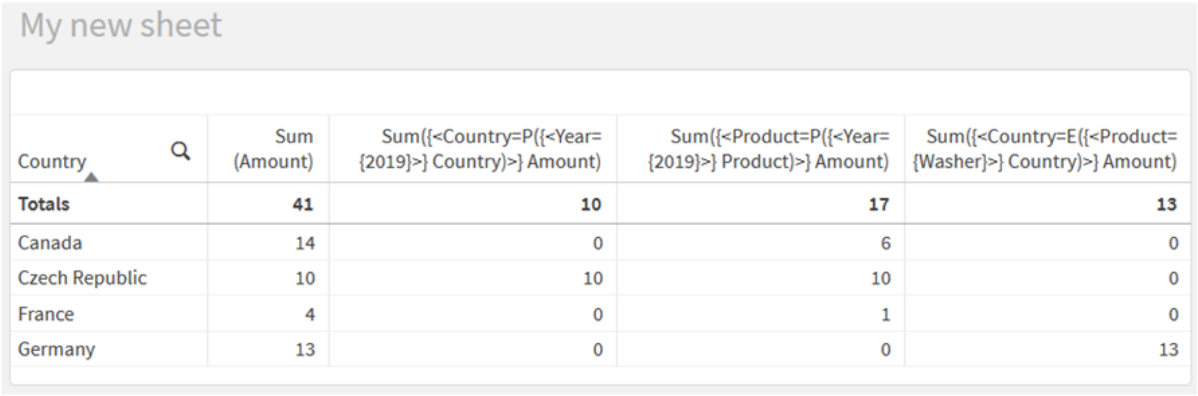
表 - 使用集合函数的集合修饰符

国家	Sum (Amount)	Sum({<Country=P ({<Year= {2019}>}Country)>} Amount)	Sum({<Product=P ({<Year= {2019}>}Product)>} Amount)	Sum({<Country=E ({<Product= {Washer}>}Country)>} Amount)
总计	**41**	**10**	**17**	**13**
加拿大	14	0	6	0
捷克共和国	10	10	10	0
法国	4	0	1	0
Germany	13	0	0	13

解释

- 维度：
  - `Country`
- 度量：
  - `Sum(Amount)`
    没有集合表达式的总和 `Amount`。
  - `Sum({<Country=P({<Year={2019}>} Country)>} Amount)`
    总和 `Amount`，针对与年份 `2019` 相关的国家。但是，它不会将计算限制为 `2019`。
  - `Sum({<Product=P({<Year={2019}>} Product)>} Amount)`
    总和 `Amount`，针对与年份 `2019` 相关的产品。但是，它不会将计算限制为 `2019`。
  - `Sum({<Country=E({<Product={Washer}>} Country)>} Amount)`
    总和 `Amount`，针对与产品 `Washer` 不相关的国家。

使用集合函数的集合修饰符



教程 - 创建集合表达式

可以构建集表达式以支持数据分析。在这种情况下，分析通常被称为集合分析。集合分析提供了一种定义范围的方法，该范围不同于当前选择所定义的记录集。

您将学到的内容

本教程提供用于使用集合修饰符、标识符和运算符构建集合表达式的数据和图表表达式。

谁应当完成该教程

本教程面向熟悉使用脚本编辑器和图表表达式的应用程序开发人员。

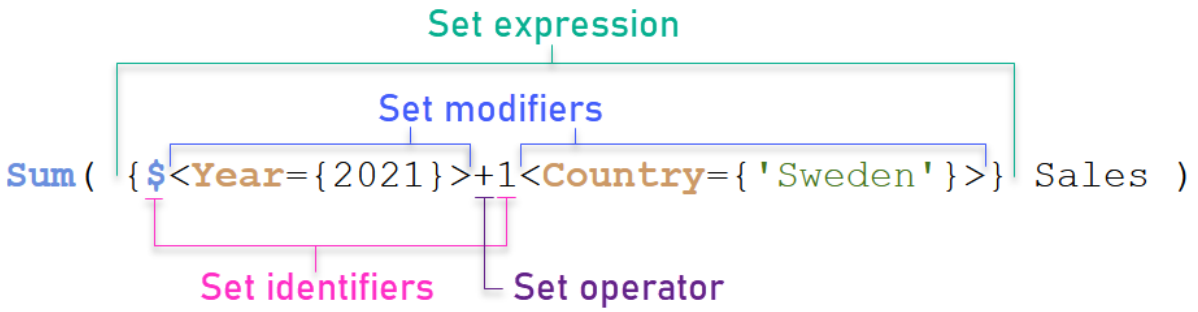
在开始之前需要做的工作

Qlik Sense Enterprise Professional 访问权限分配，允许您加载数据和创建应用程序。

集合表达式中的元素

集合表达式包含在聚合函数中，例如 Sum()、Max()、Min()、Avg() 或 Count()。集合表达式由称为元素的构建块构造而成。这些元素是集合修饰符、标识符和运算符。

集合表达式中的元素



例如，以上集合表达式是从聚合 Sum(Sales) 生成的。集合表达式包含在外部的花括号中：{ }

表达式中的第一个操作数为：$<Year={2021}>

此操作数返回当前选择的年份 2021 的销售额。修饰符 <Year={2021}>，包含 2021 年的选择。$ 集合标识符表示集合表达式基于当前选择。

表达式中的第二个操作数为：1<Country={'Sweden'}>

该操作数对 Sweden 返回 Sales。修饰符 <Country={'Sweden'}>，包含国家 Sweden 的选择。1 集合标识符表示将忽略在应用程序中所做的选择。

最后，+集合运算符指示表达式返回一个集合，该集合由属于两个集合操作数中任何一个的记录组成。

创建集合表达式教程

完成以下步骤以创建本教程中所示的集合表达式。

创建新应用程序并加载数据

执行以下操作：

1. 创建新应用程序。
2. 单击**脚本编辑器**。或者，单击导航栏中的**准备 > 数据加载编辑器**。
3. 在**数据加载编辑器**中创建新部分。
4. 复制以下数据并粘贴到新部分中：*集合表达式教程数据 (page 186)*
5. 单击**加载数据**。数据作为内联加载加载。

使用修饰符创建集合表达式

集合修饰符由一个或多个字段名组成，每个字段名后面都有一个应在该字段上进行的选择。修饰符由尖括号括起。例如，在此集合表达式中：

```
Sum ( {<Year = {2015}>} Sales )
```

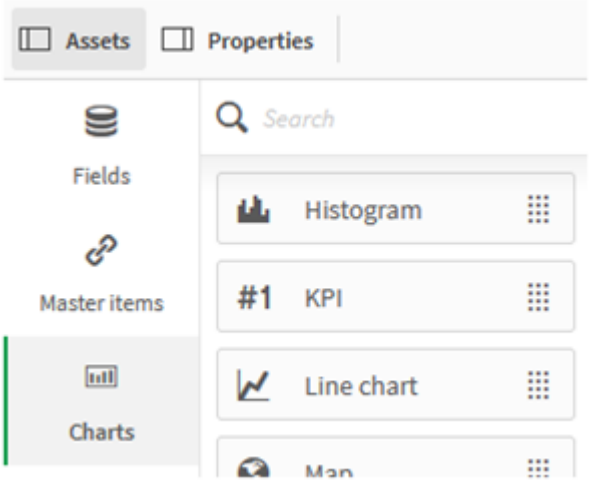
修饰符为：

```
<Year = {2015}>
```

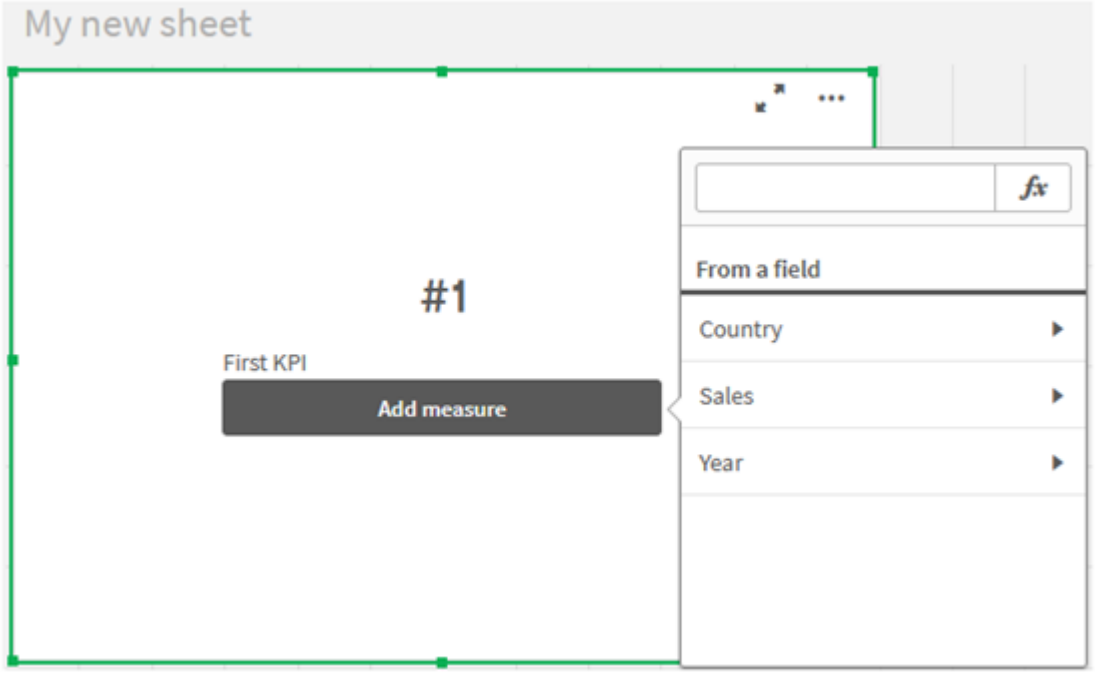
此修饰符指定仅选择 2015 年的数据。包含修饰符的花括号表示集合表达式。

执行以下操作：

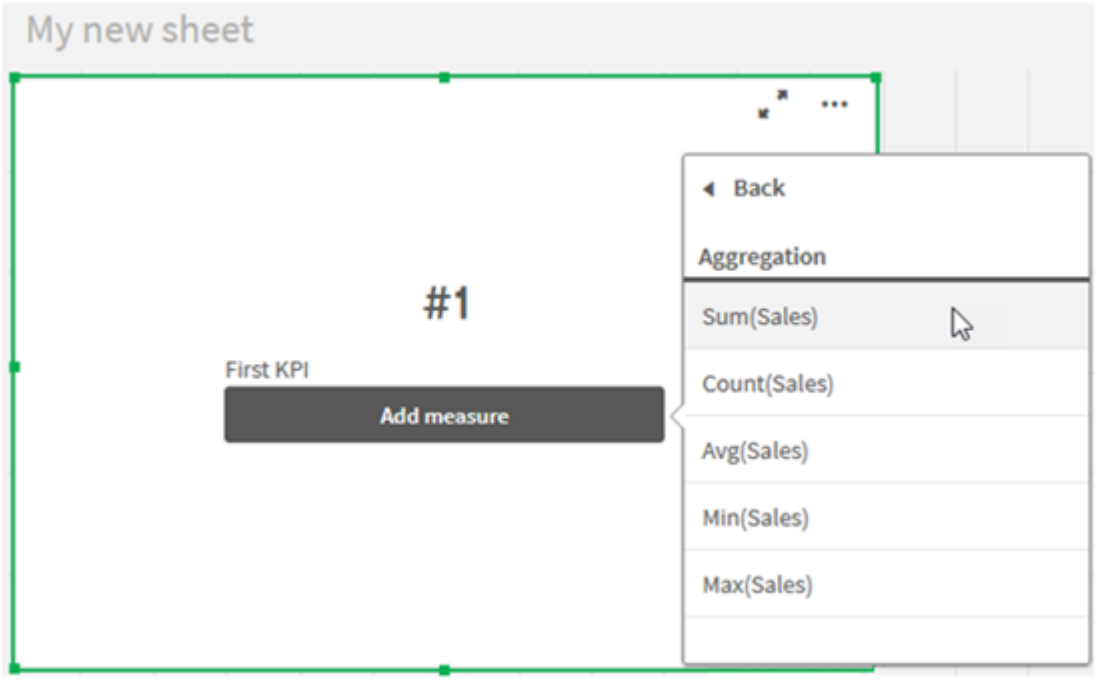
1. 在工作表中，从导航栏打开**资产**面板，然后单击**图表**。



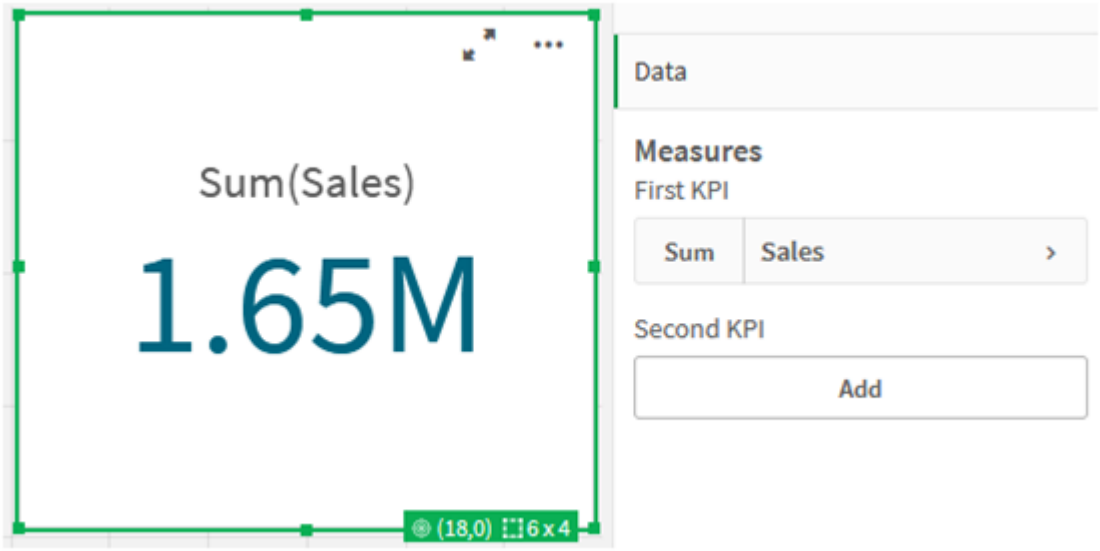
2. 将 **KPI** 拖到工作表上，然后单击**添加度量**。



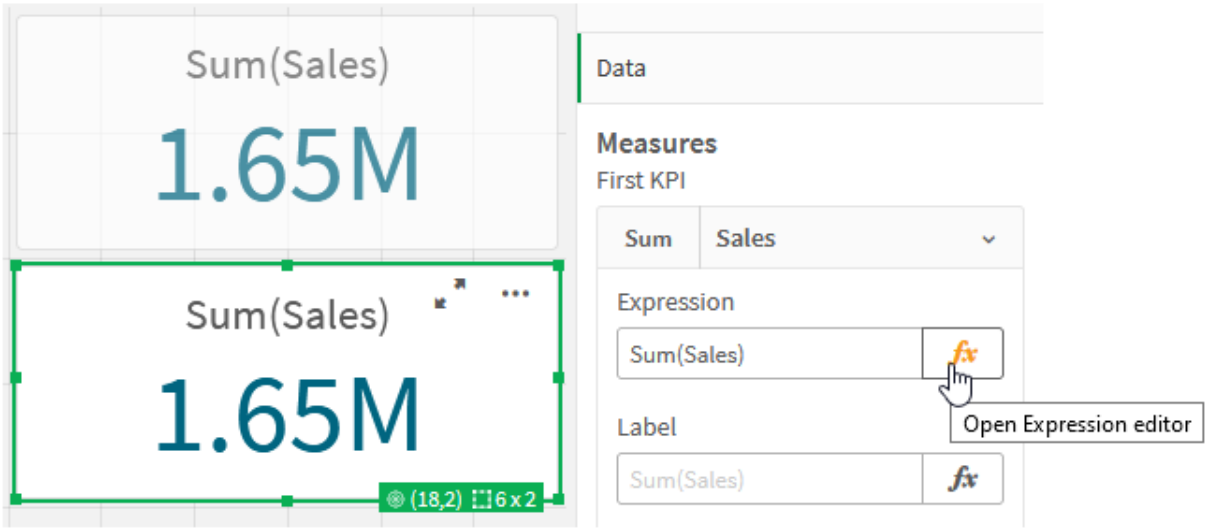
3. 单击 `Sales`，然后为聚合选择 `Sum(Sales)`。



KPI 显示所有年份的销售额总和。

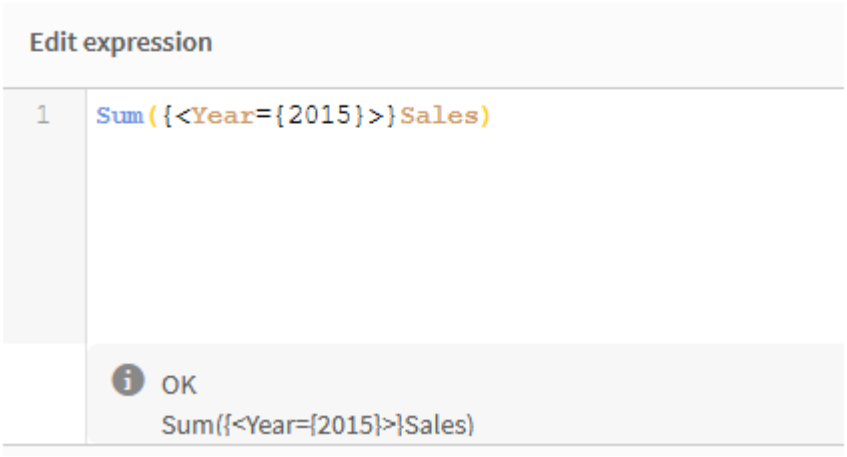


4. 复制并粘贴 KPI 以新建 KPI。
5. 单击新的 KPI，单击**度量**下方的**销售**，然后单击**打开表达式编辑器**。

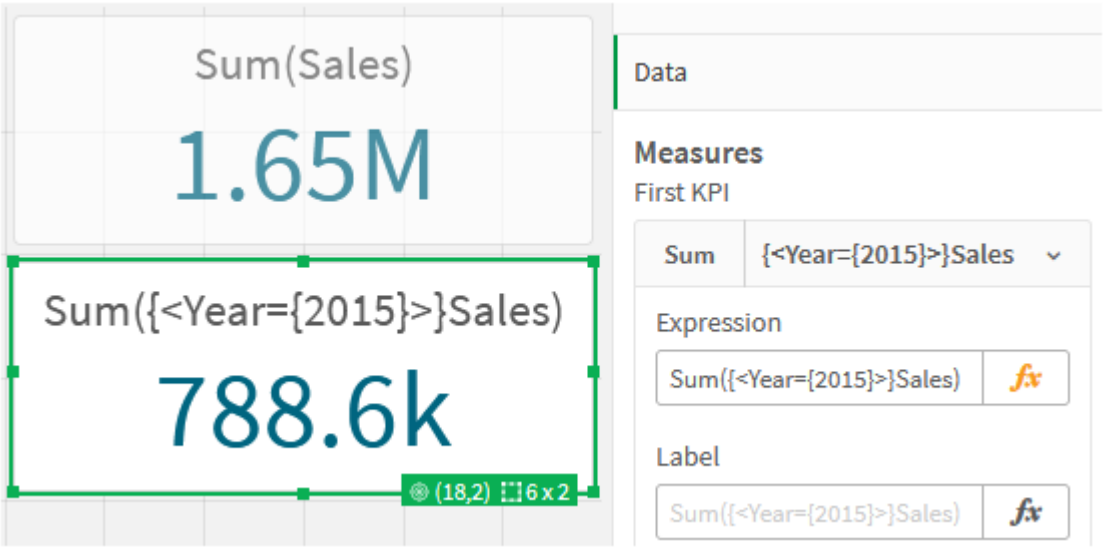


表达式编辑器连同聚合 `Sum(Sales)` 打开。

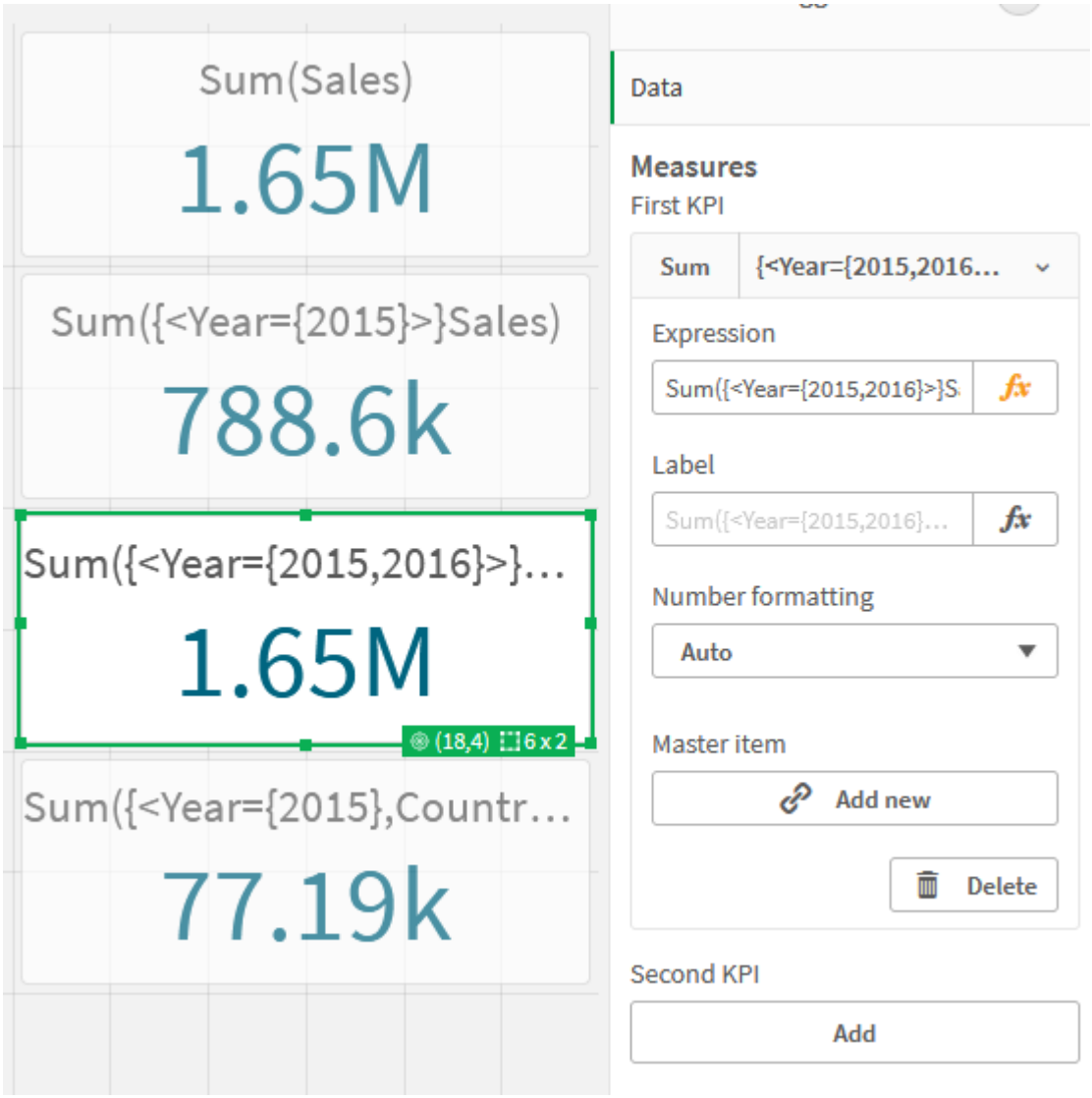
6. 在表达式编辑器中，创建表达式以仅对 2015 求 Sales 总和：
   i. 添加花括号以指示集合表达式：`Sum({}Sales)`
   i. 添加尖括号表示集合修饰符：`Sum({<>}Sales)`
   ii. 在尖括号中，添加要选择的字段，在本例中该字段为 `Year`，后跟等号。接下来，用另一组花括号括起 2015。所的的修饰符为：`{<Year={2015}>}`。
   整个表达式为：
   `Sum({<Year={2015}>}Sales)`



   iii. 单击**应用**保存表达式并关闭表达式编辑器。2015 年的 Sales总和如 KPI 中所示。



7. 使用以下表达式再创建两个 KPI：
   `Sum({<Year={2015,2016}>}Sales)`
   上方的修饰符为 `<Year={2015,2016}>`。表达式将返回 2015 年和 2016 年 Sales 的总和。
   `Sum({<Year={2015},Country={'Germany'}>} Sales)`
   上方的修饰符为 `<Year={2015}, Country={'Germany'}>`。表达式将返回 2015 年的 Sales 总和，其中 2015 年与 Germany 相交。
   使用集合修饰符的 KPI



添加集合标识符

上方的集合表达式将使用当前选择作为基础，因为未使用标识符。接下来，添加标识符以指定进行选择时的行为。

执行以下操作：

在工作表上，构建或复制以下集合表达式：

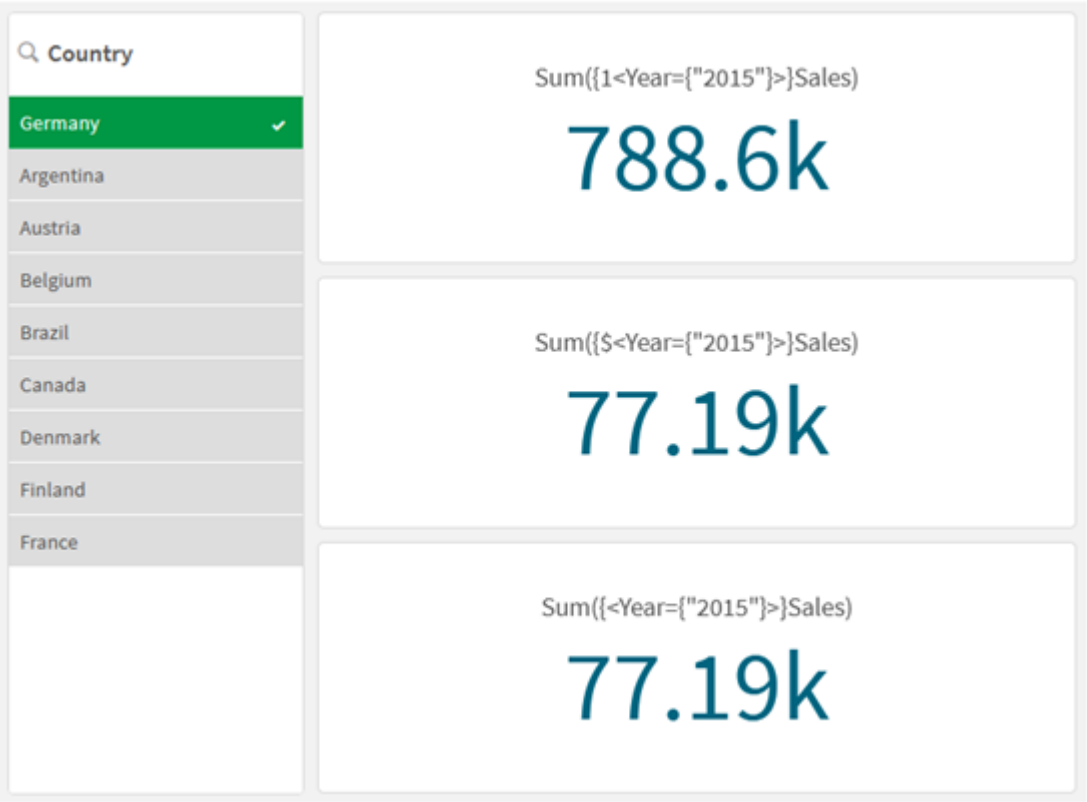
```
Sum({$<Year={"2015"}>}Sales
```

$ 标识符将根据数据中的当前选择创建集合表达式。这也是未使用标识符时的默认行为。

```
Sum({1<Year={"2015"}>}Sales)
```

1 标识符将导致 2015 年上 Sum(Sales) 的聚合忽略当前选择。当用户进行其他选择时，聚合值不会更改。例如，当在下面选择了 Germany 时，2015 年总金额的值不变。

使用集合修饰符和标识符的 KPI



添加运算符

集合运算符用于包含、排除或交汇数据集。所有运算符都将集合用作操作数，并返回集协作为结果。

可以在两种不同的情况下使用集合运算符：

- 对集合标识符执行集合操作，表示数据中的记录集合。
- 对元素集、字段值或集合修饰符内部执行集合操作。

执行以下操作：

在工作表上，构建或复制以下集合表达式：

```
Sum({$<Year={2015}>+1<Country={'Germany'}>}Sales)
```

这里，加号 (+) 运算符为 2015 和 Germany 生成数据集的并集。正如上文对集合标识符所解释的，美元符号 ($) 标识符表示将使用第一个操作数 <Year={2015}> 的当前选择。1 标识符表示将忽略第二个操作数 <Country={'Germany'}> 的选择。

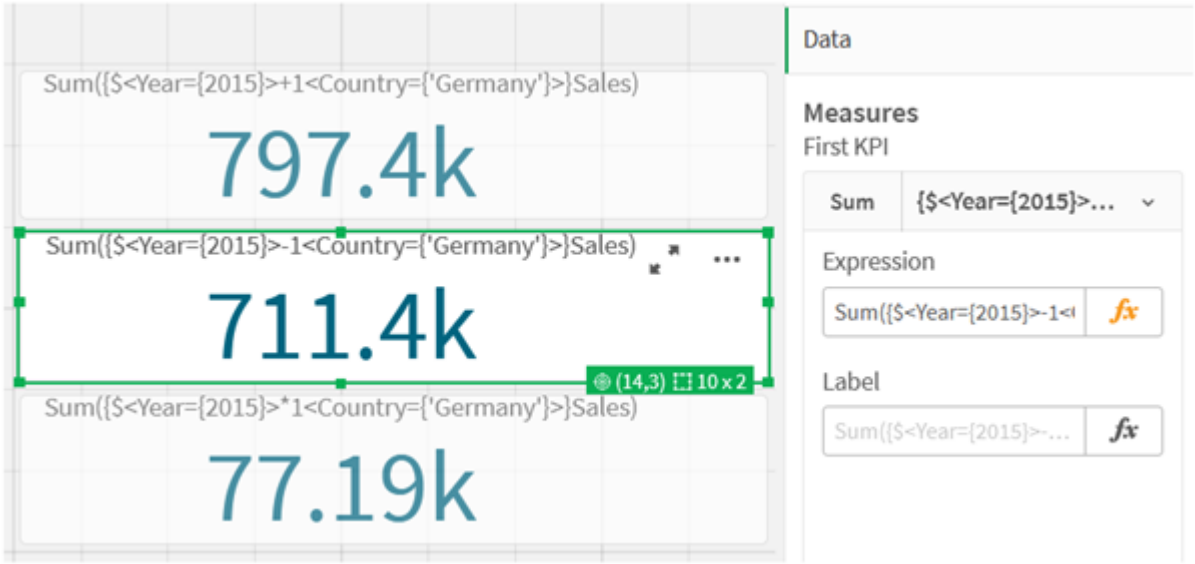
使用加号 (+) 运算符的 KPI

或者，使用减号 (-) 返回由属于 2015 但不属于 Germany 的记录组成的数据集。或者，使用星号 (*) 返回由属于这两个集合的记录组成的集合。

```
Sum({$<Year={2015}>-1<Country={'Germany'}>}Sales)
```

```
Sum({$<Year={2015}>*1<Country={'Germany'}>}Sales)
```

使用运算符的 KPI



集合表达式教程数据

加载脚本

将以下数据作为内联加载载入，然后在教程中创建图表表达式。

```
//Create table SalesByCountry SalesByCountry: Load * Inline [ Country, Year, Sales Argentina,
2016, 66295.03 Argentina, 2015, 140037.89 Austria, 2016, 54166.09 Austria, 2015, 182739.87
Belgium, 2016, 182766.87 Belgium, 2015, 178042.33 Brazil, 2016, 174492.67 Brazil, 2015,
2104.22 Canada, 2016, 101801.33 Canada, 2015, 40288.25 Denmark, 2016, 45273.25 Denmark, 2015,
106938.41 Finland, 2016, 107565.55 Finland, 2015, 30583.44 France, 2016, 115644.26 France,
2015, 30696.98 Germany, 2016, 8775.18 Germany, 2015, 77185.68 ];
```

集合表达式语法

完整语法(不包括选用标准括号定义优先级)使用 Backus-Naur 形式进行介绍：

```
set_expression ::= { set_entity { set_operator set_entity } }
set_entity ::= set_identifier [ set_modifier ] | set_modifier
set_identifier ::= 1 | $ | $N | $_N | bookmark_id | bookmark_name
set_operator ::= + | - | * | /
set_modifier ::= < field_selection {, field_selection } >
field_selection ::= field_name [ = | += | -= | *= | /= ] element_set_
expression
element_set_expression ::= [ - ] element_set { set_operator element_set }
element_set ::= [ field_name ] | { element_list } | element_function
element_list ::= element { , element }
element_function ::= ( P | E ) ( [set_expression] [field_name] )
element ::= field_value | " search_mask "
```

3.3 图表表达式的一般语法

以下通用语法结构可用于图表表达式，具有许多可选参数：

```
expression ::= ( constant | expressionname | operator1 expression | expression operator2
expression | function | aggregation function | (expression ) )
```

其中：

constant 是由单引号括起来的字符串(文本，日期或时间)或数字。写入的常数没有千分位分隔符，但使用小数点作为小数位分隔符。

expressionname 是同一个图表中另一个表达式的名称(标签)。

operator1 是一元运算符(作用于一个表达式，位于右边)。

operator2 是二元运算符(作用于两个表达式，每边一个)。

```
function ::= functionname ( parameters )
parameters ::= expression { , expression }
```

参数的数字和类型不是任意的。它们取决于所使用的函数。

```
aggregationfunction ::= aggregationfunctionname ( parameters2 )
parameters2 ::= aggrexpression { , aggrexpression }
```

参数的数字和类型不是任意的。它们取决于所使用的函数。

3.4 聚合的一般语法

以下通用语法结构可用于聚合，具有许多可选参数：

```
aggrexpression ::= ( fieldref | operator1 aggrexpression | aggrexpression operator2
aggrexpression | functioninaggr | ( aggrexpression ) )
```

fieldref 是字段名。

```
functionaggr ::= functionname ( parameters2 )
```

表达式和函数因此可以自由放置，只要 **fieldref** 始终正好被一个聚合函数包围，并且如果表达式返回一个可解释的值，Qlik Sense 就不会给出任何错误信息。

4 运算符

本节介绍可在 Qlik Sense 中使用的运算符。可以使用两种类型的运算符:

- 一元运算符(只需要一个操作数)
- 二元运算符(需要两个操作数)

大多数运算符是二元的。

可定义以下运算符:

- 位运算符
- 逻辑运算符
- 数字运算符
- 关系运算符
- 字符串运算符

4.1 位运算符

所有位运算符可将操作数转换(截断)为带正负号的整数(32 位),并以相同方式返回结果。逐位执行所有运算。如果不能将操作数解释为一个数字,该操作将返回 NULL。

位运算符

运算符	全名	说明
bitnot	位元反置	一元运算符运算返回逐位执行的操作数的逻辑反置。 **示例:** `bitnot 17` 返回 -18
bitand	位与	运算返回逐位执行的操作数的逻辑 AND。 **示例:** `17 bitand 7` 返回 1
bitor	位或	运算返回逐位执行的操作数的逻辑 OR。 **示例:** `17 bitor 7` 返回 23
bitxor	位异或	运算返回逐位执行的操作数的逻辑异或。 **示例:** `17 bitxor 7` 返回 22

运算符	全名	说明
>>	位右移	该操作返回向右移的第一个操作数。步数在第二个操作数中进行定义。 **示例：** 8 >> 2 返回 2
<<	位左移	该操作返回向左移的第一个操作数。步数在第二个操作数中进行定义。 **示例：** 8 << 2 返回 32

4.2 逻辑运算符

所有逻辑运算符都可解释逻辑操作数并返回结果 True (-1) 或 False (0)。

逻辑运算符

运算符	说明
not	逻辑反。很少使用的一元运算符。此运算返回操作数的逻辑反值。
and	逻辑与。此运算返回操作数的逻辑与。
or	逻辑或。此运算返回操作数的逻辑或。
Xor	逻辑异或。此运算返回操作数的逻辑异或。运算规则很像逻辑或，但不同的是，如果两个操作数都是 True，则结果为 False。

4.3 数字运算符

全部数字运算符使用数值式操作数并返回数值结果。

数字运算符

运算符	说明
+	正值(一元运算符)符号或算术加法。二元运算返回两个操作数的总和。
-	负值(一元运算符)符号或算术减法。一元运算返回操作数乘以 -1 的结果，而二元运算返回这两个操作数的差值。
*	算术乘法。此运算返回两个操作数的相乘结果。
/	算术除法。此运算返回两个操作数之间的比率。

4.4 关系运算符

所有关系运算符均会比较操作数值，并返回 True (-1) 或 False (0) 作为结果。所有关系运算符均为二进制。

关系运算符

运算符	说明
<	小于。如果两个操作书可使用数值解释，即可进行数值比较。运算操作会返回比较评估的逻辑值。
<=	小于或等于。如果两个操作书可使用数值解释，即可进行数值比较。运算操作会返回比较评估的逻辑值。
>	大于。如果两个操作书可使用数值解释，即可进行数值比较。运算操作会返回比较评估的逻辑值。
>=	大于或等于。如果两个操作书可使用数值解释，即可进行数值比较。运算操作会返回比较评估的逻辑值。
=	等于。如果两个操作书可使用数值解释，即可进行数值比较。运算操作会返回比较评估的逻辑值。
<>	不等于。如果两个操作书可使用数值解释，即可进行数值比较。运算操作会返回比较评估的逻辑值。
precedes	不同于 **<** 运算符，比较之前无须尝试用数值解释参数值。如果运算符左边的值拥有文本呈现形式，且该文本呈现形式在字符串比较中位于右边值文本呈现形式之前，则运算操作会返回正确结果。 **示例：** `'1 ' precedes ' 2'` 返回 FALSE `' 1' precedes ' 2'` 返回 TRUE 因为空格 (' ') 的 ASCII 值是小于数字的 ASCII 值的值。 将该示例与以下示例进行比较： `'1 ' < ' 2'` 返回 TRUE `' 1' < ' 2'` 返回 TRUE

运算符	说明
follows	不同于 **>** 运算符，比较之前无须尝试用数值解释参数值。如果运算符左边的值拥有文本呈现形式，且该文本呈现形式在字符串比较中位于右边值文本呈现形式之后，则运算操作会返回正确结果。 **示例：** `' 2' follows '1'` 返回 FALSE ' 2' follows ' 1' 返回 TRUE 因为空格 (' ') 的 ASCII 值是小于数字的 ASCII 值的值。 将该示例与以下示例进行比较： `' 2' > ' 1'` 返回 TRUE `' 2' > '1 '` 返回 TRUE

4.5 字符串运算符

有两种字符串运算符。一种使用操作数的字符串值并返回字符串结果。另一种比较操作数，然后返回布尔值以表明匹配情况。

&

字符串串联运算。此运算可以返回一个文本字符串，包含两个轮换操作数字符串。

示例：

`'abc' & 'xyz'` 返回“abcxyz”

like

字符串与通配符字符相比较。如果运算符之前的字符串与运算符之后的字符串相匹配，则此运算将返回布尔值 True (-1)。第二个字符串可能包含星号 * 通配符（任意数量的任意字符）或问号 ?（一个任意字符）。

示例：

`'abc' like 'a*'` 用于返回 True (-1)

`'abcd' like 'a?c*'` 用于返回 True (-1)

`'abc' like 'a??bc'` 用于返回 False (0)

5 脚本和图表函数

使用加载脚本和图表表达式中的函数转换和聚合数据。

许多函数能够以相同的方式用于数据加载脚本和图表表达式，但也有一些例外：

- 一些函数只能用于数据加载脚本，称为脚本函数。
- 一些函数只能用于图表表达式，称为图表函数。
- 一些函数可用于数据加载脚本和图表表达式，但在参数和应用方面存在差异。在不同的主题中分别介绍了脚本函数或图表函数。

5.1 用于服务器端扩展的分析连接 (SSE)

仅当您配置了分析连接并且启动了 Qlik Sense 时，通过分析连接启用的功能才可用。

您可在 QMC 中配置分析连接，请参阅指南管理 Qlik Sense 站点中的主题“创建分析连接”。

在 Qlik Sense Desktop 中，可通过编辑 *Settings.ini* 文件配置分析连接，请参阅指南 Qlik Sense Desktop 中的“配置 Qlik Sense Desktop 中的分析连接”。

5.2 聚合函数

被称为聚合函数的函数家族包含将多个字段值作为其输入信息并每个组返回单个结果的函数，在此类函数中，分组通过图表维度或脚本语句中的 **group by** 子句定义。

聚合函数包括 **Sum()**、**Count()**、**Min()**、**Max()** 等更多函数。

大多数聚合函数均可在数据加载脚本和图表表达式中使用，但语法不同。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

在数据加载脚本中使用聚合函数

聚合函数只能在 **LOAD** 和 **SELECT** 语句内使用。

在图表表达式中使用聚合函数

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

聚合函数会聚合选择项定义的可能记录集合。但替代记录集合可使用集合分析中的集合表达式定义。

如何计算聚合

聚合在特定表的记录上循环，聚合其中的记录。例如，**计数**(<Field>) 将计算 <Field> 所在的表格中记录的数目。如果只想聚合不同字段值，则需要使用 **distinct** 子句，例如 **Count**(**distinct** <Field>)。

如果聚合函数包含来自不同表的字段，则聚合函数将循环遍历组成字段的表的交叉乘积记录。这会降低性能，因此应该避免此类聚合，尤其在您有大量数据时是这样。

关键字段聚合

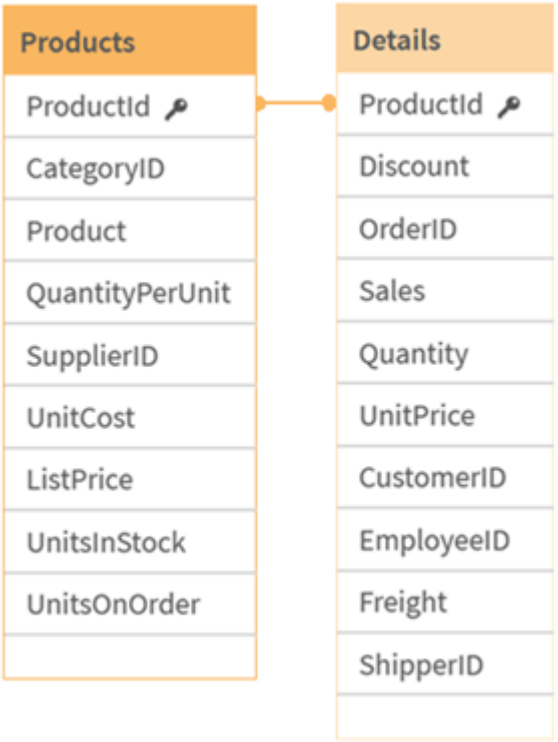
聚合的计算方式意味着您不能聚合关键字段，因为不清楚应该使用哪个表进行聚合。例如，如果字段 <Key> 链接两个表，则不清楚**Count**(<Key>) 是否应返回第一个表或第二个表中的记录数。

但是，如果使用 **distinct** 子句，则聚合是定义良好的，可以进行计算。

因此，如果在聚合函数中使用关键字段而不使用 **distinct** 子句，则 Qlik Sense 将返回一个可能毫无意义的数字。解决方案是要么使用 **distinct** 子句，要么使用关键字段的副本 – 一个只驻留在一个表中的副本。

例如，在下表中，ProductID 是表之间的键。

产品和明细表之间的 ProductID 键



Count(ProductID) 可以在 Products 表中计数（每个产品只有一条记录 - ProductID 是主键），也可以在 Details 表中计数（每个产品很可能有多条记录）。如果要计算不同产品的数量，应该使用 Count (distinct ProductID)。如果要计算特定表中的行数，则不应使用键。

基本聚合函数

基本聚合函数概述

基本聚合函数是一组最常用的聚合函数。

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

数据加载脚本中的基本聚合函数

FirstSortedValue

FirstSortedValue() 将返回来自 **value** 指定表达式的值，相当于 **sort_weight** 参数排序的结果，例如，单价最低的产品名称。排序顺序中的第 n 个值，可在 **rank** 中指定。如果在指定 **rank** 下多个结果值共享同一 **sort_weight**，则此函数返回 NULL。排序的值会迭代于 **group by** 子句定义的大量记录，或者如果 **group by** 子句未定义，就会在整个数据集之间聚合。

```
FirstSortedValue ([ distinct ] expression, sort_weight [, rank ])
```

Max

Max() 用于查找表达式中聚合数据的最高数值，该数值由 **group by** 子句定义。通过指定 **rank** n，可以查找第 n 个最高值。

```
Max ( expression[, rank])
```

Min

Min() 用于返回表达式中聚合数据的最低数值，该数值由 **group by** 子句定义。通过指定 **rank** n，可以查找第 n 个最低值。

```
Min ( expression[, rank])
```

Mode

Mode() 用于返回表达式中聚合数据的最常出现的值(即模式值)，该值由 **group by** 子句定义。**Mode()** 函数用于返回数字值和文本值。

```
Mode (expression )
```

Only

Only() 用于返回一个值(如果从聚合数据得出一个且只有一个可能值)。如果记录只包含一个值，则返回该值，否则返回 NULL 值。使用 **group by** 子句计算多个记录的值。**Only()** 函数用于返回数字值和文本值。

```
Only (expression )
```

Sum

Sum() 用于计算表达式中聚合的值的总和，该总和由 **group by** 子句定义。

```
Sum ([distinct]expression)
```

图表表达式中的基本聚合函数

图表聚合函数只能在图表表达式的字段中使用。单个聚合函数的参数表达式不能包含其他聚合函数。

FirstSortedValue

FirstSortedValue() 将返回来自 **value** 指定表达式的值，相当于 **sort_weight** 参数排序的结果，例如，单价最低的产品名称。排序顺序中的第 n 个值，可在 **rank** 中指定。如果在指定 **rank** 下多个结果值共享同一 **sort_weight**，则此函数返回 NULL。

```
FirstSortedValue - 图表函数([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] value, sort_weight [,rank])
```

Max

Max() 用于查找聚合数据白最高值。通过指定 **rank** n，可以查找第 n 个最高值。

```
Max - 图表函数Max() 用于查找聚合数据白最高值。通过指定 rank n，可以查找第 n 个最高值。 您可能还想查看 FirstSortedValue 和 rangemax，其功能与 Max 函数相似。 Max([{SetExpression}] [TOTAL [<fld {,fld}>]] expr [,rank])数字 参数参数说明expr表达式或字段包含要度量的数据。rankrank 的默认值为 1，相当于最高值。通过指定 rank 为 2，将返回第二个最高值。如果指定 rank 为 3，将返回第三个最高值，以此类推。SetExpression聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。 TOTAL 如果在函数参数前面出现单词 TOTAL，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 TOTAL [<fld {.fld}>](其中 TOTAL 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。 数据CustomerProductUnitSalesUnitPrice
AstridaAA416AstridaAA1015AstridaBB99BetacabBB510BetacabCC220BetacabDD-25CanutilityAA815CanutilityCC-19示例和结果示例结果Max(UnitSales)10，因为这是 UnitSales 中的最大值。订单的值通过销售的单位数量 (UnitSales) 乘以单位价格计算得出。Max(UnitSales*UnitPrice)150，因为这是计算 (UnitSales)*(UnitPrice) 所有可能值的结果的最大值。Max(UnitSales, 2)9，这是第二大值。Max(TOTAL UnitSales)10，因为 TOTAL 限定符意味着在忽略图表维度的情况下找到的最大值。对于以 Customer 为维度的图表，TOTAL 限定符将确保返回整个数据集的最大值，而不是每个客户的最大 UnitSales 值。创建选择项 Customer B。Max({1} TOTAL UnitSales)10，与创建的选择项无关，因为不管作出哪种选择，Set Analysis 表达式 {1} 都会定义该记录集合将被评估为 ALL。示例中所使用的数据：ProductData:LOAD * inline
[Customer|Product|UnitSales|UnitPriceAstrida|AA|4|16Astrida|AA|10|15Astrida|BB|9|9Betacab|BB|5|10Betacab|CC|2|20Betacab|DD||25Canutility|AA|8|15Canutility|CC||19] (delimiter is '|'); FirstSortedValue RangeMax ([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr [,rank])
```

Min

Min() 用于查找聚合数据白最低值。通过指定 **rank** n，可以查找第 n 个最低值。

```
Min - 图表函数([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr [,rank])
```

Mode

Mode() 用于查找聚合数据的最常出现的值(即模式值)。**Mode()** 函数可处理文本值和数字值。

```
Mode - 图表函数 ({[SetExpression] [TOTAL [<fld {,fld}>]]} expr)
```

Only

Only() 用于返回一个值(如果从聚合数据得出一个且只有一个可能值)。例如，如果有多个产品的单价为 9，则只搜索单价为 9 的产品将会返回 NULL。

```
Only - 图表函数([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr)
```

Sum

Sum() 用于计算聚合数据之间表达式或字段指定值的总和。

```
Sum - 图表函数([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr])
```

FirstSortedValue

FirstSortedValue() 将返回来自 **value** 指定表达式的值，相当于 **sort_weight** 参数排序的结果，例如，单价最低的产品名称。排序顺序中的第 n 个值，可在 **rank** 中指定。如果在指定 **rank** 下多个结果值共享同一 **sort_weight**，则此函数返回 NULL。排序的值会迭代于 **group by** 子句定义的大量记录，或者如果 **group by** 子句未定义，就会在整个数据集之间聚合。

语法：

```
FirstSortedValue ([ distinct ] value, sort-weight [, rank ])
```

返回数据类型： 双

参数：

参数

参数	说明
value Expression	此函数用于查找表达式 **value** 的值，相当于 **sort_weight** 的排序结果。
sort-weight Expression	该表达式包含要排序的数据。找到 **sort_weight** 的第一个(最低)值，由 **value** 表达式的对应值确定。如果在 **sort_weight** 前面加一个减号，则此函数会返回最后一个(最高)排序值。
rank Expression	通过指定一个大于 1 的 **rank** “n”，您会获得第 n 个排序值。
distinct	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

脚本示例

示例	结果
Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|CustomerID Astrida|AA|1|10|1 Astrida|AA|7|18|1 Astrida|BB|4|9|1 Astrida|CC|6|2|1 Betacab|AA|5|4|2 Betacab|BB|2|5|2 Betacab|DD|12|25|2 Canutility|AA|3|8|3 Canutility|CC|13|19|3 Divadip|AA|9|16|4 Divadip|AA|10|16|4 Divadip|DD|11|10|4] (delimiter is '|'); FirstSortedValue: LOAD Customer,FirstSortedValue(Product, UnitSales) as MyProductWithSmallestOrderByCustomer Resident Temp Group By Customer;	Customer MyProductWithSmallestOrderByCustomer Astrida CC Betacab AA Canutility AA Divadip DD 函数将 UnitSales 按从最小到最大的顺序排列，通过UnitSales 的最小值(最小顺序) 寻找 Customer 的值。 因为 CC 与客户 Astrida 的最小顺序(UnitSales 的值 = 2) 对应。AA 与客户 Betacab 的最小顺序 (4) 对应，AA 与客户 Canutility 的最小顺序 (8) 对应，而 DD 与客户 Divadip. 的最小顺序 (10) 对应。
前提是 **Temp** 表格像之前的示例一样加载： LOAD Customer,FirstSortedValue(Product, -UnitSales) as MyProductWithLargestOrderByCustomer Resident Temp Group By Customer;	Customer MyProductWithLargestOrderByCustomer Astrida AA Betacab DD Canutility CC Divadip - 减号位于 sort_weight 参数之首，因此，该函数将最大的排在第一个。 因为 AA 与客户 Astrida 的最大顺序 (UnitSales 的值 : 18) 对应，DD 与客户 Betacab 的最大顺序 (12) 对应，而 CC 与客户 Canutility 的最大顺序 (13) 对应。客户 Divadip 的最大顺序 (16) 有两个相同的值，因此，它会生成空结果。
前提是 **Temp** 表格像之前的示例一样加载： LOAD Customer,FirstSortedValue(distinct Product, -UnitSales) as MyProductWithSmallestOrderByCustomer Resident Temp Group By Customer;	Customer MyProductWithLargestOrderByCustomer Astrida AA Betacab DD Canutility CC Divadip AA 这一点与之前的示例相同，使用了 distinct 限定符除外。在该子句中，Divadip 的重复结果会被忽略，从而允许返回非空值。

FirstSortedValue - 图表函数

FirstSortedValue() 将返回来自 **value** 指定表达式的值，相当于 **sort_weight** 参数排序的结果，例如，单价最低的产品名称。排序顺序中的第 n 个值，可在 **rank** 中指定。如果在指定 **rank** 下多个结果值共享同一 **sort_weight**，则此函数返回 NULL。

语法：

```
FirstSortedValue([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] value,
sort_weight [,rank])
```

返回数据类型： 双

参数：

参数

参数	说明
value	输出字段。此函数用于查找表达式 **value** 的值，相当于 **sort_weight** 的排序结果。
sort_weight	输入字段。该表达式包含要排序的数据。找到 **sort_weight** 的第一个(最低)值，由 **value** 表达式的对应值确定。如果在 **sort_weight** 前面加一个减号，则此函数会返回最后一个(最高)排序值。
rank	通过指定一个大于 1 的 **rank** “n”，您会获得第 n 个排序值。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

示例和结果：

数据

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10

Customer	Product	UnitSales	UnitPrice
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

示例和结果

示例	结果
`firstsortedvalue (Product, UnitPrice)`	BB, 这是具有最低 `Product`(9) 的 `UnitPrice`。
`firstsortedvalue (Product, UnitPrice, 2)`	BB, 这是具有第二低 `UnitPrice`(10) 的 `Product`。
`firstsortedvalue (Customer, -UnitPrice, 2)`	Betacab, 这是具有第二高 `Customer`(20) 的 `Product` 的 `UnitPrice`。
`firstsortedvalue (Customer, UnitPrice, 3)`	NULL, 因为有两个相同 `Customer`(第三低) Astrida(15) 的 Canutility 值(rank 和 `UnitPrice`) 。 使用 distinct 限定符来确保不会出现意外的空结果。
`firstsortedvalue (Customer, -UnitPrice*UnitSales, 2)`	Canutility, 这是具有第二高销售订单值(`Customer` 乘以 `UnitPrice` (120)) 的 `UnitSales`。

示例中所使用的数据:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

Max

Max() 用于查找表达式中聚合数据的最高数值, 该数值由 **group by** 子句定义。通过指定 **rank** n, 可以查找第 n 个最高值。

语法:

```
Max ( expr [, rank])
```

返回数据类型：数字

参数：

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。
rank Expression	**rank** 的默认值为 1，相当于最高值。通过指定 **rank** 为 2，将返回第二个最高值。如果指定 **rank** 为 3，将返回第三个最高值，以此类推。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

示例：

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');

Max:
LOAD Customer, Max(UnitSales) as MyMax Resident Temp Group By Customer;
```

结果表

Customer	MyMax
Astrida	18
Betacab	5
Canutility	8

示例：

前提是 **Temp** 表格像之前的示例一样加载：

```
LOAD Customer, Max(UnitSales,2) as MyMaxRank2 Resident Temp Group By Customer;
```

结果表

Customer	MyMaxRank2
Astrida	10
Betacab	4
Canutility	-

Max - 图表函数

Max() 用于查找聚合数据白最高值。通过指定 **rank** n, 可以查找第 n 个最高值。

*您可能还想查看 **FirstSortedValue** 和 **rangemax**, 其功能与 **Max** 函数相似。*

语法:

```
Max([{SetExpression}] [TOTAL [<fld {,fld}>]] expr [,rank])
```

返回数据类型: 数字

参数:

参数

参数	说明
expr	表达式或字段包含要度量的数据。
rank	**rank** 的默认值为 1, 相当于最高值。通过指定 **rank** 为 2, 将返回第二个最高值。如果指定 **rank** 为 3, 将返回第三个最高值, 以此类推。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
TOTAL	如果在函数参数前面出现单词 **TOTAL**, 则计算给出当前选择项的所有可能值, 而不只是属于当前维度值的那些值, 即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表) , 您可以创建总可能值的子集。

示例和结果:

数据

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9

Customer	Product	UnitSales	UnitPrice
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

示例和结果

示例	结果
`Max(UnitSales)`	10, 因为这是 `UnitSales` 中的最大值。
订单的值通过销售的单位数量 (`UnitSales`) 乘以单位价格计算得出。 `Max (UnitSales*UnitPrice)`	150, 因为这是计算 (`UnitSales`)*(`UnitPrice`) 所有可能值的结果的最大值。
`Max(UnitSales, 2)`	9, 这是第二大值。
`Max(TOTAL UnitSales)`	10, 因为 TOTAL 限定符意味着在忽略图表维度的情况下找到的最大值。对于以 Customer 为维度的图表, TOTAL 限定符将确保返回整个数据集的最大值, 而不是每个客户的最大 UnitSales 值。
创建选择项 `Customer B`。 `Max({1} TOTAL UnitSales)`	10, 与创建的选择项无关, 因为不管作出哪种选择, Set Analysis 表达式 {1} 都会定义该记录集合将被评估为 ALL。

示例中所使用的数据:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

另请参见:

- *FirstSortedValue - 图表函数 (page 198)*

RangeMax (page 655)

Min

Min() 用于返回表达式中聚合数据的最低数值，该数值由 **group by** 子句定义。通过指定 **rank** n，可以查找第 n 个最低值。

语法：

```
Min ( expr [, rank])
```

返回数据类型： 数字

参数：

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。
rank Expression	**rank** 的默认值为 1，相当于最小值。通过指定 **rank** 为 2，将返回第二个最小值。如果指定 **rank** 为 3，将返回第三个最小值，以此类推。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

示例：

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');
Min:
LOAD Customer, Min(UnitSales) as MyMin Resident Temp Group By Customer;
```

结果表

Customer	MyMin
Astrida	2

Customer	MyMin
Betacab	4
Canutility	8

示例：

前提是 **Temp** 表格像之前的示例一样加载：

```
LOAD Customer, Min(UnitSales,2) as MyMinRank2 Resident Temp Group By Customer;
```

结果表

Customer	MyMinRank2
Astrida	9
Betacab	5
Canutility	-

Min - 图表函数

Min() 用于查找聚合数据白最低值。通过指定 **rank** n，可以查找第 n 个最低值。

*您可能还想查看 **FirstSortedValue** 和 **rangemin**，其功能与 **Min** 函数相似。*

语法：

```
Min({[SetExpression] [TOTAL [<fld {,fld}>]]} expr [,rank])
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
rank	**rank** 的默认值为 1，相当于最小值。通过指定 **rank** 为 2，将返回第二个最小值。如果指定 **rank** 为 3，将返回第三个最小值，以此类推。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

示例和结果：

数据

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Min() 函数必须根据表达式所提供值的阵列返回一个非 NULL 值(如果有)。因此在示例中，因为在数据中存在 NULL 值，函数返回通过表达式计算的第一个非 NULL 值。

示例和结果

示例	结果
`Min(UnitSales)`	2，因为此值是 `UnitSales` 中的最小非 NULL 值。
订单的值通过销售的单位数量 (`UnitSales`) 乘以单位价格计算得出。 `Min (UnitSales*UnitPrice)`	40，因为这是计算 (`UnitSales`)*(`UnitPrice`) 所有可能值的最小非 NULL 值结果。
`Min(UnitSales, 2)`	4，这是第二小的值(在 NULL 值后)。
`Min(TOTAL UnitSales)`	2，因为 TOTAL 限定符意味着在忽略图表维度的情况下找到可能最小的值。对于以 Customer 为维度的图表，TOTAL 限定符将确保返回整个数据集的最小值，而不是每个客户的最小值 UnitSales。
创建选择项 Customer B。 `Min({1} TOTAL UnitSales)`	2，与 Customer B 的选择无关。 不管作出哪种选择，Set Analysis 表达式 {1} 都会定义该记录集合将被评估为 ALL。

示例中所使用的数据：

```
ProductData:
LOAD * inline [
```

```
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

另请参见：

- *FirstSortedValue - 图表函数 (page 198)*
- *RangeMin (page 659)*

Mode

Mode() 用于返回表达式中聚合数据的最常出现的值(即模式值)，该值由 **group by** 子句定义。**Mode()** 函数用于返回数字值和文本值。

语法：

```
Mode ( expr)
```

返回数据类型： 双

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。

限制：

如果不只一个值经常出现，则返回 NULL。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

脚本示例

示例	结果
Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|CustomerID Astrida|AA|1|10|1 Astrida|AA|7|18|1 Astrida|BB|4|9|1 Astrida|CC|6|2|1 Betacab|AA|5|4|2 Betacab|BB|2|5|2 Betacab|DD Canutility|DD|3|8 Canutility|CC] (delimiter is '|'); Mode: LOAD Customer, Mode(Product) as MyMostOftenSoldProduct Resident Temp Group By Customer;	MyMostOftenSoldProduct AA 因为 AA 是唯一售出过多次的产品。

Mode - 图表函数

Mode() 用于查找聚合数据的最常出现的值(即模式值)。**Mode()** 函数可处理文本值和数字值。

语法:

Mode({[SetExpression] [**TOTAL** [<fld {,fld}>]]} expr**)**

返回数据类型: 双

参数:

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
TOTAL	如果在函数参数前面出现单词 **TOTAL**, 则计算给出当前选择项的所有可能值, 而不只是属于当前维度值的那些值, 即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表), 您可以创建总可能值的子集。

示例和结果：

数据

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

示例和结果

示例	结果
Mode(UnitPrice) 选择 Customer A。	15, 因为这是 UnitSales 中最常出现的值。 返回 NULL (-)。没有一个单值会比其他值更频繁地出现。
Mode(Product) 选择 Customer A。	AA, 因为这是 Product 中最常出现的值。 返回 NULL (-)。没有一个单值会比其他值更频繁地出现。
Mode (TOTAL UnitPrice)	15, 因为即使忽略图表维度, TOTAL 限定符也意味着最常出现的值仍是 15。
选择 Customer B。 Mode({1} TOTAL UnitPrice)	15, 与作出的选择无关, 因为不管作出哪种选择, Set Analysis 表达式 {1} 都会定义该记录集合将被评估为 ALL。

示例中所使用的数据：

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

另请参见：

- *Avg - 图表函数 (page 246)*
- *Median - 图表函数 (page 280)*

Only

Only() 用于返回一个值(如果从聚合数据得出一个且只有一个可能值)。如果记录只包含一个值，则返回该值，否则返回 NULL 值。使用 **group by** 子句计算多个记录的值。**Only()** 函数用于返回数字值和文本值。

语法：

```
Only ( expr )
```

返回数据类型：双

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');
Only:
LOAD Customer, Only(CustomerID) as MyUniqIDCheck Resident Temp Group By Customer;
```

结果表

Customer	MyUniqIDCheck
Astrida	1 因为只有客户 Astrida 拥有包括 CustomerID 的完整记录。

Only - 图表函数

Only() 用于返回一个值(如果从聚合数据得出一个且只有一个可能值)。例如,如果有多个产品的单价为 9,则只搜索单价为 9 的产品将会返回 NULL。

语法:

```
Only([{SetExpression}] [TOTAL [<fld {,fld}>]] expr)
```

返回数据类型: 双

参数:

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
TOTAL	如果在函数参数前面出现单词 **TOTAL**,则计算给出当前选择项的所有可能值,而不只是属于当前维度值的那些值,即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表),您可以创建总可能值的子集。

如果在样本数据中有多个可能的值时您想要 NULL 结果,则使用 Only()。

示例和结果:

数据

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

示例和结果

示例	结果
Only({<UnitPrice={9}>} Product)	BB, 因为这是 Product 为 9 的唯一 UnitPrice。
Only({<Product={DD}>} Customer)	Betacab, 因为它是销售 Product 的唯一 Customer, 被称为“DD”。
Only({<UnitPrice={20}>} UnitSales)	UnitPrice 为 20 的 UnitSales 数量为 2, 因为只有一个 UnitPrice =20 的 UnitSales 值。
Only({<UnitPrice={15}>} UnitSales)	NULL, 因为有两个 UnitPrice = 15 的 UnitSales 值。

示例中所使用的数据:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

Sum

Sum() 用于计算表达式中聚合的值的总和, 该总和由 **group by** 子句定义。

语法:

```
sum ( [ distinct] expr)
```

返回数据类型: 数字

参数:

参数

参数	说明
distinct	如果在表达式前面出现单词 **distinct**, 则将忽略所有重复值。
expr Expression	表达式或字段包含要度量的数据。

示例和结果:

将示例脚本添加到应用程序并运行。然后, 至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');
Sum:
LOAD Customer, Sum(UnitSales) as MySum Resident Temp Group By Customer;
```

结果表

Customer	MySum
Astrida	39
Betacab	9
Canutility	8

Sum - 图表函数

Sum() 用于计算聚合数据之间表达式或字段指定值的总和。

语法：

```
Sum([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。 *虽然支持 DISTINCT 限定符，但需慎用，因为它可能会在遗漏某些数据的情况下让读者误以为显示了总计值。*

参数	说明
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

示例和结果：

数据

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

示例和结果

示例	结果
`Sum(UnitSales)`	38.`UnitSales` 中值的合计。
`Sum(UnitSales*UnitPrice)`	505.`UnitPrice` 乘以聚合 `UnitSales` 的合计。
`Sum (TOTAL UnitSales*UnitPrice)`	对表中的所有行以及总计都返回 505，因为在忽略图表维度的情况下，TOTAL 限定符意味着总和仍是 505。
创建选择项 `Customer B`。 `Sum({1} TOTAL UnitSales*UnitPrice)`	505，与创建的选择项无关，因为不管作出哪种选择，Set Analysis 表达式 {1} 都会定义该记录集合将被评估为 ALL。

示例中所使用的数据：

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
```

```
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

计数器聚合函数

计数器聚合函数用于返回通过数据加载脚本中多个记录或通过图表维度中多个值对表达式进行计数的各种类型。

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

数据加载脚本中的计数器聚合函数

Count

Count() 用于返回表达式中聚合的值的数量，该数量由 **group by** 子句定义。

```
Count ([distinct ] expression | * )
```

MissingCount

MissingCount() 用于返回表达式中聚合的缺失值的数量，该数量由 **group by** 子句定义。

```
MissingCount ([ distinct ] expression)
```

NullCount

NullCount() 用于返回表达式中聚合的 NULL 值的数量，该数量由 **group by** 子句定义。

```
NullCount ([ distinct ] expression)
```

NumericCount

NumericCount() 用于返回表达式中数值的数量，该数量由 **group by** 子句定义。

```
NumericCount ([ distinct ] expression)
```

TextCount

TextCount() 用于返回表达式中聚合的非数字的字段值的数量，该数量由 **group by** 子句定义。

```
TextCount ([ distinct ] expression)
```

图表表达式中的计数器聚合函数

以下计数器聚合函数可用于图表中：

Count

Count() 用于聚合每个图表维度中值、文本和数字的数量。

```
Count - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

MissingCount

MissingCount() 用于聚合每个图表维度中缺失值的数量。缺失值均是非数字值。

```
MissingCount - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] expr)
```

NullCount

NullCount() 用于聚合每个图表维度中 NULL 值的数量。

```
NullCount - 图表函数({[SetExpression][DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

NumericCount

NumericCount() 用于聚合每个图表维度中数值的数量。

```
NumericCount - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

TextCount

TextCount() 用于聚合每个图表维度中非数字字段值的数量。

```
TextCount - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

Count

Count() 用于返回表达式中聚合的值的数量，该数量由 **group by** 子句定义。

语法：

```
Count( [distinct ] expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
distinct	如果在表达式前出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

脚本示例

示例	结果
Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|UnitPrice Astrida|AA|1|4|16 Astrida|AA|7|10|15 Astrida|BB|4|9|9 Betacab|CC|6|5|10 Betacab|AA|5|2|20 Betacab|BB|1|25| 25 Canutility|AA|3|8|15 Canutility|CC||19 Divadip|CC|2|4|16 Divadip|DD|3|1|25] (delimiter is '|'); Count1: LOAD Customer,Count(OrderNumber) as OrdersByCustomer Resident Temp Group By Customer;	Customer OrdersByCustomer Astrida 3 Betacab 3 Canutility 2 Divadip 2 只要表格中的表格包含维度 Customer，否则 OrdersByCustomer 的结果为 3, 2。
前提是 **Temp** 表格像之前的示例一样加载： LOAD Count(OrderNumber) as TotalOrderNumber Resident Temp;	TotalOrderNumber 10
前提是 **Temp** 表格像第一个示例一样加载： LOAD Count(distinct OrderNumber) as TotalOrderNumber Resident Temp;	TotalOrderNumber 8 由于存在具有相同值 1 的两个 OrderNumber 值以及一个空值。

Count - 图表函数

Count() 用于聚合每个图表维度中值、文本和数字的数量。

语法：

```
Count({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

示例和结果：

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD	1	25	25
Canutility	AA	3	8	15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

除非另行说明，以下示例假定已选择所有客户。

示例和结果

示例	结果
`Count(OrderNumber)`	10，因为有 10 个字段包含 OrderNumber 值，并已对所有记录（甚至包括空白记录）计数。 *"0"计为一个值，而不是一个空单元格。但是，如果维度的度量聚合为 0，则图表中将不包括该维度。*
`Count(Customer)`	10，因为 Count 将评估全部字段中的发生次数。
`Count(DISTINCT [Customer])`	4，因为使用 Distinct 限定符，Count 仅评估唯一的发生次数。

示例	结果
假定已选择客户 Canutility `Count(OrderNumber)/Count({1} TOTAL OrderNumber)`	0.2，因为表达式会将所选客户的订单数量返回为相对于所有客户订单的百分比形式。在此例中为 2/10。
假定已选择客户 Astrida 和 Canutility `Count(TOTAL <Product> OrderNumber)`	5，因为该值是仅对所选客户的产品所下订单的数量，并且已对空白单元格计数。

示例中所使用的数据：

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB|1|25| 25
Canutility|AA|3|8|15
Canutility|CC|||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

MissingCount

MissingCount() 用于返回表达式中聚合的缺失值的数量，该数量由 **group by** 子句定义。

语法：

```
MissingCount ( [ distinct ] expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。
distinct	如果在表达式前出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

脚本示例

示例	结果
Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|UnitPrice Astrida|AA|1|4|16 Astrida|AA|7|10|15 Astrida|BB|4|9|9 Betacab|CC|6|5|10 Betacab|AA|5|2|20 Betacab|BB||| 25 Canutility|AA|||15 Canutility|CC| ||19 Divadip|CC|2|4|16 Divadip|DD|3|1|25] (delimiter is '|'); MissCount1: LOAD Customer,MissingCount(OrderNumber) as MissingOrdersByCustomer Resident Temp Group By Customer; Load MissingCount(OrderNumber) as TotalMissingCount Resident Temp;	Customer MissingOrdersByCustomer Astrida 0 Betacab 1 Canutility 2 Divadip 0 第二个语句指定： TotalMissingCount 3 在包含该维度的表格中。
前提是 **Temp** 表格像之前的示例一样加载： LOAD MissingCount(distinct OrderNumber) as TotalMissingCountDistinct Resident Temp;	TotalMissingCountDistinct 1 因为只有一个 OrderNumber 缺少一个值。

MissingCount - 图表函数

MissingCount() 用于聚合每个图表维度中缺失值的数量。缺失值均是非数字值。

语法：

MissingCount({[SetExpression] [**DISTINCT**] [**TOTAL** [<fld {,fld}>]]} expr**)**

返回数据类型： 整数

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

示例和结果：

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

示例和结果

示例	结果
`MissingCount([OrderNumber])`	3，因为 10 个 OrderNumber 字段中有 3 个是空白 *“0”计为一个值，而不是一个空单元格。但是，如果维度的度量聚合为 0，则图表中将不包括该维度。*
`MissingCount ([OrderNumber])/MissingCount ({1} Total [OrderNumber])`	表达式会将所选客户的不完整订单数量返回为相对于所有客户不完整订单数量的分数形式。在所有客户的 OrderNumber 值中，总共有 3 个缺失值。因此，对于 Product 有缺失值的每个 Customer，结果为 1/3。

示例中所使用的数据：

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
```

```
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC| ||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

NullCount

NullCount() 用于返回表达式中聚合的 NULL 值的数量，该数量由 **group by** 子句定义。

语法：

```
NullCount ( [ distinct ] expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。
distinct	如果在表达式前出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

脚本示例

示例	结果
Set NULLINTERPRET = NULL; Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|CustomerID Astrida|AA|1|10|1 Astrida|AA|7|18|1 Astrida|BB|4|9|1 Astrida|CC|6|2|1 Betacab|AA|5|4|2 Betacab|BB|2|5|2 Betacab|DD||| Canutility|AA|3|8| Canutility|CC|NULL||] (delimiter is '|'); Set NULLINTERPRET=; NullCount1: LOAD Customer,NullCount(OrderNumber) as NullOrdersByCustomer Resident Temp Group By Customer; LOAD NullCount(OrderNumber) as TotalNullCount Resident Temp;	Customer NullOrdersByCustomer Astrida 0 Betacab 0 Canutility 1 第二个语句指定: TotalNullCount 1 在包含该维度的表格中,因为只有一条记录包含 NULL 值。

NullCount - 图表函数

NullCount() 用于聚合每个图表维度中 NULL 值的数量。

语法:

NullCount({[SetExpression][**DISTINCT**] [**TOTAL** [<fld {,fld}>]]} expr**)**

返回数据类型: 整数

参数:

参数

参数	说明
expr	表达式或字段包含要度量的数据。
set_ expression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**,则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**,则计算给出当前选择项的所有可能值,而不只是属于当前维度值的那些值,即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表),您可以创建总可能值的子集。

示例和结果：

示例和结果

示例	结果
NullCount([OrderNumber])	1，因为我们在内联 **LOAD** 语句中使用 NullInterpret 引入了 NULL 值。

示例中所使用的数据：

```
Set NULLINTERPRET = NULL;
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD|||
Canutility|AA|3|8|
Canutility|CC|NULL||
] (delimiter is '|');
Set NULLINTERPRET=;
```

NumericCount

NumericCount() 用于返回表达式中数值的数量，该数量由 **group by** 子句定义。

语法：

```
NumericCount ( [ distinct ] expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。
distinct	如果在表达式前出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

脚本示例

示例	结果
`LOAD NumericCount(OrderNumber) as TotalNumericCount Resident Temp;`	第二个语句指定： TotalNumericCount 7 在包含该维度的表格中。
前提是 **Temp** 表格像之前的示例一样加载： `LOAD NumericCount(distinct OrderNumber) as TotalNumericCountDistinct Resident Temp;`	TotalNumericCountDistinct 6 因为只有一个 OrderNumber 与另一个重复，因此结果为 6 不会重复。

示例：

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC| ||19
Divadip|CC|2|4|16
Divadip|DD|7|1|25
] (delimiter is '|');
NumCount1:
LOAD Customer,NumericCount(OrderNumber) as NumericCountByCustomer Resident Temp Group By
Customer;
```

结果表

Customer	NumericCountByCustomer
Astrida	3
Betacab	2
Canutility	0
Divadip	2

NumericCount - 图表函数

NumericCount() 用于聚合每个图表维度中数值的数量。

语法：

```
NumericCount({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
set_expression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

示例和结果：

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

除非另行说明，以下示例假定已选择所有客户。

示例和结果

示例	结果
NumericCount ([OrderNumber])	7, 因为 OrderNumber 中的 10 个字段中有 3 个是空白。 *"0"计为一个值，而不是一个空单元格。但是，如果维度的度量聚合为 0，则图表中将不包括该维度。*
NumericCount ([Product])	0, 因为所有产品名称都是采用文本形式。通常可以使用此函数检查没有文本字段的内容为数字。
NumericCount (DISTINCT [OrderNumber])/Count (DISTINCT [OrderNumber)]	对不同数字订单号的所有数量计数，并除以数字和非数字订单号的数量。如果所有字段值都是数字值，则此值为 1。通常可以使用此函数检查所有字段值是否都是数字值。在此例中，OrderNumber 的 8 个不同的数值和非数值中有 7 个不同的数值，因此表达式返回 0.875。

示例中所使用的数据：

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC| ||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

TextCount

TextCount() 用于返回表达式中聚合的非数字的字段值的数量，该数量由 **group by** 子句定义。

语法：

```
TextCount ( [ distinct ] expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr Expression	表达式或字段包含要度量的数据。
distinct	如果在表达式前出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

示例：

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC| ||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
TextCount1:
LOAD Customer,TextCount(Product) as ProductTextCount Resident Temp Group By Customer;
```

结果表

Customer	ProductTextCount
Astrida	3
Betacab	3
Canutility	2
Divadip	2

示例：

```
LOAD Customer,TextCount(OrderNumber) as OrderNumberTextCount Resident Temp Group By Customer;
```

结果表

Customer	OrderNumberTextCount
Astrida	0
Betacab	1
Canutility	2
Divadip	0

TextCount - 图表函数

TextCount() 用于聚合每个图表维度中非数字字段值的数量。

语法：

```
TextCount({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

返回数据类型： 整数

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

示例和结果：

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

示例和结果

示例	结果
TextCount([Product])	10, 因为 Product 中的 10 个字段全部是文本字段。 *"0"计为一个值, 而不是一个空单元格。但是, 如果维度的度量聚合为 0, 则图表中将不包括该维度。空白单元格被评估为非文本, 因此不计入 TextCount。*
TextCount ([OrderNumber])	3, 因为已对空白单元格计数。通常将使用此函数检查是否有数字字段包含文本值或为非零值。
TextCount (DISTINCT [Product])/Count ([Product)]	对 Product 不同文本值的所有数量 (4) 计数, 并除以 Product 值的总数 (10)。结果为 0.4。

示例中所使用的数据:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|1|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB|||| 25
Canutility|AA|||15
Canutility|CC|||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

财务聚合函数

本部分介绍财务运作中与付款和现金流相关的聚合函数。

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

数据加载脚本中的财务聚合函数

IRR

IRR() 函数用于返回聚合内部回报率, 以揭示迭代于 group by 子句定义的大量记录上的表达式的数值表示的现金流系列。

```
IRR (expression)
```

XIRR

XIRR() 函数用于返回聚合内部回报率，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表(不必为周期性的)。所有付款按 365 天一年年折扣。

```
XIRR (valueexpression, dateexpression )
```

NPV

NPV() 用于根据迭代于 group by 子句定义的大量记录上的 **value** 的数值表示的一系列未来付款(负值)和收入(正值)以及每周期的 **discount_rate**，返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

```
NPV (rate, expression)
```

XNPV

XNPV() 函数用于返回聚合净现值，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表(不必为周期性的)。比率为每周期的利率。所有付款按 365 天一年年折扣。

```
XNPV (rate, valueexpression, dateexpression)
```

图表表达式中的财务聚合函数

以下财务聚合函数可用于图表中。

IRR

IRR() 用于返回通过在图表维度上迭代的 **value** 指定表达式中数值表示的一系列现金流的聚合内部回报率。

```
 IRR - 图表函数[TOTAL [<fld {,fld}>]] value)
```

NPV

NPV() 用于根据每周期的 **discount_rate** 和通过图表维度迭代 **value** 的数值表示的一系列未来付款(负值)和收入(正值)返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

```
NPV - 图表函数([TOTAL [<fld {,fld}>]] discount_rate, value)
```

XIRR

XIRR() 用于返回通过图表维度迭代 **pmt** 和 **date** 指定表达式中成对数值表示的现金流时间表(即不一定是周期性的)。所有付款按 365 天一年年折扣。

```
XIRR - 图表函数 (page 236)([TOTAL [<fld {,fld}>]] pmt, date)
```

XNPV

XNPV() 用于返回通过图表维度迭代 **pmt** 和 **date** 指定表达式中成对数值表示的现金流时间表(不一定是周期性)的聚合净现值。所有付款按 365 天一年年折扣。

```
XNPV - 图表函数([TOTAL [<fld{,fld}>]] discount_rate, pmt, date)
```

IRR

IRR() 函数用于返回聚合内部回报率，以揭示迭代于 group by 子句定义的大量记录上的表达式的数值表示的现金流系列。

这些现金流不必是均值，因为它们可用于年金。但是，现金流必须定期出现，例如每月或每年。内部回报率是指投资回报的利率，该利率由定期出现的支出（负值）和收入（正值）构成。计数函数至少需要一个正值和一个负值。

语法：

```
IRR(value)
```

返回数据类型： 数字

参数：

参数

参数	说明
value	表达式或字段包含要度量的数据。

限制：

文本值，NULL 值和缺失值都忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

示例和结果：

示例和结果

示例	年	IRR2013
Cashflow: LOAD 2013 as Year, * inline [Date|Discount|Payments 2013-01-01|0.1|-10000 2013-03-01|0.1|3000 2013-10-30|0.1|4200 2014-02-01|0.2|6800] (delimiter is '|'); Cashflow1: LOAD Year,IRR(Payments) as IRR2013 Resident Cashflow Group By Year;	2013	0.1634

IRR - 图表函数

IRR() 用于返回通过在图表维度上迭代的 **value** 指定表达式中数值表示的一系列现金流的聚合内部回报率。

这些现金流不必是均值，因为它们可用于年金。但是，现金流必须定期出现，例如每月或每年。内部收益率由定期发生的付款（负值）和收入（正值）构成的投资回报率决定。计算此函数至少需要一个正值和一个负值。

语法：

```
IRR([TOTAL [<fld {,fld}>]] value)
```

返回数据类型：数字

参数：

参数

参数	说明
value	表达式或字段包含要度量的数据。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

文本值，NULL 值和缺失值都忽略不计。

示例和结果：

示例和结果

示例	结果
`IRR (Payments)`	0.1634 假定付款是周期性的，例如每月。 *如果付款是非周期性的，则只要提供付款的日期，就可在 XIRR 示例中使用 Date 字段。*

示例中所使用的数据：

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Discount|Payments
2013-01-01|0.1|-10000
2013-03-01|0.1|3000
2013-10-30|0.1|4200
2014-02-01|0.2|6800
] (delimiter is '|');
```

另请参见：

- *XIRR - 图表函数 (page 236)*
- *Aggr - 图表函数 (page 373)*

NPV

NPV() 用于根据迭代于 group by 子句定义的大量记录上的 **value** 的数值表示的一系列未来付款(负值)和收入(正值)以及每周期的 **discount_rate**，返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

语法：

```
NPV(discount_rate, value)
```

返回数据类型：数字。结果默认采用货币数字格式。

参数：

参数

参数	说明
discount_rate	**discount_rate** 是在整个期间的折扣率。
value	表达式或字段包含要度量的数据。

限制：

文本值，NULL 值和缺失值都忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

示例和结果

示例	年	NPV1_2013
Cashflow: LOAD 2013 as Year, * inline [Date|Discount|Payments 2013-01-01|0.1|-10000 2013-03-01|0.1|3000 2013-10-30|0.1|4200 2014-02-01|0.2|6800] (delimiter is '|'); Cashflow1: LOAD Year,NPV(0.2, Payments) as NPV1_2013 Resident Cashflow Group By Year;	2013	-$540.12

示例和结果

示例	年	折扣	NPV2_2013
前提是 **Cashflow** 表格像之前的示例一样加载： `LOAD Year,NPV(Discount, Payments) as NPV2_2013 Resident Cashflow Group By Year, Discount;` 注意，Group By 子句按 Year 和 Discount 对结果进行排序。将第一个参数 discount_rate 指定为字段 (Discount)，而不是一个特定数字，因此需要第二个排序标准。字段可以包含不同的值，因此必须对聚合记录进行排序以允许不同的 Year 和 Discount 值。 ;	2013 2013	0.1 0.2	-$3456.05 $5666.67

NPV - 图表函数

NPV() 用于根据每周期的 **discount_rate** 和通过图表维度迭代 **value** 的数值表示的一系列未来付款（负值）和收入（正值）返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

语法：

```
NPV([TOTAL [<fld {,fld}>]] discount_rate, value)
```

返回数据类型： 数字 结果默认采用货币数字格式。

参数：

参数

参数	说明
discount_rate	**discount_rate** 是在整个期间的折扣率。
value	表达式或字段包含要度量的数据。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。 **TOTAL** 限定符后可能紧跟着一系列由尖括号括起来的一个或多个字段名。这些字段名应该是图表维度变量的子集。此时，计算会忽略所有图表维度变量，但会计算已列出的变量，即列出的维度字段内字段值的各组合均会返回一个值。此外，当前并非为图表内维度的字段也可能会包括在列表之中。这对于组维度可能极为有用，其中未固定维度字段。在组中列出全部变量会导致函数在钻取级变化时生效。

限制：

discount_rate 和 **value** 不能包含聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

文本值，NULL 值和缺失值都忽略不计。

示例和结果：

示例和结果

示例	结果
NPV(Discount, Payments)	-$540.12

示例中所使用的数据：

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Discount|Payments
2013-01-01|0.1|-10000
2013-03-01|0.1|3000
2013-10-30|0.1|4200
2014-02-01|0.2|6800
] (delimiter is '|');
```

另请参见：

- *XNPV - 图表函数 (page 238)*
- *Aggr - 图表函数 (page 373)*

XIRR

XIRR() 函数用于返回聚合内部回报率，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表（不必为周期性的）。所有付款按 365 天一年年折扣。

语法：

```
XIRR(pmt, date )
```

返回数据类型： 数字

参数：

参数

参数	说明
pmt	付款。表达式或字段包含与在 **date** 中指定的付款时间表对应的现金流。
date	表达式或字段包含与在 **pmt** 中指定的现金流支付对应的日期时间表。

限制：

数据对的任意部分或两部分内存在文本值、NULL 值和缺失值会导致整个数据对被忽略。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

示例和结果

示例	年	XIRR2013
Cashflow: LOAD 2013 as Year, * inline [Date|Discount|Payments 2013-01-01|0.1|-10000 2013-03-01|0.1|3000 2013-10-30|0.1|4200 2014-02-01|0.2|6800] (delimiter is '|'); Cashflow1: LOAD Year,XIRR(Payments, Date) as XIRR2013 Resident Cashflow Group By Year;	2013	0.5385

XIRR - 图表函数

XIRR() 用于返回通过图表维度迭代 **pmt** 和 **date** 指定表达式中成对数值表示的现金流时间表(即不一定是周期性的)。所有付款按 365 天一年年折扣。

语法：

```
XIRR([TOTAL [<fld {,fld}>]] pmt, date)
```

返回数据类型： 数字

参数：

参数

参数	说明
pmt	付款。表达式或字段包含与在 **date** 中指定的付款时间表对应的现金流。
date	表达式或字段包含与在 **pmt** 中指定的现金流支付对应的日期时间表。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

pmt 和 **date** 不能包含聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

示例和结果：

示例和结果

示例	结果
XIRR(Payments, Date)	0.5385

示例中所使用的数据：

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Discount|Payments
2013-01-01|0.1|-10000
2013-03-01|0.1|3000
2013-10-30|0.1|4200
2014-02-01|0.2|6800
] (delimiter is '|');
```

另请参见：

- *IRR - 图表函数 (page 231)*
- *Aggr - 图表函数 (page 373)*

XNPV

XNPV() 函数用于返回聚合净现值，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表（不必为周期性的）。比率为每周期的利率。所有付款按 365 天一年年折扣。

语法：

```
XNPV(discount_rate, pmt, date)
```

返回数据类型： 数字。结果默认采用货币数字格式。.

参数：

参数

参数	说明
discount_rate	**discount_rate** 是在整个期间的折扣率。
pmt	表达式或字段包含要度量的数据。
date	表达式或字段包含与在 **pmt** 中指定的现金流支付对应的日期时间表。

限制：

数据对的任意部分或两部分内存在文本值、NULL 值和缺失值会导致整个数据对被忽略。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

示例和结果

示例	年	XNPV1_2013
Cashflow: LOAD 2013 as Year, * inline [Date|Discount|Payments 2013-01-01|0.1|-10000 2013-03-01|0.1|3000 2013-10-30|0.1|4200 2014-02-01|0.2|6800] (delimiter is '|'); Cashflow1: LOAD Year,XNPV(0.2, Payments, Date) as XNPV1_2013 Resident Cashflow Group By Year;	2013	$2104.37

示例和结果：

示例	年	折扣	XNPV2_2013
前提是 **Cashflow** 表格像之前的示例一样加载： LOAD Year,XNPV(Discount, Payments, Date) as XNPV2_2013 Resident Cashflow Group By Year, Discount; 注意，Group By 子句按 Year 和 Discount 对结果进行排序。将第一个参数 discount_rate 指定为字段 (Discount)，而不是一个特定数字，因此需要第二个排序标准。字段可以包含不同的值，因此必须对聚合记录进行排序以允许不同的 Year 和 Discount 值。	2013 2013	0.1 0.2	-$3164.35 $6800.00

XNPV - 图表函数

XNPV() 用于返回通过图表维度迭代 **pmt** 和 **date** 指定表达式中成对数值表示的现金流时间表(不一定是周期性)的聚合净现值。所有付款按 365 天一年年折扣。

语法：

```
XNPV([TOTAL [<fld{,fld}>]] discount_rate, pmt, date)
```

返回数据类型：数字 结果默认采用货币数字格式。

参数：

参数

参数	说明
discount_rate	**discount_rate** 是在整个期间的折扣率。
pmt	付款。表达式或字段包含与在 **date** 中指定的付款时间表对应的现金流。
date	表达式或字段包含与在 **pmt** 中指定的现金流支付对应的日期时间表。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

discount_rate、**pmt** 和 **date** 不能包含聚合函数，除非这些内部聚合包含 **TOTAL** 或 **ALL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

示例和结果：

示例和结果

示例	结果
`XNPV(Discount, Payments, Date)`	-$3164.35

示例中所使用的数据：

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Discount|Payments
2013-01-01|0.1|-10000
2013-03-01|0.1|3000
2013-10-30|0.1|4200
2014-02-01|0.2|6800
] (delimiter is '|');
```

另请参见：

- *NPV - 图表函数 (page 234)*
- *Aggr - 图表函数 (page 373)*

统计聚合函数

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

数据加载脚本中的统计聚合函数

以下统计聚合函数可用于脚本中。

Avg

Avg() 用于查找迭代于 **group by** 子句定义的大量记录的表达式中聚合数据的平均值。

```
Avg  ([distinct] expression)
```

Correl

Correl() 用于返回聚合相关系数，以揭示迭代于 **group by** 子句定义的大量记录上的 x-expression 和 y-expression 的成对数值表示的现金流明细表(不必为周期性的)。

```
Correl (x-expression, y-expression)
```

Fractile

Fractile() 用于查找与迭代于 **group by** 子句定义的大量记录的表达式中聚合数据的包含性分位数(位数)对应的值。

```
Fractile (expression, fractile)
```

FractileExc

FractileExc() 用于查找与迭代于 **group by** 子句定义的大量记录的表达式中聚合数据的排除性分位数(位数)对应的值。

```
FractileExc (expression, fractile)
```

Kurtosis

Kurtosis() 用于返回迭代于 **group by** 子句定义的大量记录的表达式中的数据峰度。

```
Kurtosis  ([distinct ] expression )
```

LINEST_B

LINEST_B() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 b 值(y 截距)，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_B (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_df

LINEST_DF() 用于返回由方程式 y=mx+b 定义的线性回归的聚合自由度，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_DF  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_f

此脚本函数用于返回由方程式 y=mx+b 定义的线性回归的聚合 F 统计量 ($r^2/(1-r^2)$)，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_F (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_m

LINEST_M() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 m 值（斜率），以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_M  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_r2

LINEST_R2() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 r^2 值（确定系数），以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_R2 (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_seb

LINEST_SEB() 用于返回由方程式 y=mx+b 定义的线性回归的 b 值标准误差，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_SEB  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_sem

LINEST_SEM() 用于返回由方程式 y=mx+b 定义的线性回归的 m 值标准误差，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_SEM  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_sey

LINEST_SEY() 用于返回由方程式 y=mx+b 定义的线性回归的 y 估计的标准误差，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_SEY  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_ssreg

LINEST_SSREG() 用于返回由方程式 y=mx+b 定义的线性回归的聚合回归平方和，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_SSREG  (y-expression, x-expression [, y0 [, x0 ]])
```

Linest_ssresid

LINEST_SSRESID() 用于返回由方程式 y=mx+b 定义的线性回归的聚合剩余平方和，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
LINEST_SSRESID  (y-expression, x-expression [, y0 [, x0 ]])
```

Median

Median() 用于返回迭代于 **group by** 子句定义的大量记录的表达式中的聚合中间值。

```
Median (expression)
```

Skew

Skew() 用于返回迭代于 **group by** 子句定义的许多记录的表达式的偏度。

```
Skew ([ distinct] expression)
```

Stdev

Stdev() 用于返回迭代于 **group by** 子句定义的大量记录的表达式中的标准偏差值。

```
Stdev ([distinct] expression)
```

Sterr

Sterr() 用于返回聚合标准误差 (stdev/sqrt(n)), 以获得表达式通过由 **group by** 子句定义的许多记录迭代表示的一系列值。

```
Sterr  ([distinct] expression)
```

STEYX

STEYX() 用于返回回归中每个 x 值的估算 y 值的聚合标准误差, 以获得通过由 **group by** 子句定义的许多记录迭代的 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

```
STEYX  (y-expression, x-expression)
```

图表表达式中的统计聚合函数

以下统计聚合函数可用于图表中。

Avg

Avg() 用于返回在图表维度上迭代的表达式或字段的聚合平均值。

```
Avg - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

Correl

Correl() 用于返回两个数据集的聚合相关系数。相关函数是数据集之间关系的度量, 用于聚合通过图表维度迭代的值对 (x,y)。

```
Correl - 图表函数({[SetExpression] [TOTAL [<fld {, fld}>]]} value1, value2 )
```

Fractile

Fractile() 用于查找与通过图表维度迭代的表达式指定的范围中聚合数据的包容性分位数(位数)对应的值。

```
Fractile - 图表函数({[SetExpression] [TOTAL [<fld {, fld}>]]} expr, fraction)
```

FractileExc

FractileExc() 用于查找与通过图表维度迭代的表达式指定的范围中聚合数据的排除性分位数(位数)对应的值。

```
FractileExc - 图表函数({[SetExpression] [TOTAL [<fld {, fld}>]]} expr,
fraction)
```

Kurtosis

Kurtosis() 用于查找通过图表维度迭代的表达式或字段的数据聚合范围的峰度。

```
Kurtosis - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

LINEST_b

LINEST_B() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 b 值(y 轴截距),以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式指定表达式中成对数值表示的一系列坐标。

```
LINEST_R2 - 图表函数({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_value
[, y0_const[, x0_const]])
```

LINEST_df

LINEST_DF() 用于返回由方程式 y=mx+b 定义的线性回归的聚合自由度,以获得在图表维度上迭代的 **x_value** 和 **y_value** 指定表达式中成对数值表示的一系列坐标。

```
LINEST_DF - 图表函数({[SetExpression] [TOTAL [<fld{, fld}>]]} y_value, x_value
[, y0_const [, x0_const]])
```

LINEST_f

LINEST_F() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 F 统计量 (r2/(1-r2)),以获得在图表维度上迭代的 **x_value** 和 **y_value** 指定表达式中成对数值表示的一系列坐标。

```
LINEST_F - 图表函数({[SetExpression] [TOTAL[<fld{, fld}>]]} y_value, x_value
[, y0_const [, x0_const]])
```

LINEST_m

LINEST_M() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 m 值(斜率),以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

```
LINEST_M - 图表函数({[SetExpression] [TOTAL[<fld{, fld}>]]} y_value, x_value
[, y0_const [, x0_const]])
```

LINEST_r2

LINEST_R2() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 r2 值(确定系数),以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

```
LINEST_R2 - 图表函数({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_value
[, y0_const[, x0_const]])
```

LINEST_seb

LINEST_SEB() 用于返回由方程式 y=mx+b 定义的线性回归的 b 值的聚合标准误差,以获得在图表维度上迭代的 **x_value** 和 **y_value** 方程式中成对数值表示的一系列坐标。

```
LINEST_SEB - 图表函数({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_
value[, y0_const[, x0_const]])
```

LINEST_sem

LINEST_SEM() 用于返回由方程式 y=mx+b 定义的线性回归的 m 值的聚合标准误差,以获得在图表维度上迭代的 **x_value** 和 **y_value** 方程式中成对数值表示的一系列坐标。

```
LINEST_SEM - 图表函数([{set_expression}][ distinct ] [total [<fld {,fld}>] ]
y-expression, x-expression [, y0 [, x0 ]] )
```

LINEST_sey

LINEST_SEY() 用于返回由方程式 y=mx+b 定义的线性回归的 y 估计的标准误差，以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

```
LINEST_SEY - 图表函数({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]])
```

LINEST_ssreg

LINEST_SSREG() 用于返回由方程式 y=mx+b 定义的线性回归的聚合回归平方和，以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

```
LINEST_SSREG - 图表函数({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]])
```

LINEST_ssresid

LINEST_SSRESID() 用于返回由方程式 y=mx+b 定义的线性回归的聚合剩余平方和，以获得在图表维度上迭代的 **x_value** 和 **y_value** 指定表达式中成对数值表示的一系列坐标。

```
LINEST_SSRESID - 图表函数LINEST_SSRESID() 用于返回由方程式 y=mx+b 定义的线性回归的聚合剩余平方和，以获得在图表维度上迭代的 x_value 和 y_value 指定表达式中成对数值表示的一系列坐标。 LINEST_SSRESID([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])数字 参数参数说明y_value表达式或字段要度量的 y 值的范围。x_value表达式或字段要度量的 x 值的范围。y0, x0可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。 SetExpression聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。 DISTINCT如果在函数参数前出现单词 DISTINCT，则将忽略计算该函数参数生成的副本。 TOTAL如果在函数参数前面出现单词 TOTAL，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。通过使用 TOTAL [<fld {.fld}>](其中 TOTAL 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 TOTAL 限定符。 有关高级嵌套聚合函数的更多信息，请结合指定维度使用 Aggr 高级函数。 数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。 An example of how to use linest functionsavg ({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]])
```

Median

Median() 用于返回在图表维度上迭代的表达式的聚合值范围的中间值。

```
Median - 图表函数({[SetExpression] [TOTAL [<fld{, fld}>]]} expr)
```

MutualInfo

MutualInfo 计算 **Aggr()** 中两个字段之间或聚合值之间的互信息(MI)。

```
MutualInfo - 图表函数 (page 281)({[SetExpression] [DISTINCT] [TOTAL target, driver [, datatype [, breakdownbyvalue [, samplesize ]]])
```

Skew

Skew() 用于返回在图表维度上迭代的表达式或字段的聚合偏度。

```
Skew - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld{ ,fld}>]]} expr)
```

Stdev

Stdev() 用于查找通过图表维度迭代的表达式或字段的数据聚合范围的标准偏差。

```
Stdev - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

Sterr

Sterr() 用于查找在通过图表维度迭代的表达式中聚合的值系列的平均值的标准误差 (stdev/sqrt (n))。

```
Sterr - 图表函数({[SetExpression] [DISTINCT] [TOTAL[<fld{, fld}>]]} expr)
```

STEYX

STEYX() 用于返回聚合标准误差，当为线性回归的每个 x 值预测 y 值时，该方程式由 **y_value** 和 **x_value** 指定表达式中成对数值表示的一系列坐标。

```
STEYX - 图表函数({[SetExpression] [TOTAL [<fld{, fld}>]]} y_value, x_value)
```

Avg

Avg() 用于查找迭代于 **group by** 子句定义的大量记录的表达式中聚合数据的平均值。

语法：

```
Avg([DISTINCT] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
DISTINCT	如果在表达式前面出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果
Temp: crosstable (Month, Sales) load * inline [Customer|Jan|Feb|Mar||Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec Astrida|46|60|70|13|78|20|45|65|78|12|78|22 Betacab|65|56|22|79|12|56|45|24|32|78|55|15 Canutility|77|68|34|91|24|68|57|36|44|90|67|27 Divadip|36|44|90|67|27|57|68|47|90|80|94] (delimiter is '|'); Avg1: LOAD Customer, Avg(Sales) as MyAverageSalesByCustomer Resident Temp Group By Customer;	Customer MyAverageSalesByCustomer Astrida 48.916667 Betacab 44.916667 Canutility 56.916667 Divadip 63.083333 这可以通过创建包括以下度量的表格在工作表中进行检查： Sum(Sales)/12
前提是 **Temp** 表格像之前的示例一样加载： LOAD Customer,Avg(DISTINCT Sales) as MyAvgSalesDistinct Resident Temp Group By Customer;	Customer MyAverageSalesByCustomer Astrida 43.1 Betacab 43.909091 Canutility 55.909091 Divadip 61 只会对特殊值进行计数。用总数除以非重复值的个数。

Avg - 图表函数

Avg() 用于返回在图表维度上迭代的表达式或字段的聚合平均值。

语法：

```
Avg([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

示例和结果：

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

函数示例

示例	结果
`Avg(Sales)`	对于包含维度 `Customer`和度量 `Avg([Sales])` 的表格，如果显示**合计**，则结果为 2566。
`Avg([TOTAL (Sales))`	对于 `Customer` 的全部值，结果为 53.458333，因为 TOTAL 限定符意味着忽略维度。
`Avg(DISTINCT (Sales))`	合计为 51.862069，因为使用 Distinct 限定符意味着仅评估每个 `Sales Customer` 中的唯一值。

示例中所使用的数据：

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
```

```
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

另请参见：

- *Aggr - 图表函数 (page 373)*

Correl

Correl() 用于返回聚合相关系数，以揭示迭代于 **group by** 子句定义的大量记录上的 x-expression 和 y-expression 的成对数值表示的现金流明细表（不必为周期性的）。

语法：

```
Correl(value1, value2)
```

返回数据类型： 数字

参数：

参数

参数	说明
value1, value2	表达式或字段，其中包含两个要度量相关系数的样本集合。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果
Salary: Load *, 1 as Grp; LOAD * inline ["Employee name"|Gender|Age|Salary Aiden Charles|Male|20|25000 Brenda Davies|Male|25|32000 Charlotte Edberg|Female|45|56000 Daroush Ferrara|Male|31|29000 Eunice Goldblum|Female|31|32000 Freddy Halvorsen|Male|25|26000 Gauri Indu|Female|36|46000 Harry Jones|Male|38|40000 Ian Underwood|Male|40|45000 Jackie Kingsley|Female|23|28000] (delimiter is '|'); Correl1: LOAD Grp, Correl(Age,Salary) as Correl_Salary Resident Salary Group By Grp;	在包含维度 Correl_Salary 的表格中，在数据加载脚本中计算的 Correl() 结果将显示：0.9270611

Correl - 图表函数

Correl() 用于返回两个数据集的聚合相关系数。相关函数是数据集之间关系的度量，用于聚合通过图表维度迭代的值对 (x,y)。

语法：

```
Correl([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] value1, value2 )
```

返回数据类型： 数字

参数：

参数

参数	说明
value1, value2	表达式或字段，其中包含两个要度量相关系数的样本集合。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

示例和结果：

函数示例

示例	结果
`Correl(Age, Salary)`	对于包含维度 `Employee name`以及度量 `Correl(Age, Salary)` 的表格，结果为 0.9270611。仅显示合计单元格的结果。
`Correl(TOTAL Age, Salary))`	0.927. 此结果和随后的结果显示为三个小数位，以便增强可读性。 如果创建具有维度 Gender 的筛选器窗格，并在其中做出选择，则在选择 Female 时，您将看到结果 0.951；在选择 Male 时，您将看到结果 0.939。这是因为此选择项排除了不属于 Gender 的其他值的所有结果。
`Correl({1} TOTAL Age, Salary))`	0.927. 与选择项无关。这是因为集合表达式 {1} 忽略了所有选择项和维度。
`Correl(TOTAL <Gender> Age, Salary))`	在合计单元格中，结果为 0.927，对于 Male 的全部值，结果为 0.939，对于 Female 的全部值，结果为 0.951。此结果相当于在筛选器窗格中基于 Gender 做出选择的结果。

示例中所使用的数据：

```
Salary:
LOAD * inline [
"Employee name"|Gender|Age|Salary
Aiden Charles|Male|20|25000
Brenda Davies|Male|25|32000
Charlotte Edberg|Female|45|56000
Daroush Ferrara|Male|31|29000
Eunice Goldblum|Female|31|32000
Freddy Halvorsen|Male|25|26000
Gauri Indu|Female|36|46000
Harry Jones|Male|38|40000
Ian Underwood|Male|40|45000
Jackie Kingsley|Female|23|28000
] (delimiter is '|');
```

另请参见：

- *Aggr - 图表函数 (page 373)*
- *Avg - 图表函数 (page 246)*
- *RangeCorrel (page 647)*

Fractile

Fractile() 用于查找与迭代于 **group by** 子句定义的大量记录的表达式中聚合数据的包含性分位数(位数)对应的值。

您可使用 FractileExc (page 254) 来计算排除性分位数

语法:

```
Fractile(expr, fraction)
```

返回数据类型: 数字

函数返回与 `rank = fraction * (N-1) + 1` 定义的排名对应的值,其中 `N` 是 `expr` 中的值数目。如果排名是非整数数字,则会在两个最接近的值之间进行插值。

参数:

参数

参数	说明
expr	包含计算分位数时要使用的数据的表达式或字段。
fraction	可以计算与分位数(表示为分数的位数)对应的介于 0 和 1 之间的数字。

示例和结果:

将示例脚本添加到应用程序并运行。要查看结果,将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果
Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Fractile1: LOAD Type, Fractile(Value,0.75) as MyFractile Resident Table1 Group By Type;	在包含维度 Type 和 MyFractile 的表格中，在数据加载脚本中计算的 Fractile() 结果为： Type MyFractile Comparison 27.5 Observation 36

Fractile - 图表函数

Fractile() 用于查找与通过图表维度迭代的表达式指定的范围中聚合数据的包容性分位数(位数)对应的值。

您可使用 FractileExc - 图表函数 (page 255) 来计算排除性分位数

语法：

Fractile([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr, fraction**)**

返回数据类型： 数字

函数返回与 rank = fraction * (N-1) + 1 定义的排名对应的值，其中 N 是 expr 中的值数目。如果排名是非整数数字，则会在两个最接近的值之间进行插值。

参数：

参数

参数	说明
expr	包含计算分位数时要使用的数据的表达式或字段。
fraction	可以计算与分位数(表示为分数的位数)对应的介于 0 和 1 之间的数字。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

示例和结果：

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

函数示例

示例	结果
`Fractile(Sales, 0.75)`	对于包含维度 `Customer`和度量 `Fractile([Sales])` 的表格，如果显示**合计**，则结果为 71.75。此结果是 75% 的值所属的 `Sales` 值分布中的点。
`Fractile(TOTAL Sales, 0.75))`	对于 `Customer` 的全部值，结果为 71.75，因为 TOTAL 限定符意味着忽略维度。
`Fractile (DISTINCT Sales, 0.75)`	合计为 70，因为使用 DISTINCT 限定符意味着仅评估每个 `Sales Customer` 中的唯一值。

示例中所使用的数据：

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

另请参见：

- *Aggr - 图表函数 (page 373)*

FractileExc

FractileExc() 用于查找与迭代于 **group by** 子句定义的大量记录的表达式中聚合数据的排除性分位数(位数)对应的值。

> *您可使用 Fractile (page 251) 来计算包含性分位数*

语法：

```
FractileExc(expr, fraction)
```

返回数据类型： 数字

函数返回与 rank = fraction * (N+1) 定义的排名对应的值，其中 N 是 expr 中的值数目。如果排名是非整数数字，则会在两个最接近的值之间进行插值。

参数：

参数

参数	说明
expr	包含计算分位数时要使用的数据的表达式或字段。
fraction	可以计算与分位数(表示为分数的位数)对应的介于 0 和 1 之间的数字。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果
Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Fractile1: LOAD Type, FractileExc(Value,0.75) as MyFractile Resident Table1 Group By Type;	在包含维度 Type 和 MyFractile 的表格中，在数据加载脚本中计算的 FractileExc() 结果为： Type MyFractile Comparison 28.5 Observation 38

FractileExc - 图表函数

FractileExc() 用于查找与通过图表维度迭代的表达式指定的范围中聚合数据的排除性分位数(位数)对应的值。

您可使用 Fractile - 图表函数 (page 252) 来计算包含性分位数

语法：

```
FractileExc([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr,
fraction)
```

返回数据类型：数字

函数返回与 rank = fraction * (N+1) 定义的排名对应的值，其中 N 是 expr 中的值数目。如果排名是非整数数字，则会在两个最接近的值之间进行插值。

参数：

参数

参数	说明
expr	包含计算分位数时要使用的数据的表达式或字段。
fraction	可以计算与分位数(表示为分数的位数)对应的介于 0 和 1 之间的数字。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

示例和结果：

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

函数示例

示例	结果
FractileExc (Sales, 0.75)	对于包含维度 Customer和度量 FractileExc([Sales]) 的表格，如果显示**合计**，则结果为 75.25。此结果是 75% 的值所属的 Sales 值分布中的点。
FractileExc(TOTAL Sales, 0.75))	对于 Customer 的全部值，结果为 75.25，因为 TOTAL 限定符意味着忽略维度。
FractileExc (DISTINCT Sales, 0.75)	合计为 73.50，因为使用 DISTINCT 限定符意味着仅评估每个 Customer 的 Sales 中的唯一值。

示例中所使用的数据：

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

另请参见：

Aggr - 图表函数 (page 373)

Kurtosis

Kurtosis() 用于返回迭代于 **group by** 子句定义的大量记录的表达式中的数据峰度。

语法：

```
Kurtosis([distinct ] expr )
```

返回数据类型：数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
distinct	如果在表达式前面出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果
Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Kurtosis1: LOAD Type, Kurtosis(Value) as MyKurtosis1, Kurtosis(DISTINCT Value) as MyKurtosis2 Resident Table1 Group By Type;	在包含维度 Type、MyKurtosis1 和 MyKurtosis2 的表格中，在数据加载脚本中计算的 Kurtosis() 结果为： Type MyKurtosis1 MyKurtosis2 Comparison -1.1612957 -1.4982366 Observation -1.1148768 -0.93540144

Kurtosis - 图表函数

Kurtosis() 用于查找通过图表维度迭代的表达式或字段的数据聚合范围的峰度。

语法：

```
Kurtosis([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

示例和结果：

Example table

Type	Value																			
Comparison	2	27	38	31	1	19	1	34	3	1	2	3	2	1	2	1	3	29	37	2
Observation	35	40	12	15	21	14	46	10	28	48	16	30	32	48	31	22	12	39	19	25

函数示例

示例	结果
Kurtosis (Value)	对于包含维度 Type和度量 Kurtosis(Value) 的表格，如果显示表格的**合计**，并且数字格式设置为 3 个有效数字，则结果为 1.252。对于 Comparison，结果为 1.161，对于 Observation，结果为 1.115。
Kurtosis (TOTAL Value))	对于 Type 的全部值，结果为 1.252，因为 TOTAL 限定符意味着忽略维度。

示例中所使用的数据：

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

另请参见：

Avg - 图表函数 (page 246)

LINEST_B

LINEST_B() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 b 值（y 截距），以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_B (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_B - 图表函数

LINEST_B() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 b 值（y 轴截距），以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式指定表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_B([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value
[, y0_const [ , x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

参数	说明
y0_const, x0_const	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_DF

LINEST_DF() 用于返回由方程式 y=mx+b 定义的线性回归的聚合自由度，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_DF (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_DF - 图表函数

LINEST_DF() 用于返回由方程式 y=mx+b 定义的线性回归的聚合自由度，以获得在图表维度上迭代的 **x_value** 和 **y_value** 指定表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_DF([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

参数	说明
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_F

此脚本函数用于返回由方程式 y=mx+b 定义的线性回归的聚合 F 统计量 ($r^2/(1-r^2)$)，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_F (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_F - 图表函数

LINEST_F() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 F 统计量 (r2/(1-r2))，以获得在图表维度上迭代的 **x_value** 和 **y_value** 指定表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_F([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

参数	说明
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_M

LINEST_M() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 m 值(斜率)，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_M (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_M - 图表函数

LINEST_M() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 m 值（斜率），以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_M([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

参数	说明
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_R2

LINEST_R2() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 r^2 值(确定系数)，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_R2 (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_R2 - 图表函数

LINEST_R2() 用于返回由方程式 y=mx+b 定义的线性回归的聚合 r2 值（确定系数），以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_R2([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

参数	说明
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_SEB

LINEST_SEB() 用于返回由方程式 y=mx+b 定义的线性回归的 b 值标准误差，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_SEB (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_SEB - 图表函数

LINEST_SEB() 用于返回由方程式 y=mx+b 定义的线性回归的 b 值的聚合标准误差，以获得在图表维度上迭代的 **x_value** 和 **y_value** 方程式中成对数值表示的一系列坐标。

语法：

```
LINEST_SEB([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

参数	说明
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_SEM

LINEST_SEM() 用于返回由方程式 y=mx+b 定义的线性回归的 m 值标准误差，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_SEM (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型： 数字

参数：

参数	说明
y_value	表达式或字段要度量的 y 值的范围。

参数	说明
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_SEM - 图表函数

LINEST_SEM() 用于返回由方程式 y=mx+b 定义的线性回归的 m 值的聚合标准误差，以获得在图表维度上迭代的 **x_value** 和 **y_value** 方程式中成对数值表示的一系列坐标。

语法：

```
LINEST_SEM([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

返回数据类型： 数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_SEY

LINEST_SEY() 用于返回由方程式 y=mx+b 定义的线性回归的 y 估计的标准误差，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_SEY (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型： 数字

参数：

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*

LINEST_SEY - 图表函数

LINEST_SEY() 用于返回由方程式 y=mx+b 定义的线性回归的 y 估计的标准误差，以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_SEY([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

返回数据类型： 数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_SSREG

LINEST_SSREG() 用于返回由方程式 y=mx+b 定义的线性回归的聚合回归平方和，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_SSREG (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型： 数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*

LINEST_SSREG - 图表函数

LINEST_SSREG() 用于返回由方程式 y=mx+b 定义的线性回归的聚合回归平方和，以获得在图表维度上迭代的 **x_value** 和 **y_value** 表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_SSREG([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_
value[, y0_const[, x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

LINEST_SSRESID

LINEST_SSRESID() 用于返回由方程式 y=mx+b 定义的线性回归的聚合剩余平方和，以获得通过由 **group by** 子句定义的许多记录迭代 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法：

```
LINEST_SSRESID (y_value, x_value[, y0 [, x0 ]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y(0), x(0)	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。 除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

如何使用 linest 函数的示例 (page 298)

LINEST_SSRESID - 图表函数

LINEST_SSRESID() 用于返回由方程式 y=mx+b 定义的线性回归的聚合剩余平方和，以获得在图表维度上迭代的 **x_value** 和 **y_value** 指定表达式中成对数值表示的一系列坐标。

语法：

```
LINEST_SSRESID([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value,
x_value[, y0_const[, x0_const]])
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

参数	说明
y0, x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。  *除非同时声明 y0 和 x0，否则此函数至少需要计算两个有效数据对。如果声明 y0 和 x0，则此函数需要计算单个数据对。*
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

另请参见：

- *如何使用 linest 函数的示例 (page 298)*
- *Avg - 图表函数 (page 246)*

Median

Median() 用于返回迭代于 **group by** 子句定义的大量记录的表达式中的聚合中间值。

语法：

```
Median (expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。

示例:使用中间值的脚本表达式

示例 - 脚本表达式

加载脚本

在本示例的数据加载编辑器中加载以下内联数据和脚本表达式。

```
Table 1: Load RecNo() as RowNo, Letter, Number Inline [Letter, Number A,1 A,3 A,4 A,9 B,2 B,8
B,9];                                   Median: LOAD Letter, Median(Number) as MyMedian Resident Table1 Grou
```

创建可视化

在 Qlik Sense 工作表中创建表可视化,以 **Letter** 和 **MyMedian** 为维度。

结果

Letter	MyMedian
A	3.5
B	8

解释

当数字按从最小到最大的顺序排序时,中位数被视为"中间"数字。如果数据集的值为偶数,则函数返回两个中间值的平均值。在本例中,为 **A** 和 **B** 的每组值计算中值,分别为 3.5 和 8。

Median - 图表函数

Median() 用于返回在图表维度上迭代的表达式的聚合值范围的中间值。

语法:

```
Median([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

返回数据类型: 数字

参数:

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**,则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**,则计算给出当前选择项的所有可能值,而不只是属于当前维度值的那些值,即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表),您可以创建总可能值的子集。

限制:

聚合函数的参数不能包含其他聚合函数,除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息,请结合指定维度使用 **Aggr** 高级函数。

示例:使用中间值的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入,以创建以下图表表达式示例。

```
Load RecNo() as RowNo, Letter, Number Inline [Letter, Number A,1 A,3 A,4 A,9 B,2 B,8 B,9];
```

创建可视化

在 Qlik Sense 工作表中创建表可视化,以 **Letter** 为维度。

图表表达式

在表格中添加以下表达式作为度量:

```
Median(Number)
```

结果

Letter	Median(Number)
Totals	4
A	3.5
B	8

解释

当数字按从最小到最大的顺序排序时,中位数被视为"中间"数字。如果数据集的值为偶数,则函数返回两个中间值的平均值。在本例中,为 **A** 和 **B** 的每组值计算中值,分别为 3.5 和 8。

总计的中位数由所有值计算得出,等于 4。

另请参见:

- *Avg - 图表函数 (page 246)*

MutualInfo - 图表函数

MutualInfo 计算 **Aggr()** 中两个字段之间或聚合值之间的互信息(MI)。

MutualInfo 返回两个数据集的聚合互信息。这允许在字段和潜在驱动因素之间进行关键驱动因素分析。互信息是数据集之间关系的一种度量，并为在图表维度上迭代的 (x,y) 对值进行聚合。互信息在 0 和 1 之间测量，可以格式化为百分位值。**MutualInfo** 由选择或集合表达式定义。

MutualInfo 允许不同种类的 MI 分析：

- 对范围 MI：计算驱动程序字段和目标字段之间的 MI。
- 驱动因素按值分解：MI 是在驱动因素字段和目标字段中的单个字段值之间计算的。
- 特性选择：使用网格图中 **MutualInfo** 的生成一个矩阵，其中所有字段基于 MI 相互比较。

MutualInfo 不一定表示共享互信息的字段之间的因果关系。两个字段可以共享相互的信息，但对彼此来说可能不相等。例如，当比较冰淇淋销量和室外温度时，**MutualInfo** 会显示两者之间的互信息。它不会指出是室外温度推动冰淇淋销售（这是可能的），还是冰淇淋销售推动室外温度（这是不可能的）。

在计算互信息时，关联会影响来自不同表的字段的值之间的对应关系和频率。

相同字段或选择的返回值可能略有不同。这是由于每次 **MutualInfo** 调用都是对随机选择的样本进行操作，并且 **MutualInfo** 算法本身具有随机性。

MutualInfo 可应用于 **Aggr()** 函数。

语法：

```
MutualInfo({SetExpression}] [DISTINCT] [TOTAL] field1, field2 , datatype [,
breakdownbyvalue [, samplesize ]])
```

返回数据类型： 数字

参数：

参数

参数	说明
field1, field2	表达式或字段，其中包含两个要度量交互信息的样本集合。
datatype	数据类型包含在目标和驱动因素中， 1 或 'dd' 用于 discrete:discrete 2 或 'cc' 用于 continuous:continuous 3 或 'cd' 用于 continuous:discrete 4 或 'dc' 用于 discrete:continuous 数据类型不区分大小写。

参数	说明
breakdownbyvalue	与驱动因素中的值相对应的静态值。如果提供，计算将计算该值的 MI 贡献。您可使用 **ValueList()** 或 **ValueLoop()**。如果添加了 **Null()**，则计算将为驱动因素中的所有值计算整个 MI。 按值细分要求驱动因素包含离散数据。
samplesize	要从目标和驱动程序中采样的值的数目。采样是随机的。**MutualInfo** 需要 80 的最小样本大小。默认设置下，**MutualInfo** 最多仅可采样 10,000 个数据对，因为 **MutualInfo** 可以为资源密集型。您可在样本大小中指定更大数目的数据对。如果 **MutualInfo** 超时，请减小样本大小。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

函数示例

示例	结果
`mutualinfo(Age, Salary, 1)`	对于包含维度 `Employee name` 和度量 `mutualinfo(Age, Salary, 1)` 的表格，结果为 0.99820986。仅显示合计单元格的结果。
`mutualinfo (TOTAL Age, Salary, 1, null (), 81)`	如果创建具有维度 Gender 的筛选器窗格，并在其中做出选择，则在选择 Female 时，您将看到结果 0.99805677；在选择 Male 时，您将看到结果 0.99847373。这是因为此选择项排除了不属于 Gender 的其他值的所有结果。
`mutualinfo (TOTAL Age, Gender, 1, ValueLoop (25,35))`	0.68196996. 从 Gender 选择任何值将把此更改为 0。
`mutualinfo({1} TOTAL Age, Salary, 1, null ())`	0.99820986. 这与选择项无关。集合表达式 {1} 忽略了所有选择项和维度。

示例中所使用的数据:

```
Salary:

LOAD * inline [

"Employee name"|Age|Gender|Salary

Aiden Charles|20|Male|25000

Ann Lindquist|69|Female|58000

Anna Johansen|37|Female|36000

Anna Karlsson|42|Female|23000

Antonio Garcia|20|Male|61000

Benjamin Smith|42|Male|27000

Bill Yang|49|Male|50000

Binh Protzmann|69|Male|21000

Bob Park|51|Male|54000

Brenda Davies|25|Male|32000

Celine Gagnon|48|Female|38000

Cezar Sandu|50|Male|46000

Charles Ingvar Jönsson|27|Male|58000

Charlotte Edberg|45|Female|56000

Cindy Lynn|69|Female|28000

Clark Wayne|63|Male|31000

Daroush Ferrara|31|Male|29000

David Cooper|37|Male|64000

David Leg|58|Male|57000

Eunice Goldblum|31|Female|32000

Freddy Halvorsen|25|Male|26000

Gauri Indu|36|Female|46000

George van Zaant|59|Male|47000
```

```
Glenn Brown|58|Male|40000

Harry Jones|38|Male|40000

Helen Brolin|52|Female|66000

Hiroshi Ito|24|Male|42000

Ian Underwood|40|Male|45000

Ingrid Hendrix|63|Female|27000

Ira Baumel|39|Female|39000

Jackie Kingsley|23|Female|28000

Jennica Williams|36|Female|48000

Jerry Tessel|31|Male|57000

Jim Bond|50|Male|58000

Joan Callins|60|Female|65000

Joan Cleaves|25|Female|61000

Joe Cheng|61|Male|41000

John Doe|36|Male|59000

John Lemon|43|Male|21000

Karen Helmkey|54|Female|25000

Karl Berger|38|Male|68000

Karl Straubaum|30|Male|40000

Kaya Alpan|32|Female|60000

Kenneth Finley|21|Male|25000

Leif Shine|63|Male|70000

Lennart Skoglund|63|Male|24000

Leona Korhonen|46|Female|50000

Lina André|50|Female|65000

Louis Presley|29|Male|36000

Luke Langston|50|Male|63000
```

```
Marcus Salvatori|31|Male|46000

Marie Simon|57|Female|23000

Mario Rossi|39|Male|62000

Markus Danzig|26|Male|48000

Michael Carlen|21|Male|45000

Michelle Tyson|44|Female|69000

Mike Ashkenaz|45|Male|68000

Miro Ito|40|Male|39000

Nina Mihn|62|Female|57000

Olivia Nguyen|35|Female|51000

Olivier Simenon|44|Male|31000

Östen Ärlig|68|Male|57000

Pamala Garcia|69|Female|29000

Paolo Romano|34|Male|45000

Pat Taylor|67|Female|69000

Paul Dupont|34|Male|38000

Peter Smith|56|Male|53000

Pierre Clouseau|21|Male|37000

Preben Jørgensen|35|Male|38000

Rey Jones|65|Female|20000

Ricardo Gucci|55|Male|65000

Richard Ranieri|30|Male|64000

Rob Carsson|46|Male|54000

Rolf Wesenlund|25|Male|51000

Ronaldo Costa|64|Male|39000

Sabrina Richards|57|Female|40000

Sato Hiromu|35|Male|21000
```

```
Sehoon Daw|57|Male|24000

Stefan Lind|67|Male|35000

Steve Cioazzi|58|Male|23000

Sunil Gupta|45|Male|40000

Sven Svensson|45|Male|55000

Tom Lindwall|46|Male|24000

Tomas Nilsson|27|Male|22000

Trinity Rizzo|52|Female|48000

Vanessa Lambert|54|Female|27000

] (delimiter is '|');
```

Skew

Skew() 用于返回迭代于 **group by** 子句定义的许多记录的表达式的偏度。

语法：

```
Skew([ distinct] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
DISTINCT	如果在表达式前面出现单词 **distinct**, 则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，构建包括 Type 和 *MySkew* 的垂直表作为维度。

结果数据

示例	结果
Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Skew1: LOAD Type, Skew(Value) as MySkew Resident Table1 Group By Type;	Skew() 计算的结果是： • Type 是 MySkew • Comparison 是 0.86414768 • Observation 是 0.32625351

Skew - 图表函数

Skew() 用于返回在图表维度上迭代的表达式或字段的聚合偏度。

语法：

```
Skew([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

示例和结果：

将示例脚本添加到应用程序并运行。然后使用 `Type` 作为维度并使用 `Skew(Value)` 作为度量以构建垂直表。

`Totals` 应在表格的属性中启用。

示例	结果

`Table1:`
`crosstable LOAD recno()`
`as ID, * inline [`
`Observation|Comparison`
`35|2`
`40|27`
`12|38`
`15|31`
`21|1`
`14|19`
`46|1`
`10|34`
`28|3`
`48|1`
`16|2`
`30|3`
`32|2`
`48|1`
`31|2`
`22|1`
`12|3`
`39|29`
`19|37`
`25|2 ] (delimiter is`
`'|');`
	Skew(Value) 计算的结果是：
	
	• `Total`是 0.23522195
	• `Comparison` 是 0.86414768
	• `Observation` 是 0.32625351

另请参见：

- *Avg - 图表函数 (page 246)*

Stdev

Stdev() 用于返回迭代于 **group by** 子句定义的大量记录的表达式中的标准偏差值。

语法：

```
Stdev([distinct] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
distinct	如果在表达式前面出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，构建包括 `Type` 和 `MyStdev` 的垂直表作为维度。

结果数据

示例	结果
Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Stdev1: LOAD Type, Stdev(Value) as MyStdev Resident Table1 Group By Type;	Stdev() 计算的结果是： • `Type` 是 `MyStdev` • `Comparison` 是 14.61245 • `Observation` 是 12.507997

Stdev - 图表函数

Stdev() 用于查找通过图表维度迭代的表达式或字段的数据聚合范围的标准偏差。

语法：

```
Stdev([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

返回数据类型： 数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

示例和结果：

将示例脚本添加到应用程序并运行。然后使用 `Type` 作为维度并使用 `Stdev(Value)` 作为度量以构建垂直表。

`Totals` 应在表格的属性中启用。

示例	结果
Stdev(Value) Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|');	Stdev(Value) 计算的结果是: • Total是 15.47529 • Comparison 是 14.61245 • Observation 是 12.507997

另请参见:

- *Avg - 图表函数 (page 246)*
- *STEYX - 图表函数 (page 296)*

Sterr

Sterr() 用于返回聚合标准误差 (stdev/sqrt(n)), 以获得表达式通过由 **group by** 子句定义的许多记录迭代表示的一系列值。

语法:

```
Sterr ([distinct] expr)
```

返回数据类型: 数字

参数:

参数

参数	说明
expr	表达式或字段包含要度量的数据。
distinct	如果在表达式前面出现单词 **distinct**, 则将忽略所有重复值。

限制：

文本值，NULL 值和缺失值都忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果
Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Sterr1: LOAD Type, Sterr(Value) as MySterr Resident Table1 Group By Type;	在包含维度 Type 和 MySterr 的表格中，在数据加载脚本中计算的 Sterr() 结果为： Type MySterr Comparison 3.2674431 Observation 2.7968733

Sterr - 图表函数

Sterr() 用于查找在通过图表维度迭代的表达式中聚合的值系列的平均值的标准误差 (stdev/sqrt (n))。

语法：

```
Sterr([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

返回数据类型：数字

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {,fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

文本值，NULL 值和缺失值都忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。然后使用 `Type` 作为维度并使用 `Sterr(Value)` 作为度量以构建垂直表。

`Totals` 应在表格的属性中启用。

示例	结果
Table1: crosstable LOAD recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|');	Sterr(Value) 计算的结果是: • Total是 2.4468583 • Comparison 是 3.2674431 • Observation 是 2.7968733

另请参见:

- *Avg - 图表函数 (page 246)*
- *STEYX - 图表函数 (page 296)*

STEYX

STEYX() 用于返回回归中每个 x 值的估算 y 值的聚合标准误差, 以获得通过由 **group by** 子句定义的许多记录迭代的 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

语法:

```
STEYX (y_value, x_value)
```

返回数据类型: 数字

参数:

参数

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。

限制：

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果
Trend: Load *, 1 as Grp; LOAD * inline [Month|KnownY|KnownX Jan|2|6 Feb|3|5 Mar|9|11 Apr|6|7 May|8|5 Jun|7|4 Jul|5|5 Aug|10|8 Sep|9|10 Oct|12|14 Nov|15|17 Dec|14|16] (delimiter is '|'); STEYX1: LOAD Grp, STEYX(KnownY, KnownX) as MySTEYX Resident Trend Group By Grp;	在包含维度 MySTEYX 的表格中，在数据加载脚本中计算的 STEYX() 结果为 2.0714764。

STEYX - 图表函数

STEYX() 用于返回聚合标准误差，当为线性回归的每个 x 值预测 y 值时，该方程式由 **y_value** 和 **x_value** 指定表达式中成对数值表示的一系列坐标。

语法：

```
STEYX([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value)
```

返回数据类型：数字

参数：

参数

参数	说明
y_value	表达式或字段，其中包含要度量的已知 y 值范围。
x_value	表达式或字段，其中包含要度量的已知 x 值范围。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

限制：

聚合函数的参数不能包含其他聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合指定维度使用 **Aggr** 高级函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

示例和结果：

将示例脚本添加到应用程序并运行。然后，构建包括 `KnownY` 和 `KnownX` 的垂直表作为维度，并构建包括 `Steyx(KnownY,KnownX)` 的垂直表作为度量。

`Totals` 应在表格的属性中启用。

示例	结果
Trend: LOAD * inline [Month|KnownY|KnownX Jan|2|6 Feb|3|5 Mar|9|11 Apr|6|7 May|8|5 Jun|7|4 Jul|5|5 Aug|10|8 Sep|9|10 Oct|12|14 Nov|15|17 Dec|14|16] (delimiter is '|');	STEYX(KnownY,KnownX) 计算的结果是 2.071(如果数字格式设置为 3 个小数位。)

另请参见:

- *Avg - 图表函数 (page 246)*
- *Sterr - 图表函数 (page 293)*

如何使用 linest 函数的示例

linest 函数用于查找与线性回归分析相关的值。本节介绍如何通过使用样本数据查找 Qlik Sense中可用的 linest 函数值来创建可视化内容。linest 所有函数均可用于数据加载脚本和图表表达式。

有关语法和参数的说明, 请参阅单独的 linest 图表函数和脚本函数主题。

示例中使用的数据和脚本表达式

在下面的 linest() 示例的数据加载编辑器中加载以下内联数据和脚本表达式。

```
T1: LOAD *, 1 as Grp; LOAD * inline [ X|Y 1|0 2|1 3|3 4|8 5|14 6|20 7|0 8|50 9|25 10|60 11|38 12|19 13|26 14|143 15|98 16|27 17|59 18|78 19|158 20|279 ] (delimiter is '|');                                R1: LOAD Grp, linest_B(Y,X) as Linest_B, linest_DF(Y,X) as Linest_DF, linest_F(Y,X) as Linest_F, linest_M(Y,X) as Linest_M, linest_R2(Y,X) as Linest_R2, linest_SEB(Y,X,1,1) as Linest_SEB, linest_SEM(Y,X) as Linest_SEM, linest_SEY(Y,X) as Linest_SEY, linest_SSREG(Y,X) as Linest_SSREG, linest_SSRESID(Y,X) as Linest_SSRESID resident T1 group by Grp;
```

示例 1: 使用 linest 的脚本表达式

示例: 脚本表达式

从数据加载脚本计算创建可视化

使用以下字段作为列在 Qlik Sense 工作表中创建表格可视化:

- Linest_B
- Linest_DF
- Linest_F
- Linest_M

- Linest_R2
- Linest_SEB
- Linest_SEM
- Linest_SEY
- Linest_SSREG
- Linest_SSRESID

结果

表格包含在数据加载脚本中执行 linest 计算的结果，如下所示：

结果表

Linest_B	Linest_DF	Linest_F	Linest_M	Linest_R2	Linest_SEB
-35.047	18	20.788	8.605	0.536	22.607

结果表

Linest_SEM	Linest_SEY	Linest_SSREG	Linest_SSRESID
1.887	48.666	49235.014	42631.186

示例 2: 使用 linest 的图表表达式

示例：图表表达式

使用以下字段作为维度在 Qlik Sense 工作表中创建表格可视化：

```
ValueList('Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_
SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID')
```

该表达式使用组合维度函数为具有 linest 函数名称的维度创建标签。您可以将标签更改为 **Linest functions** 以节省空间。

在表格中添加以下表达式作为度量：

```
Pick(Match(ValueList('Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_
SEB','Linest_SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID'),'Linest_b', 'Linest_
df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_SEM','Linest_SEY','Linest_
SSREG','Linest_SSRESID'),Linest_b(Y,X),Linest_df(Y,X),Linest_f(Y,X),Linest_m(Y,X),Linest_r2
(Y,X),Linest_SEB(Y,X,1,1),Linest_SEM(Y,X),Linest_SEY(Y,X),Linest_SSREG(Y,X),Linest_SSRESID
(Y,X) )
```

该表达式将根据组合维度中的相应名称显示每个 linest 函数的结果值。`Linest_b(Y,X)` 的结果显示在 **linest_b** 旁边，以此类推。

结果

结果表

Linest functions	Linest function results
Linest_b	-35.047
Linest_df	18

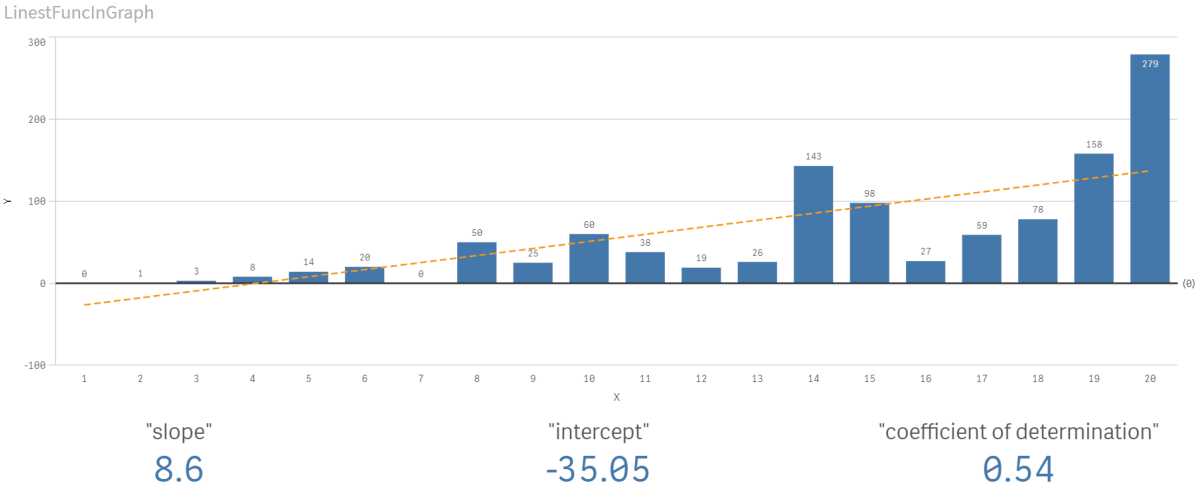
Linest functions	Linest function results
Linest_f	20.788
Linest_m	8.605
Linest_r2	0.536
Linest_SEB	22.607
Linest_SEM	1.887
Linest_SEY	48.666
Linest_SSREG	49235.014
Linest_SSRESID	42631.186

示例 3: 使用 linest 的图表表达式

示例：图表表达式

1. 在 Qlik Sense 工作表中创建条形图可视化，其中 **X** 为尺寸标注，**Y** 为度量。
2. 将线性趋势线添加到 Y 度量。
3. 添加 KPI 可视化到工作表。
    1. 添加 *slope* 作为 KPI 标签。
    2. 添加 `sum(Linest_M)` 作为 KPI 表达式。
4. 添加第二个 KPI 可视化到工作表。
    1. 添加 *intercept* 作为 KPI 标签。
    2. 添加 `Sum(Linest_B)` 作为 KPI 表达式。
5. 添加第三个 KPI 可视化到工作表。
    1. 添加 *确定系数* 作为 KPI 的标签。
    2. 添加 `Sum(Linest_R2)` 作为 KPI 表达式。

结果



解释

条形图显示 X 和 Y 数据的绘图。相关 linest() 函数为趋势线所依据的线性回归方程提供值，即 y = m * x + b。该方程使用“最小二乘”方法，通过返回描述最适合数据的直线的数组来计算直线（趋势线）。

KPI 显示 linest() 函数 **sum(Linest_M)** 用于斜率和 Y 截距的 **sum(Linest_B)**，这是线性回归方程中的变量，以及相应的确定系数聚合 R2 值。

统计检验函数

统计测试函数可用于数据加载脚本和图表表达式，但语法不同。

卡方检验函数

通常用于定性变量研究。该函数可用来将在单向频率表中观察到的频率与预期的频率进行比较，或者研究列联表中两个变量之间的连接。

T 检验函数

T 检验函数用于统计检查两个总体均值。双样本 T 检验检查两个样本是否不同，通常在两个正态分布具有未知方差和实验使用小样本时使用。

Z 检验函数

两个总体均值的统计检测手段。双样本 Z 检验检查两个样本是否不同，通常在两个正态分布具有已知方差和实验使用大样本时使用。

卡方检验函数

通常用于定性变量研究。该函数可用来将在单向频率表中观察到的频率与预期的频率进行比较，或者研究列联表中两个变量之间的连接。Chi-squared test functions are used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more groups. Often a histogram is used, and the different bins are compared to an expected distribution.

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

Chi2Test_chi2

Chi2Test_chi2() 返回一个或两个值系列的**聚合卡方**检验值。

```
Chi2Test_chi2() 返回一个或两个值系列的聚合卡方检验值。(col, row, actual_value[, expected_value])
```

Chi2Test_df

Chi2Test_df() 用于返回一个或两个值系列的聚合卡方检验 df 值（自由度）。

```
Chi2Test_df() 用于返回一个或两个值系列的聚合卡方检验 df 值(自由度)。(col, row, actual_value[, expected_value])
```

Chi2Test_p

Chi2Test_p() 用于返回一个或两个值系列的聚合卡方检验 P 值(显著性)。

```
Chi2Test_p - 图表函数(col, row, actual_value[, expected_value])
```

另请参见:

- *T 检验函数 (page 304)*
- *Z 检验函数 (page 335)*

Chi2Test_chi2

Chi2Test_chi2() 返回一个或两个值系列的聚合卡方检验值。

如果在数据加载脚本中使用此函数,则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数,则值迭代于图表维度。

> *全部 Qlik Sense chi^2 检验函数都具有相同的参数。*

语法:

```
Chi2Test_chi2(col, row, actual_value[, expected_value])
```

返回数据类型: 数字

参数:

参数

参数	说明
col, row	可以对值矩阵中的每一列和每一行进行检验。
actual_value	位于指定**col**和**row**位置的数据的观察值。
expected_value	位于指定**col**和**row**位置的分布的预期值。

限制:

值表达式内出现文本值,NULL 值和缺失值都将会导致函数返回 NULL 值。

示例:

```
Chi2Test_chi2( Grp, Grade, Count )
Chi2Test_chi2( Gender, Description, Observed, Expected )
```

另请参见:

- *如何在图表中使用 chi2-test 函数的示例 (page 348)*
- *如何在数据加载脚本中使用 chi2-test 函数的示例 (page 351)*

Chi2Test_df

Chi2Test_df() 用于返回一个或两个值系列的聚合卡方检验 df 值(自由度)。

如果在数据加载脚本中使用此函数,则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数,则值迭代于图表维度。

全部 Qlik Sense chi^2 检验函数都具有相同的参数。

语法:

```
Chi2Test_df(col, row, actual_value[, expected_value])
```

返回数据类型: 数字

参数:

参数

参数	说明
col, row	可以对值矩阵中的每一列和每一行进行检验。
actual_value	位于指定**col**和**row**位置的数据的观察值。
expected_value	位于指定**col**和**row**位置的分布的预期值。

限制:

值表达式内出现文本值,NULL 值和缺失值都将会导致函数返回 NULL 值。

示例:

```
Chi2Test_df( Grp, Grade, Count )
Chi2Test_df( Gender, Description, Observed, Expected )
```

另请参见:

- *如何在图表中使用 chi2-test 函数的示例 (page 348)*
- *如何在数据加载脚本中使用 chi2-test 函数的示例 (page 351)*

Chi2Test_p - 图表函数

Chi2Test_p() 用于返回一个或两个值系列的聚合卡方检验 P 值(显著性)。既可以根据指定 **col** 和 **row** 矩阵内的变体所用的 **actual_value** 测试值完成此检验,也可以通过比较 **actual_value** 的值和 **expected_value** 的相应值(如果指定)完成此检验。

如果在数据加载脚本中使用此函数,则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数,则值迭代于图表维度。

全部 Qlik Sense chi^2 检验函数都具有相同的参数。

语法：

```
Chi2Test_p(col, row, actual_value[, expected_value])
```

返回数据类型：数字

参数：

参数

参数	说明
col, row	可以对值矩阵中的每一列和每一行进行检验。
actual_value	位于指定**col**和**row**位置的数据的观察值。
expected_value	位于指定**col**和**row**位置的分布的预期值。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
Chi2Test_p( Grp, Grade, Count )
Chi2Test_p( Gender, Description, Observed, Expected )
```

另请参见：

如何在图表中使用 chi2-test 函数的示例 (page 348)
如何在数据加载脚本中使用 chi2-test 函数的示例 (page 351)

T 检验函数

T 检验函数用于统计检查两个总体均值。双样本 T 检验检查两个样本是否不同，通常在两个正态分布具有未知方差和实验使用小样本时使用。

在以下各节中，根据应用于每个函数类型的学生检验样本对 T 检验统计检验函数分组。

创建典型的 t-test 报告 (page 352)

两个独立样本 T 检验

以下函数应用于两个独立学生样本 T 检验。

ttest_conf

TTest_conf 用于返回两个独立样本的聚合 T 检验置信区间值。

TTest_conf 用于返回两个独立样本的聚合 T 检验置信区间值。 (grp, value [, sig[, eq_var]])

ttest_df

TTest_df() 用于返回两个独立值系列的聚合学生 T 检验值(自由度) 。

TTest_df() 用于返回两个独立值系列的聚合学生 T 检验值(自由度)。 (grp, value [, eq_var)

ttest_dif

TTest_dif() 是一个数字函数, 用于返回两个独立值系列的聚合学生 T 检验平均差。

TTest_dif() 是一个数字函数, 用于返回两个独立值系列的聚合学生 T 检验平均差。 (grp, value)

ttest_lower

TTest_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

TTest_lower() 用于返回两个独立值系列的置信区间底端的聚合值。 (grp, value [, sig[, eq_var]])

ttest_sig

TTest_sig() 用于返回两个独立值系列的聚合学生 T 检验双尾级显著性。

TTest_sig() 用于返回两个独立值系列的聚合学生 T 检验双尾级显著性。 (grp, value [, eq_var])

ttest_sterr

TTest_sterr() 用于返回两个独立值系列的聚合学生 T 检验平均差标准误差。

TTest_sterr() 用于返回两个独立值系列的聚合学生 T 检验平均差标准误差。 (grp, value [, eq_var])

ttest_t

TTest_t() 用于返回两个独立值系列的聚合 T 值。

TTest_t() 用于返回两个独立值系列的聚合 T 值。 (grp, value [, eq_var])

ttest_upper

TTest_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

TTest_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。 (grp, value [, sig [, eq_var]])

两个独立加权样本 T 检验

以下函数应用于两个独立学生样本 T 检验, 其中输入数据系列给定为加权两列格式。

ttestw_conf

TTestw_conf() 用于返回两个独立值系列的聚合 T 值。

TTestw_conf() 用于返回两个独立值系列的聚合 T 值。 (weight, grp, value [, sig[, eq_var]])

ttestw_df

TTestw_df() 用于返回两个独立值系列的聚合学生 T 检验 df 值(自由度)。

```
TTestw_df() 用于返回两个独立值系列的聚合学生 T 检验 df 值(自由度)。 (weight, grp, value [, eq_var])
```

ttestw_dif

TTestw_dif() 用于返回两个独立值系列的聚合学生 T 检验平均差。

```
TTestw_dif() 用于返回两个独立值系列的聚合学生 T 检验平均差。 ( weight, grp, value)
```

ttestw_lower

TTestw_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

```
TTestw_lower() 用于返回两个独立值系列的置信区间底端的聚合值。 (weight, grp, value [, sig[, eq_var]])
```

ttestw_sig

TTestw_sig() 用于返回两个独立值系列的聚合学生 T 检验双尾级显著性。

```
TTestw_sig() 用于返回两个独立值系列的聚合学生 T 检验双尾级显著性。 ( weight, grp, value [, eq_var])
```

ttestw_sterr

TTestw_sterr() 用于返回两个独立值系列的聚合学生 T 检验平均差标准误差。

```
TTestw_sterr() 用于返回两个独立值系列的聚合学生 T 检验平均差标准误差。 (weight, grp, value [, eq_var])
```

ttestw_t

TTestw_t() 用于返回两个独立值系列的聚合 T 值。

```
TTestw_t() 用于返回两个独立值系列的聚合 T 值。 (weight, grp, value [, eq_var])
```

ttestw_upper

TTestw_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

```
TTestw_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。 (weight, grp, value [, sig [, eq_var]])
```

一个样本 T 检验

以下函数应用于一个学生样本 T 检验。

ttest1_conf

TTest1_conf() 用于返回值系列的聚合置信区间值。

```
TTest1_conf() 用于返回值系列的聚合置信区间值。 (value [, sig])
```

ttest1_df

TTest1_df() 用于返回值系列的聚合学生 T 检验 df 值(自由度)。

TTest1_df() 用于返回值系列的聚合学生 T 检验 df 值(自由度)。 (value)

ttest1_dif
TTest1_dif() 用于返回值系列的聚合学生 T 检验平均差。

TTest1_dif() 用于返回值系列的聚合学生 T 检验平均差。 (value)

ttest1_lower
TTest1_lower() 用于返回值系列的置信区间底端的聚合值。

TTest1_lower() 用于返回值系列的置信区间底端的聚合值。 (value [, sig])

ttest1_sig
TTest1_sig() 用于返回值系列的聚合学生 T 检验双尾级显著性。

TTest1_sig() 用于返回值系列的聚合学生 T 检验双尾级显著性。 (value)

ttest1_sterr
TTest1_sterr() 用于返回值系列的聚合学生 T 检验平均差标准误差。

TTest1_sterr() 用于返回值系列的聚合学生 T 检验平均差标准误差。 (value)

ttest1_t
TTest1_t() 用于返回值系列的聚合 T 值。

TTest1_t() 用于返回值系列的聚合 T 值。 (value)

ttest1_upper
TTest1_upper() 用于返回值系列的置信区间顶端的聚合值。

TTest1_upper() 用于返回值系列的置信区间顶端的聚合值。 (value [, sig])

一个加权样本 T 检验

以下函数应用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

ttest1w_conf
TTest1w_conf() 是一个**数值**函数，用于返回值系列的聚合置信区间值。

TTest1w_conf() 是一个数值函数，用于返回值系列的聚合置信区间值。 (weight, value [, sig])

ttest1w_df
TTest1w_df() 用于返回值系列的聚合学生 T 检验 df 值(自由度)。

TTest1w_df() 用于返回值系列的聚合学生 T 检验 df 值(自由度)。 (weight, value)

ttest1w_dif
TTest1w_dif() 用于返回值系列的聚合学生 T 检验平均差。

TTest1w_dif() 用于返回值系列的聚合学生 T 检验平均差。 (weight, value)

ttest1w_lower
TTest1w_lower() 用于返回值系列的置信区间底端的聚合值。

TTest1w_lower() 用于返回值系列的置信区间底端的聚合值。 (weight, value [, sig])

ttest1w_sig
TTest1w_sig() 用于返回值系列的聚合学生 T 检验双尾级显著性。

TTest1w_sig() 用于返回值系列的聚合学生 T 检验双尾级显著性。 (weight, value)

ttest1w_sterr
TTest1w_sterr() 用于返回值系列的聚合学生 T 检验平均差标准误差。

TTest1w_sterr() 用于返回值系列的聚合学生 T 检验平均差标准误差。 (weight, value)

ttest1w_t
TTest1w_t() 用于返回值系列的聚合 T 值。

TTest1w_t() 用于返回值系列的聚合 T 值。 (weight, value)

ttest1w_upper
TTest1w_upper() 用于返回值系列的置信区间顶端的聚合值。

TTest1w_upper() 用于返回值系列的置信区间顶端的聚合值。 (weight, value [, sig])

TTest_conf

TTest_conf 用于返回两个独立样本的聚合 T 检验置信区间值。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

TTest_conf (grp, value [, sig [, eq_var]])

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_conf( Group, Value )
TTest_conf( Group, Value, Sig, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest_df

TTest_df() 用于返回两个独立值系列的聚合学生 T 检验值(自由度) 。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest_df (grp, value [, eq_var])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_df( Group, Value )
TTest_df( Group, Value, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest_dif

TTest_dif() 是一个数字函数，用于返回两个独立值系列的聚合学生 T 检验平均差。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest_dif (grp, value [, eq_var] )
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_dif( Group, Value )
TTest_dif( Group, Value, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest_lower

TTest_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest_lower (grp, value [, sig [, eq_var]])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_lower( Group, Value )
TTest_lower( Group, Value, Sig, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest_sig

TTest_sig() 用于返回两个独立值系列的聚合学生 T 检验双尾级显著性。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest_sig (grp, value [, eq_var])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_sig( Group, Value )
TTest_sig( Group, Value, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest_sterr

TTest_sterr() 用于返回两个独立值系列的聚合学生 T 检验平均差标准误差。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest_sterr (grp, value [, eq_var])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_sterr( Group, Value )
TTest_sterr( Group, Value, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest_t

TTest_t() 用于返回两个独立值系列的聚合 T 值。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest_t(grp, value[, eq_var])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_t( Group, Value, false )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest_upper

TTest_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest_upper (grp, value [, sig [, eq_var]])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest_upper( Group, Value )
TTest_upper( Group, Value, sig, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTestw_conf

TTestw_conf() 用于返回两个独立值系列的聚合 T 值。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTestw_conf (weight, grp, value [, sig [, eq_var]])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_conf( Weight, Group, Value )
TTestw_conf( Weight, Group, Value, sig, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTestw_df

TTestw_df() 用于返回两个独立值系列的聚合学生 T 检验 df 值(自由度)。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTestw_df (weight, grp, value [, eq_var])
```

返回数据类型：数字

参数：

参数

参数	说明
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_df( Weight, Group, Value )
TTestw_df( Weight, Group, Value, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTestw_dif

TTestw_dif() 用于返回两个独立值系列的聚合学生 T 检验平均差。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTestw_dif (weight, grp, value)
```

返回数据类型：数字

参数：

参数

参数	说明
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_dif( Weight, Group, Value )
TTestw_dif( Weight, Group, Value, false )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTestw_lower

TTestw_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTestw_lower (weight, grp, value [, sig [, eq_var]])
```

返回数据类型：数字

参数：

参数

参数	说明
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。

参数	说明
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_lower( Weight, Group, Value )
TTestw_lower( Weight, Group, Value, sig, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTestw_sig

TTestw_sig() 用于返回两个独立值系列的聚合学生 T 检验双尾级显著性。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTestw_sig ( weight, grp, value [, eq_var])
```

返回数据类型： 数字

参数：

参数

参数	说明
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_sig( Weight, Group, Value )
TTestw_sig( Weight, Group, Value, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTestw_sterr

TTestw_sterr() 用于返回两个独立值系列的聚合学生 T 检验平均差标准误差。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTestw_sterr (weight, grp, value [, eq_var])
```

返回数据类型： 数字

参数：

参数

参数	说明
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_sterr( Weight, Group, Value )
TTestw_sterr( Weight, Group, Value, false )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTestw_t

TTestw_t() 用于返回两个独立值系列的聚合 T 值。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ttestw_t (weight, grp, value [, eq_var])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_t( Weight, Group, Value )
TTestw_t( Weight, Group, Value, false )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTestw_upper

TTestw_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

此函数适用于两个独立学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTestw_upper (weight, grp, value [, sig [, eq_var]])
```

返回数据类型： 数字

参数：

参数

参数	说明
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTestw_upper( Weight, Group, Value )
TTestw_upper( Weight, Group, Value, sig, false )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1_conf

TTest1_conf() 用于返回值系列的聚合置信区间值。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_conf (value [, sig ])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_conf( Value )
TTest1_conf( Value, 0.005 )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest1_df

TTest1_df() 用于返回值系列的聚合学生 T 检验 df 值(自由度)。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_df (value)
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_df( Value )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1_dif

TTest1_dif() 用于返回值系列的聚合学生 T 检验平均差。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_dif (value)
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_dif( Value )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1_lower

TTest1_lower() 用于返回值系列的置信区间底端的聚合值。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_lower (value [, sig])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_lower( Value )
TTest1_lower( Value, 0.005 )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1_sig

TTest1_sig() 用于返回值系列的聚合学生 T 检验双尾级显著性。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_sig (value)
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_sig( Value )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1_sterr

TTest1_sterr() 用于返回值系列的聚合学生 T 检验平均差标准误差。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_sterr (value)
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_sterr( value )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest1_t

TTest1_t() 用于返回值系列的聚合 T 值。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_t (value)
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_t( value )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest1_upper

TTest1_upper() 用于返回值系列的置信区间顶端的聚合值。

此函数适用于一个学生样本 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1_upper (value [, sig])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1_upper( Value )
TTest1_upper( Value, 0.005 )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest1w_conf

TTest1w_conf() 是一个**数值**函数，用于返回值系列的聚合置信区间值。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_conf (weight, value [, sig ])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_conf( Weight, Value )
TTest1w_conf( Weight, Value, 0.005 )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest1w_df

TTest1w_df() 用于返回值系列的聚合学生 T 检验 df 值（自由度）。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_df (weight, value)
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_df( Weight, Value )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest1w_dif

TTest1w_dif() 用于返回值系列的聚合学生 T 检验平均差。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_dif (weight, value)
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_dif( Weight, Value )
```

另请参见：

创建典型的 t-test 报告 (page 352)

TTest1w_lower

TTest1w_lower() 用于返回值系列的置信区间底端的聚合值。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_lower (weight, value [, sig ])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_lower( Weight, Value )
TTest1w_lower( Weight, Value, 0.005 )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1w_sig

TTest1w_sig() 用于返回值系列的聚合学生 T 检验双尾级显著性。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_sig (weight, value)
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_sig( Weight, Value )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1w_sterr

TTest1w_sterr() 用于返回值系列的聚合学生 T 检验平均差标准误差。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_sterr (weight, value)
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_sterr( Weight, Value )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1w_t

TTest1w_t() 用于返回值系列的聚合 T 值。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_t ( weight, value)
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_t( Weight, Value )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

TTest1w_upper

TTest1w_upper() 用于返回值系列的置信区间顶端的聚合值。

此函数适用于一个学生样本 T 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
TTest1w_upper (weight, value [, sig])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
weight	**value** 中的每个值都可以根据 **weight** 中的相应加权值计数一次或多次。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
TTest1w_upper( Weight, Value )
TTest1w_upper( Weight, Value, 0.005 )
```

另请参见：

- *创建典型的 t-test 报告 (page 352)*

Z 检验函数

两个总体均值的统计检测手段。双样本 Z 检验检查两个样本是否不同，通常在两个正态分布具有已知方差和实验使用大样本时使用。

根据应用于函数的输入数据系列类型对 Z 检验统计检验函数分组。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

如何使用 z-test 函数的示例 (page 356)

一列格式函数

以下函数适用于具有简单输入数据系列的 z 检验。

ztest_conf

ZTest_conf() 用于返回值系列的聚合 Z 值。

```
ZTest_conf() 用于返回值系列的聚合 z 值。(value [, sigma [, sig ])
```

ztest_dif

ZTest_dif() 用于返回值系列的聚合 Z 检验平均差。

```
ZTest_dif() 用于返回值系列的聚合 z 检验平均差。(value [, sigma])
```

ztest_sig

ZTest_sig() 用于返回值系列的聚合 Z 检验双尾级显著性。

```
ZTest_sig() 用于返回值系列的聚合 z 检验双尾级显著性。(value [, sigma])
```

ztest_sterr

ZTest_sterr() 用于返回值系列的聚合 Z 检验平均差标准误差。

```
ZTest_sterr() 用于返回值系列的聚合 z 检验平均差标准误差。(value [, sigma])
```

ztest_z

ZTest_z() 用于返回值系列的聚合 Z 值。

```
ZTest_z() 用于返回值系列的聚合 z 值。(value [, sigma])
```

ztest_lower

ZTest_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

```
ZTest_lower() 用于返回两个独立值系列的置信区间底端的聚合值。(grp, value [, sig [,
eq_var]])
```

ztest_upper

ZTest_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

```
ZTest_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。(grp, value [, sig [, eq_var]])
```

加权两列格式函数

以下函数适用于 z 检验，其中输入数据系列给定为加权两列格式。

ztestw_conf

ZTestw_conf() 用于返回值系列的聚合 Z 置信区间值。

```
ZTestw_conf() 用于返回值系列的聚合 z 置信区间值。(weight, value [, sigma [, sig]])
```

ztestw_dif

ZTestw_dif() 用于返回值系列的聚合 Z 检验平均差。

```
ZTestw_dif() 用于返回值系列的聚合 z 检验平均差。 (weight, value [, sigma])
```

ztestw_lower

ZTestw_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

```
ZTestw_lower() 用于返回两个独立值系列的置信区间底端的聚合值。(weight, value [, sigma])
```

ztestw_sig

ZTestw_sig() 用于返回值系列的聚合 Z 检验双尾级显著性。

```
ZTestw_sig() 用于返回值系列的聚合 z 检验双尾级显著性。(weight, value [, sigma])
```

ztestw_sterr

ZTestw_sterr() 用于返回值系列的聚合 Z 检验平均差标准误差。

```
ZTestw_sterr() 用于返回值系列的聚合 z 检验平均差标准误差。(weight, value [, sigma])
```

ztestw_upper

ZTestw_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

```
ZTestw_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。(weight, value [, sigma])
```

ztestw_z

ZTestw_z() 用于返回值系列的聚合 Z 值。

```
ZTestw_z() 用于返回值系列的聚合 z 值。(weight, value [, sigma])
```

ZTest_z

ZTest_z() 用于返回值系列的聚合 Z 值。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_z(value[, sigma])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。假定总体均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该均值。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTest_z( Value-TestValue )
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTest_sig

ZTest_sig() 用于返回值系列的聚合 Z 检验双尾级显著性。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_sig(value[, sigma])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。假定总体均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该均值。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTest_sig(Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTest_dif

ZTest_dif() 用于返回值系列的聚合 Z 检验平均差。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_dif(value[, sigma])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。假定总体均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该均值。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTest_dif(Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTest_sterr

ZTest_sterr() 用于返回值系列的聚合 Z 检验平均差标准误差。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_sterr(value[, sigma])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。假定总体均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该均值。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTest_sterr(Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTest_conf

ZTest_conf() 用于返回值系列的聚合 Z 值。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_conf(value[, sigma[, sig]])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。假定总体均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该均值。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTest_conf(Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTest_lower

ZTest_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_lower (grp, value [, sig [, eq_var]])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTest_lower( Group, Value )
ZTest_lower( Group, Value, sig, false )
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTest_upper

ZTest_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_upper (grp, value [, sig [, eq_var]])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTest_upper( Group, Value )
ZTest_upper( Group, Value, sig, false )
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTestw_z

ZTestw_z() 用于返回值系列的聚合 Z 值。

此函数适用于 z 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTestw_z (weight, value [, sigma])
```

返回数据类型：数字

参数：

参数

参数	说明
value	值应通过 **value** 返回。假定样本均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该值。
weight	**value** 中的每个样本值都可以根据 **weight** 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTestw_z( Weight, Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTestw_sig

ZTestw_sig() 用于返回值系列的聚合 Z 检验双尾级显著性。

此函数适用于 z 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTestw_sig (weight, value [, sigma])
```

返回数据类型：数字

参数：

参数

参数	说明
value	值应通过 **value** 返回。假定样本均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该值。

参数	说明
weight	**value** 中的每个样本值都可以根据 **weight** 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTestw_sig( Weight, Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTestw_dif

ZTestw_dif() 用于返回值系列的聚合 Z 检验平均差。

此函数适用于 z 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTestw_dif ( weight, value [, sigma])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	值应通过 **value** 返回。假定样本均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该值。
weight	**value** 中的每个样本值都可以根据 **weight** 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTestw_dif( Weight, Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTestw_sterr

ZTestw_sterr() 用于返回值系列的聚合 Z 检验平均差标准误差。

此函数适用于 z 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTestw_sterr (weight, value [, sigma])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	值应通过 **value** 返回。假定样本均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该值。
weight	**value** 中的每个样本值都可以根据 **weight** 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTestw_sterr( Weight, Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTestw_conf

ZTestw_conf() 用于返回值系列的聚合 Z 置信区间值。

此函数适用于 z 检验，其中输入数据系列给定为加权两列格式。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTest_conf(weight, value[, sigma[, sig]])
```

返回数据类型：数字

参数：

参数

参数	说明
value	可以计算样本值。假定总体均值为 0。如果您想要围绕另一均值执行检验，则从样本值中减去该均值。
weight	**value** 中的每个样本值都可以根据 **weight** 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 **sigma** 中声明标准偏差。如果省略 **sigma**，则会使用实际样本标准偏差。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTestw_conf( Weight, Value-TestValue)
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTestw_lower

ZTestw_lower() 用于返回两个独立值系列的置信区间底端的聚合值。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTestw_lower (grp, value [, sig [, eq_var]])
```

返回数据类型：数字

参数：

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTestw_lower( Group, Value )
ZTestw_lower( Group, Value, sig, false )
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

ZTestw_upper

ZTestw_upper() 用于返回两个独立值系列的置信区间顶端的聚合值。

此函数适用于独立学生样本的 T 检验。

如果在数据加载脚本中使用此函数，则值会迭代于 group by 子句定义的大量记录。

如果在图表表达式中使用此函数，则值迭代于图表维度。

语法：

```
ZTestw_upper (grp, value [, sig [, eq_var]])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	可以计算样本值。样本值必须按照在 **group** 中所指定的两个值进行逻辑分组。如果在加载脚本中没有提供样本值的字段名称，则会自动将字段命名为 **Value**。
grp	该字段包含两个样本组的每个样本组的名称。如果在加载脚本中没有提供样本组的字段名称，则会自动将字段命名为 **Type**。
sig	可以在 **sig** 中指定双尾级显著性。如果省略，应将 **sig** 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 **eq_var** 为 False (0)，则可以假定两个样本的单独方差。如果指定 **eq_var** 为 True (1)，则可以假定两个样本之间的两方差齐。

限制：

值表达式内出现文本值，NULL 值和缺失值都将会导致函数返回 NULL 值。

示例：

```
ZTestw_upper( Group, Value )
ZTestw_upper( Group, Value, sig, false )
```

另请参见：

如何使用 z-test 函数的示例 (page 356)

统计检验函数示例

本节包含应用于图表和数据加载脚本的统计检验函数的示例。

如何在图表中使用 chi2-test 函数的示例

chi2-test 函数用于查找与卡方统计分析相关的值。

本节介绍如何通过使用样本数据查找 Qlik Sense 可用的卡方分布检验函数值来创建可视化内容。有关语法和参数说明，请参阅单独的 chi2-test 图表函数主题。

为样本加载数据

有三个样本数据集合，它们介绍了可载入脚本的三个不同的统计样本。

执行以下操作：

1. 创建新应用程序。
2. 在数据加载中，输入以下内容：

```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the
top of the script.
Sample_1:
LOAD * inline [
Grp,Grade,Count
I,A,15
I,B,7
I,C,9
I,D,20
I,E,26
I,F,19
II,A,10
II,B,11
II,C,7
II,D,15
II,E,21
II,F,16
];
// Sample_2 data is pre-aggregated: If raw data is used, it must be aggregated using
count()...
Sample_2:
LOAD * inline [
Sex,Opinion,OpCount
1,2,58
1,1,11
1,0,10
2,2,35
2,1,25
2,0,23 ] (delimiter is ',');
// Sample_3a data is transformed using the crosstable statement...
Sample_3a:
crosstable(Gender, Actual) LOAD
Description,
[Men (Actual)] as Men,
[Women (Actual)] as Women;
LOAD * inline [
Men (Actual),Women (Actual),Description
58,35,Agree
11,25,Neutral
10,23,Disagree ] (delimiter is ',');
// Sample_3b data is transformed using the crosstable statement...
Sample_3b:
crosstable(Gender, Expected) LOAD
Description,
[Men (Expected)] as Men,
[Women (Expected)] as Women;
LOAD * inline [
Men (Expected),Women (Expected),Description
45.35,47.65,Agree
17.56,18.44,Neutral
16.09,16.91,Disagree ] (delimiter is ',');
// Sample_3a and Sample_3b will result in a (fairly harmless) Synthetic Key...
```

3. 单击 加载数据。

创建 chi2-test 图表函数可视化内容

示例：样本 1

执行以下操作：

1. 在数据加载编辑器中，单击转到应用视图，然后单击您以前创建的表格。
   随即打开工作表视图。
2. 单击**编辑工作表**以编辑工作表。
3. 在**图表**中添加表格，在**字段**中添加 Grp、Grade 和 Count 作为维度。
   此表格将显示样本数据。
4. 添加另一个使用以下表达式作为维度的表格：
   `ValueList('p','df','Chi2')`
   这样可以使用组合维度函数为具有三个 chi2-test 函数名称的维度创建标签。
5. 在表格中添加以下表达式作为度量：
   ```
   IF(ValueList('p','df','Chi2')='p',Chi2Test_p(Grp,Grade,Count),
   IF(ValueList('p','df','Chi2')='df',Chi2Test_df(Grp,Grade,Count),
   Chi2Test_Chi2(Grp,Grade,Count)))
   ```
   这样可以将表格中每个 chi2-test 函数的结果值放在其相关组合维度旁。
6. 将度量的**数字格式**设置为**数字**和 **3** 个**有效数字**。

> *在度量的表达式中，可以使用以下表达式：* `Pick(Match(ValueList('p','df','Chi2'),'p','df','Chi2'),Chi2Test_p(Grp,Grade,Count),Chi2Test_df(Grp,Grade,Count),Chi2Test_Chi2(Grp,Grade,Count))`

结果：

样本 1 数据的 chi2-test 函数结果表格中将包含以下值：

结果表

p	df	Chi2
0.820	5	2.21

示例：样本 2

执行以下操作：

1. 在样本 1 示例中所编辑的表格中，在**图表**中添加表格，在**字段**中添加 Sex、Opinion 和 OpCount 作为维度。
2. 使用**复制**和**粘贴**命令从样本 1 中复制结果表格。编辑度量中的表达式，并将三个 chi2-test 函数中的参数替换为样本 2 数据中所使用的字段名称，例如：`Chi2Test_p(Sex,Opinion,OpCount)`。

结果：

样本 2 数据的 chi2-test 函数结果表格中将包含以下值：

结果表

p	df	Chi2
0.000309	2	16.2

示例：样本 3

执行以下操作：

1. 以样本 1 和样本 2 数据示例中的方式再创建两个表格。在维度表格中，使用以下字段作为维度：Gender、Description、Actual 和 Expected。
2. 在结果表格中，使用样本 3 数据中所使用的字段名称，例如：`Chi2Test_p (Gender,Description,Actual,Expected)`。

结果：

样本 3 数据的 chi2-test 函数结果表格中将包含以下值：

结果表

p	df	Chi2
0.000308	2	16.2

如何在数据加载脚本中使用 chi2-test 函数的示例

chi2-test 函数用于查找与卡方统计分析相关的值。本部分介绍如何在数据加载脚本中使用 Qlik Sense 可用的卡方分布检验函数。有关语法和参数说明，请参阅单独的 chi2-test 脚本函数主题。

此示例使用包含获得 (A-F) 分数的两组学生(I 和 II) 的学生人数的表格。

Data table

Group	A	B	C	D	E	F
I	15	7	9	20	26	19
II	10	11	7	15	21	16

加载样本数据

执行以下操作：

1. 创建新应用程序。
2. 在数据加载编辑器中，输入以下内容：

```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the top of the script.
Sample_1:
LOAD * inline [
Grp,Grade,Count
I,A,15
I,B,7
I,C,9
```

```
I,D,20
I,E,26
I,F,19
II,A,10
II,B,11
II,C,7
II,D,15
II,E,21
II,F,16
];
```

3. 单击 加载数据。

现在，您已加载样本数据。

加载 chi2-test 函数值

现在，我们将根据新表格中的样本数据加载 chi2-test 值，这些值已经按 Grp 进行了分组。

执行以下操作：

1. 在数据加载编辑器中，将以下内容添加到脚本的末尾：
```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the top of the script.
Chi2_table:
LOAD Grp,
Chi2Test_chi2(Grp, Grade, Count) as chi2,
Chi2Test_df(Grp, Grade, Count) as df,
Chi2Test_p(Grp, Grade, Count) as p
resident Sample_1 group by Grp;
```
2. 单击 加载数据。

现在，您已经加载名为 Chi2_table 的表格中的 chi2-test 值。

结果

您可以在**预览**下方的数据模型查看器中查看生成的 chi2-test 值，如下所示：

Results

Grp	chi2	df	p
I	16.00	5	0.007
II	9.40	5	0.094

创建典型的 t-test 报告

典型的学生 t-test 报表可以包括具有 **Group Statistics** 和 **Independent Samples Test** 结果的表格。

在以下部分中，我们将使用应用于两个独立样本组 Observation 和 Comparison 的 Qlik Senset-test 函数来创建这些表格。这些样本的相应表格如下所示：

组统计

Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

Independent Sample Test

独立的样本测试

Type	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)
Equal Variance not Assumed	3.534	37.116717335823	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	3.534	38	0.001	15.2	4.30101	6.49306	23.9069

加载样本数据

执行以下操作:

1. 创建具有新表格的新应用,并打开该表格。
2. 在数据加载编辑器中输入以下内容:

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
```

```
19|37
25|2 ] (delimiter is '|');
```

在此加载脚本中包括 **recno()**，因为 **crosstable** 需要三个参数。因此，**recno()** 仅提供额外参数，在这种情况下提供每一行的 ID。如果不使用此函数，则不会加载 **Comparison** 样本值。

3. 单击 加载数据。

创建 Group Statistics 表格

执行以下操作：

1. 在数据加载编辑器中，单击 转到应用视图，然后单击您以前创建的表格。
   这样将打开工作表视图。
2. 单击 **编辑工作表**以编辑工作表。
3. 在**图表**中添加表格，在**字段**中添加以下表达式作为度量：

示例表达式

标签	表达式
N	Count(Value)
Mean	Avg(Value)
Standard Deviation	Stdev(Value)
Standard Error Mean	Sterr(Value)

4. 在表格中添加 Type 作为维度。
5. 单击**排序**，并将 Type 移到排序列表顶部。

结果：

这些样本的 Group Statistics 表格如下所示：

组统计

Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

创建 Two Independent Sample Student's T-test 表格

执行以下操作：

1. 单击 **编辑工作表**以编辑工作表。
2. 在表格中添加以下表达式作为维度。`=ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1))`

3. 在**图表**中添加使用以下表达式作为度量的表格：

示例表达式

标签	表达式
conf	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_conf(Type, Value),TTest_conf(Type, Value, 0))
t	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_t(Type, Value),TTest_t(Type, Value, 0))
df	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_df(Type, Value),TTest_df(Type, Value, 0))
Sig. (2-tailed)	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_sig(Type, Value),TTest_sig(Type, Value, 0))
Mean Difference	TTest_dif(Type, Value)
Standard Error Difference	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_sterr(Type, Value),TTest_sterr(Type, Value, 0))
95% Confidence Interval of the Difference (Lower)	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_lower(Type, Value,(1-(95)/100)/2),TTest_lower (Type, Value,(1-(95)/100)/2, 0))
95% Confidence Interval of the Difference (Upper)	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_upper(Type, Value,(1-(95)/100)/2),TTest_upper (Type, Value,(1-(95)/100)/2, 0))

结果：

独立的样本测试

Type	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)
Equal Variance not Assumed	3.534	37.116717335823	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	3.534	38	0.001	15.2	4.30101	6.49306	23.9069

如何使用 z-test 函数的示例

z-test 函数用于查找与大量数据样本的 z-test 统计分析相关的值，通常大于 30，其中方差为已知。

本节介绍如何通过使用样本数据查找 Qlik Sense中可用的 z-test 函数值来创建可视化内容。有关语法和参数说明，请参阅单独的 z-test 图表函数主题。

加载样本数据

此处使用的样本数据与 t-test 函数示例中所使用的样本数据相同。对于 Z 检验分析，样本数据大小通常被视为过小，但足以用于说明如何在 Qlik Sense 中使用不同的 z-test 函数。

执行以下操作：

1. 创建具有新表格的新应用，并打开该表格。

   如果为 t-test 函数创建了应用，则可以使用该应用，并为这些函数创建新表格。

2. 在数据加载编辑器中，输入以下内容：

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

   在此加载脚本中包括 **recno()**，因为 **crosstable** 需要三个参数。因此，**recno()** 仅提供额外参数，在这种情况下提供每一行的 ID。如果不使用此函数，则不会加载 **Comparison** 样本值。

3. 单击  加载数据。

创建 z-test 图表函数可视化内容

执行以下操作：

1. 在数据加载编辑器中，单击  以转到应用视图，然后单击您在加载数据时创建的表格。
   随即打开工作表视图。

2. 单击**编辑工作表**以编辑工作表。
3. 在**图表**中添加表格，在**字段**中添加 Type 作为维度。
4. 在表格中添加以下表达式作为度量。

示例表达式

标签	表达式
ZTest Conf	ZTest_conf(Value)
ZTest Dif	ZTest_dif(Value)
ZTest Sig	ZTest_sig(Value)
ZTest Sterr	ZTest_sterr(Value)
ZTest Z	ZTest_z(Value)

*您可能希望调整度量的数字格式，以便查看有意义的值。如果将大多数度量的数字格式设置为**数字>简单**，而不是**Auto**，此表格更易于阅读。但是例如对于 ZTest Sig，使用数字格式：**自定义**，然后将格式调整为 **###**。*

结果：

样本数据的 z-test 函数结果表格中将包含以下值：

结果表

Type	ZTest Conf	ZTest Dif	ZTest Sig	ZTest Sterr	ZTest Z
Comparison	6.40	11.95	0.000123	3.27	3.66
Value	5.48	27.15	0.001	2.80	9.71

创建 z-testw 图表函数可视化内容

z-testw 函数在输入数据系列采用加权两列格式时使用。表达式需要使用参数 weight 的值。此处的示例始终使用值 2，但您可以使用表达式，用于定义每个观测项的 weight 值。

示例和结果：

z-test 函数使用相同的样本数据和数字格式，z-testw 函数的结果表格将包含以下值：

结果表

Type	ZTestw Conf	ZTestw Dif	ZTestw Sig	ZTestw Sterr	ZTestw Z
Comparison	3.53	2.95	5.27e-005	1.80	3.88
Value	2.97	34.25	0	4.52	20.49

字符串聚合函数

本节介绍字符串相关的聚合函数。

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

数据加载脚本中的字符串聚合函数

Concat

Concat() 用于组合字符串值。此脚本函数用于返回迭代于 **group by** 子句定义的许多记录的所有表达式值的聚合字符串串联。

```
Concat ([ distinct ] expression [, delimiter [, sort-weight]])
```

FirstValue

FirstValue() 用于返回首先从表达式定义的记录加载，然后通过 **group by** 子句排序的值。

> *此函数仅可用作脚本函数。*

```
FirstValue  (expression)
```

LastValue

LastValue() 用于返回最后从表达式定义的记录加载，然后通过 **group by** 子句排序的值。

> *此函数仅可用作脚本函数。*

```
LastValue  (expression)
```

MaxString

MaxString() 用于查找表达式中的字符串值，并返回通过 **group by** 子句定义的大量记录按字母顺序排序的最后一个文本值。

```
MaxString (expression )
```

MinString

MinString() 用于查找表达式中的字符串值，并返回通过 **group by** 子句定义的大量记录按字母顺序排序的第一个文本值。

```
MinString (expression )
```

图表中的字符串聚合函数

以下图表函数可用于在图表中聚合字符串。

Concat

Concat() 用于组合字符串值。该函数用于返回通过每个维度计算的所有表达式值的聚合字符串串联。

```
Concat - 图表函数({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] string[, delimiter[, sort_weight]])
```

MaxString

MaxString() 用于查找表达式或字段中的字符串值，并以字母排序顺序返回最后一个字母值。

```
MaxString - 图表函数({[SetExpression] [TOTAL [<fld{, fld}>]]} expr)
```

MinString

MinString() 用于查找表达式或字段中的字符串值，并以字母排序顺序返回第一个文本值。

```
MinString - 图表函数({[SetExpression] [TOTAL [<fld {, fld}>]]} expr)
```

Concat

Concat() 用于组合字符串值。此脚本函数用于返回迭代于 **group by** 子句定义的许多记录的所有表达式值的聚合字符串串联。

语法：

```
Concat ([ distinct ] string [, delimiter [, sort-weight]])
```

返回数据类型： 字符串

参数：

表达式或字段，其中包含要处理的字符串。

参数

参数	说明
string	表达式或字段，其中包含要处理的字符串。
delimiter	每个值均由delimiter内的字符串分隔。
sort-weight	串联的顺序可由维度 **sort-weight** 的值决定，如果存在，与最低值对应的字符串首先出现在串联中.
distinct	如果在表达式前出现单词 **distinct**，则将忽略所有重复值。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

示例和结果

示例	结果	结果一次添加至工作表
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); Concat1: LOAD SalesGroup,Concat(Team) as TeamConcat1 Resident TeamData Group By SalesGroup;	SalesGroup East West	TeamConcat1 AlphaBetaDeltaGammaGamma EpsilonEtaThetaZeta
前提是 **TeamData** 表格像之前的示例一样加载： LOAD SalesGroup,Concat(distinct Team,'-') as TeamConcat2 Resident TeamData Group By SalesGroup;	SalesGroup East West	TeamConcat2 Alpha-Beta-Delta-Gamma Epsilon-Eta-Theta-Zeta
前提是 **TeamData** 表格像之前的示例一样加载。因为已经为 **sort-weight** 添加参数，因此将会按维度 Amount 值对结果进行排序： LOAD SalesGroup,Concat(distinct Team,'- ',Amount) as TeamConcat2 Resident TeamData Group By SalesGroup;	SalesGroup East West	TeamConcat2 Delta-Beta-Gamma-Alpha Eta-Epsilon-Zeta-Theta

Concat - 图表函数

Concat() 用于组合字符串值。该函数用于返回通过每个维度计算的所有表达式值的聚合字符串串联。

语法：

```
Concat({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} string[, delimiter
[, sort_weight]])
```

返回数据类型：字符串

参数：

参数

参数	说明
string	表达式或字段，其中包含要处理的字符串。
delimiter	每个值均由delimiter内的字符串分隔。
sort-weight	串联的顺序可由维度 **sort-weight** 的值决定，如果存在，与最低值对应的字符串首先出现在串联中.
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 **DISTINCT**，则将忽略计算该函数参数生成的副本。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

示例和结果：

Results table

SalesGroup	Amount	Concat(Team)	Concat(TOTAL <SalesGroup> Team)
East	25000	Alpha	AlphaBetaDeltaGammaGamma
East	20000	BetaGammaGamma	AlphaBetaDeltaGammaGamma
East	14000	Delta	AlphaBetaDeltaGammaGamma
West	17000	Epsilon	EpsilonEtaThetaZeta
West	14000	Eta	EpsilonEtaThetaZeta
West	23000	Theta	EpsilonEtaThetaZeta
West	19000	Zeta	EpsilonEtaThetaZeta

函数示例

示例	结果
`Concat(Team)`	此表格通过维度 SalesGroup 和 Amount 以及度量 Concat(Team) 中的变体构造。在忽略“总计”结果的情况下，请注意，即使分布在两个 SalesGroup 值中的八个 Team 值都有数据，在表格中连接多个 Team 字符串值的度量 Concat(Team) 的唯一结果仍是包含维度 Amount 20000 的行，它提供了结果 BetaGammaGamma。这是因为在输入数据中 Amount 20000 有三个值。当度量分布在维度中时，所有其他结果均不串联，因为 SalesGroup 和 Amount 的每个组合只有一个 Team 值。
`Concat (DISTINCT Team,', ')`	Beta, Gamma。因为 DISTINCT 限定符意味着忽略重复的 Gamma 结果。此外，将分隔符参数定义为后跟空格的逗号。
`Concat (TOTAL <SalesGroup> Team)`	如果使用 TOTAL 限定符，则会串联所有 Team 值的所有字符串值。指定字段选择项 <SalesGroup> 时，此函数会将结果划分到维度 SalesGroup 的两个值中。对于 SalesGroupEast，结果为 AlphaBetaDeltaGammaGamma。对于 SalesGroupWest，结果为 EpsilonEtaThetaZeta。
`Concat (TOTAL <SalesGroup> Team,';', Amount)`	通过为 **sort-weight** 添加参数Amount，将按维度 Amount 的值对结果排序。结果变为 DeltaBetaGammaGammaAlpha 和 EtaEpsilonZEtaTheta。

示例中所使用的数据：

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
West|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
West|Epsilon|01/09/2013|17000
West|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
West|Theta|01/12/2013|23000
] (delimiter is '|');
```

FirstValue

FirstValue() 用于返回首先从表达式定义的记录加载，然后通过 **group by** 子句排序的值。

此函数仅可用作脚本函数。

语法：

```
FirstValue ( expr)
```

返回数据类型：双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。

限制：

如果找不到任何文本值，则返回 NULL 值。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果	工作表上结果
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); FirstValue1: LOAD SalesGroup,FirstValue(Team) as FirstTeamLoaded Resident TeamData Group By SalesGroup;	SalesGroup East West	FirstTeamLoaded Gamma Zeta

LastValue

LastValue() 用于返回最后从表达式定义的记录加载，然后通过 **group by** 子句排序的值。

此函数仅可用作脚本函数。

语法：

```
LastValue ( expr )
```

返回数据类型：双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。

限制：

如果找不到任何文本值，则返回 NULL 值。

示例和结果：

将示例脚本添加到应用程序并运行。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

为了获得与下面结果列相同的外观，在属性面板中，在排序下方，从自动切换到自定义，然后取消选择数字和字母排序。

示例	结果	采用自定义排序的结果
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); LastValue1: LOAD SalesGroup,LastValue(Team) as LastTeamLoaded Resident TeamData Group By SalesGroup;	SalesGroup East West	LastTeamLoaded Beta Theta

MaxString

MaxString() 用于查找表达式中的字符串值，并返回通过 **group by** 子句定义的大量记录按字母顺序排序的最后一个文本值。

语法：

```
MaxString ( expr )
```

返回数据类型：双

参数：

参数	说明
expr	表达式或字段包含要度量的数据。

限制：

如果找不到任何文本值，则返回 NULL 值。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

示例	结果	
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); Concat1: LOAD SalesGroup,MaxString(Team) as MaxString1 Resident TeamData Group By SalesGroup;	SalesGroup East West	MaxString1 Gamma Zeta
前提是 **TeamData** 表格像之前的示例一样加载，且数据加载脚本拥有 SET 语句： SET DateFormat='DD/MM/YYYY';: LOAD SalesGroup,MaxString(Date) as MaxString2 Resident TeamData Group By SalesGroup;	SalesGroup East West	MaxString2 01/11/2013 01/12/2013

MaxString - 图表函数

MaxString() 用于查找表达式或字段中的字符串值，并以字母排序顺序返回最后一个字母值。

语法：

```
MaxString({[SetExpression] [TOTAL [<fld{, fld}>]]} expr)
```

返回数据类型：双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**（其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表），您可以创建总可能值的子集。

限制：

如果表达式不包含具有字符串呈现形式的值，则返回 NULL。

示例和结果：

结果表

SalesGroup	Amount	MaxString(Team)	MaxString(Date)
East	14000	Delta	2013/08/01
East	20000	Gamma	2013/11/01
East	25000	Alpha	2013/07/01
West	14000	Eta	2013/10/01
West	17000	Epsilon	2013/09/01
West	19000	Zeta	2013/06/01
West	23000	Theta	2013/12/01

函数示例

示例	结果
`MaxString (Team)`	维度 Amount 有三个 20000 值：两个 Gamma 值（在不同日期），和一个 Beta 值。因此度量 MaxString (Team) 的结果为 Gamma，因为此值是排序字符串中的最大值。
`MaxString (Date)`	2013/11/01 是与维度 Amount 相关的三个值中的最长 Date 值。此示例假定脚本包含 SET 语句 `SET DateFormat='YYYY-MM-DD';'`

示例中所使用的数据：

```
TeamData:
```

```
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
West|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
West|Epsilon|01/09/2013|17000
West|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
West|Theta|01/12/2013|23000
] (delimiter is '|');
```

MinString

MinString() 用于查找表达式中的字符串值，并返回通过 **group by** 子句定义的大量记录按字母顺序排序的第一个文本值。

语法：

```
MinString ( expr )
```

返回数据类型： 双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。

限制：

如果找不到任何文本值，则返回 NULL 值。

示例和结果：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

结果数据

示例	结果	
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); Concat1: LOAD SalesGroup,MinString(Team) as MinString1 Resident TeamData Group By SalesGroup;	SalesGroup East West	MinString1 Alpha Epsilon
前提是 **TeamData** 表格像之前的示例一样加载，且数据加载脚本拥有 SET 语句： SET DateFormat='DD/MM/YYYY';': LOAD SalesGroup,MinString(Date) as MinString2 Resident TeamData Group By SalesGroup;	SalesGroup East West	MinString2 01/05/2013 01/06/2013

MinString - 图表函数

MinString() 用于查找表达式或字段中的字符串值，并以字母排序顺序返回第一个文本值。

语法：

MinString({[SetExpression] [**TOTAL** [<fld {, fld}>]]} expr**)**

返回数据类型：双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
TOTAL	如果在函数参数前面出现单词 **TOTAL**，则计算给出当前选择项的所有可能值，而不只是属于当前维度值的那些值，即它会忽略图表维度。 通过使用 **TOTAL [<fld {.fld}>]**(其中 **TOTAL** 限定符后跟一个或多个字段名称作为图表维度变量的子集的列表)，您可以创建总可能值的子集。

示例和结果：

样本数据

SalesGroup	Amount	MinString(Team)	MinString(Date)
East	14000	Delta	2013/08/01
East	20000	Beta	2013/05/01
East	25000	Alpha	2013/07/01
West	14000	Eta	2013/10/01
West	17000	Epsilon	2013/09/01
West	19000	Zeta	2013/06/01
West	23000	Theta	2013/12/01

函数示例

示例	结果
`MinString (Team)`	维度 Amount 有三个 20000 值：两个 Gamma 值（在不同日期），和一个 Beta 值。因此度量 MinString (Team) 的结果为 Beta，因为此值是排序字符串中的第一个值。
`MinString (Date)`	2013/11/01 是与维度 Amount 相关的三个值中的最早 Date 值。此示例假定脚本包含 SET 语句 `SET DateFormat='YYYY-MM-DD';`

示例中所使用的数据：

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
West|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
West|Epsilon|01/09/2013|17000
West|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
West|Theta|01/12/2013|23000
] (delimiter is '|');
```

组合维度函数

组合维度是在应用中根据组合维度函数生成的值创建的，不是直接来自数据模型中的字段。当组合维度函数生成的值在图表中用作计算维度时，则可创建组合维度。例如，组合维度可让您创建维度包含根据数据生成的值的图表，即动态维度图表。

组合维度不会受到选择项影响。

以下组合维度函数可用于图表中。

ValueList

ValueList() 用于返回一组列出的值，当这组列出的值用于计算维度时将形成一个组合维度。

```
ValueList - 图表函数 (v1 {, Expression})
```

ValueLoop

ValueLoop() 用于返回一组迭代值，当这组迭代值用于计算维度时将形成一个组合维度。

```
ValueLoop - 图表函数(from [, to [, step ]])
```

ValueList - 图表函数

ValueList() 用于返回一组列出的值，当这组列出的值用于计算维度时将形成一个组合维度。

*在具有使用 **ValueList** 函数创建的组合维度的图表中，通过在图表表达式中使用相同的参数重述 **ValueList** 函数，可以引用对应特定表达式单元格的维度值。当然，此函数还可以用于布局的任意位置，但组合维度除外，因为此函数仅在聚合函数内才有实质意义。*

组合维度不会受到选择项影响。

语法：

```
ValueList(v1 {,...})
```

返回数据类型： 双

参数：

参数

参数	说明
v1	静态值(通常是字符串，但可以是数字)。
{,...}	可选静态值列表。

示例和结果：

函数示例

示例	结果
`ValueList ('Number of Orders', 'Average Order Size', 'Total Amount')`	例如，当用于在表格中创建维度时，此函数可生成三个字符串值，作为表格中的行标签。这些值随后可引用到表达式中。

示例	结果
=IF(ValueList ('Number of Orders', 'Average Order Size', 'Total Amount') = 'Number of Orders', count (SaleID), IF(ValueList ('Number of Orders', 'Average Order Size', 'Total Amount') = 'Average Order Size', avg (Amount), sum (Amount)))	此表达式可从创建的维度中获取值，并将它们引用到嵌套的 IF 语句中，作为三个聚合函数的输入： ValueList() Created dimension | Year | Added expression 522.00 Number of Orders | 2012 | 5.00 Number of Orders | 2013 | 7.00 Average Order Size | 2012 | 13.20 Average Order Size | 2013 | 15.43 Total Amount | 2012 | 66.00 Total Amount | 2013 | 108.00

示例中所使用的数据：

```
SalesPeople:
LOAD * INLINE [
SaleID|SalesPerson|Amount|Year
1|1|12|2013
2|1|23|2013
3|1|17|2013
4|2|9|2013
5|2|14|2013
6|2|29|2013
7|2|4|2013
8|1|15|2012
9|1|16|2012
10|2|11|2012
11|2|17|2012
12|2|7|2012
] (delimiter is '|');
```

ValueLoop - 图表函数

ValueLoop() 用于返回一组迭代值，当这组迭代值用于计算维度时将形成一个组合维度。该生成的值将开始于 **from** 值并结束于 **to** 值，包括步进增量的中间值。

*在具有使用 **ValueLoop** 函数创建的组合维度的图表中，通过在图表表达式中使用相同的参数重述 **ValueLoop** 函数，可以引用对应特定表达式单元格的维度值。当然，此函数还可以用于布局的任意位置，但组合维度除外，因为此函数仅在聚合函数内才有实质意义。*

组合维度不会受到选择项影响。

语法：

```
ValueLoop(from [, to [, step ]])
```

返回数据类型：双

参数：

参数

参数	说明
from	要生成的值集合中的起始值。
to	要生成的值集合中的结束值。
step	两个值之间的增量大小。

示例和结果：

函数示例

示例	结果
`ValueLoop (1, 10)`	此函数可在表格中创建维度，例如可用于编号标签等用途。此处的示例将生成编号 1 到 10 的值。这些值随后可引用到表达式中。
`ValueLoop (2, 10,2)`	此示例将生成编号 2、4、6、8 和 10 的值，因为参数 step 值为 2。

嵌套聚合函数

您可能会遇到需要将某聚合应用于另一个聚合结果的情况。这种聚合被称为嵌套聚合。

不能在大多数图表表达式中嵌套聚合。但是，如果在内部聚合函数中使用 **TOTAL** 限定符，则可以嵌套聚合。

允许不超过 100 级的嵌套。

带 TOTAL 限定符的嵌套聚合函数

示例：

您想要计算 **Sales** 字段的总和，但仅包括 **OrderDate** 为去年的交易。通过聚合函数 **`Max(TOTAL Year(OrderDate))`** 可获得去年的交易。

以下聚合将返回所需结果：

```
Sum(If(Year(OrderDate)=Max(TOTAL Year(OrderDate)), Sales))
```

Qlik Sense 需要包含这种嵌套类型的 **TOTAL** 限定符。这对于所需的比较是必要的。此类嵌套需求极为常用，是很好的做法。

另请参见：

- *Aggr - 图表函数 (page 373)*

5.3 Aggr - 图表函数

Aggr() 用于返回在声明维度或维度上计算的表达式的值的阵列。例如，每个区域的每位客户的最大销售额值。

Aggr 函数用于嵌套聚合，其中每个维度值计算一次其第一个参数（内部聚合）。维度在第二个参数（以及后续参数）中指定。

此外，其中在外部聚合函数中要将 **Aggr** 函数括起来，从而将 **Aggr** 函数的结果阵列用作其所嵌套的聚合的输入。

语法：

```
Aggr({SetExpression}[DISTINCT] [NODISTINCT ] expr, StructuredParameter{, StructuredParameter})
```

返回数据类型： 双

参数：

参数

参数	说明
expr	表达式包含聚合函数。聚合函数会默认聚合选择项定义的可能记录集合。
StructuredParameter	StructuredParameter 包括维度并可选地包括排序标准，格式为：`(Dimension(Sort-type, Ordering))` 维度是一个单一字段，并且不能为表达式。维度用于确定为其计算表达式 Aggr 的值的阵列。 如果包含排序标准，则会将对维度计算的 Aggr 函数创建的值的阵列排序。如果排序顺序会影响其中包含 Aggr 函数的表达式的结果，则这点很重要。 有关如何使用排序标准的详细信息，请参阅向结构化参数中的维度添加排序标准。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果表达式参数前面是 **distinct** 限定符，或者根本没有使用限定符，则维度值的每个特殊组合只生成一个返回值。这是实现聚合的常规方式 - 维度值的每个特殊组合将在图表中占用一行。

参数	说明
NODISTINCT	如果表达式参数前面是 **nodistinct** 限定符，各维度值组合可能生成多个返回值，具体取决于基础数据结构。如果只有一个维度，则 **aggr** 函数将返回元素数量与源数据中的行数相同的阵列。

基本聚合函数，例如 **Sum**、**Min** 和 **Avg**，在比较 Aggr() 函数以创建可产生另一个聚合的临时阶段性结果集合（虚拟表）时返回单个数值。例如，通过在 **Aggr()** 语句中按客户通过合计销售额计算平均销售额值，然后计算总和结果的平均值：**Avg(TOTAL Aggr(Sum(Sales),Customer))**。

如果想要创建多层次嵌套图表聚合，则在计算维度中使用 Aggr() 函数。

限制：

Aggr() 函数中的每个维度必须是单个字段，不能是表达式（计算维度）。

向结构化参数中的维度添加排序标准

在其基本形式中，参数 StructuredParameter 在 Aggr 函数语法中是单维度。表达式：Aggr(Sum(Sales, Month)) 查找每个月销售额的总计值。但是，当包含在另一个聚合函数中时，如果不使用排序标准，将存在意外的结果。这是因为某些维度可按数字或字母等排序。

在 StructuredParameter 参数中（位于 Aggr 函数内），您可在表达式中指定有关维度的排序标准。由此，可在 Aggr 函数生成的虚拟表格上使用排序顺序。

参数 StructuredParameter 有以下语法：

```
(FieldName, (Sort-type, Ordering))
```

可以嵌套结构化参数：

```
(FieldName, (FieldName2, (Sort-type, Ordering)))
```

排序类型可为：NUMERIC、TEXT、FREQUENCY 或 LOAD_ORDER。

和每个排序类型相关的顺序类型如下：

允许的顺序类型

排序类型	允许的顺序类型
NUMERIC	ASCENDING、DESCENDING 或 REVERSE
TEXT	ASCENDING、A2Z、DESCENDING、REVERSE 或 Z2A
FREQUENCY	DESCENDING、REVERSE 或 ASCENDING
LOAD_ORDER	ASCENDING、ORIGINAL、DESCENDING 或 REVERSE

顺序类型 REVERSE 和 DESCENDING 相同。

对于排序类型 TEXT，顺序类型 ASCENDING 和 A2Z 相同，而 DESCENDING、REVERSE 和 Z2A 相同。

对于排序类型 LOAD_ORDER，顺序类型 ASCENDING 和 ORIGINAL 相同。

示例：使用聚合的图表表达式

示例 - 图表表达式

图表表达式示例 1

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
ProductData:  LOAD * inline [ Customer|Product|UnitSales|UnitPrice Astrida|AA|4|16
Astrida|AA|10|15 Astrida|BB|9|9 Betacab|BB|5|10 Betacab|CC|2|20 Betacab|DD|25|25
Canutility|AA|8|15 Canutility|CC|0|19 ] (delimiter is '|');
```

图表表达式

在 Qlik Sense 工作表中创建 KPI 可视化。在 KPI 中添加以下表达式作为度量：

```
Avg(Aggr(Sum(UnitSales*UnitPrice), Customer))
```

结果

376.7

解释

表达式 `Aggr(Sum(UnitSales*UnitPrice), Customer)` 按 **Customer** 查找销售额总计值，并返回值阵列：对于 **Customer** 值为 295、715 和 120。

实际上，我们创建了临时的值列表，而不必创建包含这些值的明确的表格或列。

这些值可用作 **Avg()** 函数的导入值，以查找销售额平均值 376.7。

图表表达式示例 2

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
ProductData:  LOAD * inline [ Customer|Product|UnitSales|UnitPrice Astrida|AA|4|16
Astrida|AA|10|15 Astrida|BB|10|15 Astrida|BB|9|9 Betacab|BB|5|10 Betacab|BB|7|12
Betacab|CC|2|22 Betacab|CC|4|20 Betacab|DD|25|25 Canutility|AA|8|15 Canutility|AA|5|11
Canutility|CC|0|19 ] (delimiter is '|');
```

图表表达式

创建一个以 **Customer**、**Product**、**UnitPrice** 和 **UnitSales** 为维度的 Qlik Sense 工作表中的表格可视化。在表格中添加以下表达式作为度量：

```
Aggr(NODISTINCT Max(UnitPrice), Customer, Product)
```

结果

Customer	Product	UnitPrice	UnitSales	Aggr(NODISTINCT Max(UnitPrice), Customer, Product)
Astrida	AA	15	10	16
Astrida	AA	16	4	16
Astrida	BB	9	9	15
Astrida	BB	15	10	15
Betacab	BB	10	5	12
Betacab	BB	12	7	12
Betacab	CC	20	4	22
Betacab	CC	22	2	22
Betacab	DD	25	25	25
Canutility	AA	11	5	15
Canutility	AA	15	8	15
Canutility	CC	19	0	19

解释

值阵列:16、16、15、15、12、12、22、22、25、15、15 和 19。**nodistinct** 限定符意味着阵列包含源数据中每行的一个元素:每个值都是每个 **Customer** 和 **Product** 的最高 **UnitPrice**。

图表表达式示例 3

加载脚本

将以下数据作为数据加载编辑中的内联加载载入,以创建以下图表表达式示例。

```
Set vNumberOfOrders = 1000;  OrderLines: Load   RowNo() as OrderLineID,         OrderID,        OrderDate,
Round((Year(OrderDate)-2005)*1000*Rand()*Rand()*Rand1) as Sales         While Rand()<=0.5 or IterNo
()=1; Load * Where OrderDate<=Today(); Load     Rand() as Rand1,        Date(MakeDate(2013)+Floor
((365*4+1)*Rand())) as OrderDate,       RecNo() as OrderID      Autogenerate vNumberOfOrders;
Calendar: Load distinct         Year(OrderDate) as Year,        Month(OrderDate) as Month,      OrderDate
Resident OrderLines;
```

图表表达式

在 Qlik Sense 工作表中创建表可视化,以**年**和**月**为维度。在表格中添加以下表达式作为度量:

- `Sum(Sales)`
- `Sum(Aggr( Rangesum(Above(Sum(Sales),0,12)), (Year, (Numeric, Ascending)), (Month, (Numeric, Ascending)) ))` 标记为表格中的 Structured Aggr()。

结果

Year	Month	Sum(Sales)	Structured Aggr()
2013	Jan	53495	53495
2013	Feb	48580	102075
2013	Mar	25651	127726
2013	Apr	36585	164311
2013	May	61211	225522
2013	Jun	23689	249211
2013	Jul	42311	291522
2013	Aug	41913	333435
2013	Sep	28886	362361
2013	Oct	25977	388298
2013	Nov	44455	432753
2013	Dec	64144	496897
2014	Jan	67775	67775

解释

此示例按时间升序显示每年 12 个月期间的聚合值，因此是 **Aggr()** 表达式的结构化参数 (Numeric, Ascending) 部分。需要将两个特定维度作为结构化参数：**Year** 和 **Month**，已排序的 (1) **Year**(数值) 和 (2) **Month**(数值)。在表格或图表可视化中必须使用这两个维度。**Aggr()** 函数的维度列表必须与可视化中使用的对象的维度相对应。

您可在表格中或单独的折线图中比较这些度量之间的差异。

- `Sum(Aggr( Rangesum(Above(Sum(Sales),0,12)), (Year), (Month) ))`
- `Sum(Aggr( Rangesum(Above(Sum(Sales),0,12)), (Year, (Numeric, Ascending)), (Month, (Numeric, Ascending)) ))`

应该清楚地看到，只有后一个表达式执行所需的聚合值累加。

另请参见：

基本聚合函数 (page 194)

5.4 颜色函数

这些函数可用于与设置和评估图表对象颜色属性相关的表达式，以及数据加载脚本。

*由于向后兼容原因，Qlik Sense 支持颜色函数 **Color()**、**qliktechblue** 和 **qliktechgray**，但不推荐使用这些函数。*

ARGB

ARGB() 用于在表达式中设置或评估图表对象的颜色属性，其中用红色成分 **r**、绿色成分 **g** 和蓝色成分 **b**，以及透明度系数（不透明度）**alpha** 定义颜色。

```
ARGB (alpha, r, g, b)
```

HSL

HSL() 用于在表达式中设置或评估图表对象的颜色属性，其中 **hue**、**saturation** 和 **luminosity** 值介于 0 与 1 之间，用于定义颜色。

```
HSL  (hue, saturation, luminosity)
```

RGB

RGB() 返回与三个参数定义的颜色的颜色代码相对应的整数：红色分量 r、绿色分量 g 和蓝色分量 b。这些分量的整数值必须介于 0 和 255 之间。该函数可在表达式中用于设置或计算图表对象的颜色属性。

```
RGB  (r, g, b)
```

Colormix1

在表达式中使用 **Colormix1()** 可根据 0 和 1 之间的值返回双色渐变的 ARGB 颜色呈现形式。

```
Colormix1 (Value , ColorZero , ColorOne)
```

Value 为 0 和 1 之间的真实数字。

- 如果 Value = 0，则会返回 ColorZero 。
- 如果 Value = 1，则会返回 ColorOne 。
- 如果 0 < Value < 1，则会返回适当的中间值底纹。

ColorZero 是指与时间间隔低端相关联的颜色的有效 RGB 颜色呈现形式。

ColorOne 是指与时间间隔高端相关联的颜色的有效 RGB 颜色呈现形式。

示例：

```
Colormix1(0.5, red(), blue())
```

返回：

```
ARGB(255,64,0,64) (purple)
```

Colormix2

在表达式中使用 **Colormix2()** 可根据 -1 和 1 之间的值返回双色渐变的 ARGB 颜色呈现形式，同时指定中心 (0) 位置的中间颜色。

```
Colormix2 (Value ,ColorMinusOne , ColorOne[ , ColorZero])
```

Value 为 -1 和 1 之间的真实数字。

- 如果 Value = -1，则会返回第一种颜色。
- 如果 Value = 1，则会返回第二种颜色。
- 如果 -1 < Value < 1，则会返回适当的混合颜色。

ColorMinusOne 是指与时间间隔低端相关联的颜色的有效 RGB 颜色呈现形式。

ColorOne 是指与时间间隔高端相关联的颜色的有效 RGB 颜色呈现形式。

ColorZero 是指与时间间隔中心相关联的颜色的可选且有效的 RGB 颜色呈现形式。

SysColor

SysColor() 返回 Windows 系统颜色 nr 的 ARGB 颜色表现形式，其中 nr 相当于 Windows API 函数 **GetSysColor(nr)** 的参数。

```
SysColor (nr)
```

ColorMapHue

ColorMapHue() 会返回颜色表的 ARGB 颜色值，该颜色表不同于 HSV 颜色模式的色调分量。颜色表以红色开头，依次为黄色、绿色、青色、蓝色、洋红色，最后再回到红色。必须指定 x 为一个介于 0 和 1 之间的值。

```
ColorMapHue (x)
```

ColorMapJet

ColorMapJet() 会返回颜色表的 ARGB 颜色值，该颜色表以蓝色为开始，依次为青色、黄色和橙色，最后再回到红色。必须指定 x 为一个介于 0 和 1 之间的值。

```
ColorMapJet (x)
```

预定义颜色函数

可以在表达式中使用以下函数预定义颜色。每个函数均会返回 RGB 颜色呈现形式。

可以指定可选的 α 因子参数，在这种情况下，将会返回 ARGB 颜色呈现形式。α 因子为 0 表示完全透明，α 因子为 255 表示完全不透明。如果未输入 α 的值，则假定其值为 255。

预定义颜色函数

颜色函数	RGB 值
black ([alpha])	(0,0,0)
blue([alpha])	(0,0,128)
brown([alpha])	(128,128,0)
cyan([alpha])	(0,128,128)
darkgray([alpha])	(128,128,128)
green([alpha])	(0,128,0)
lightblue([alpha])	(0,0,255)

lightcyan([alpha])	(0,255,255)
lightgray([alpha])	(192,192,192)
lightgreen([alpha])	(0,255,0)
lightmagenta([alpha])	(255,0,255)
lightred([alpha])	(255,0,0)
magenta([alpha])	(128,0,128)
red([alpha])	(128,0,0)
white([alpha])	(255,255,255)
yellow([alpha])	(255,255,0)

示例和结果：

示例和结果

示例	结果
`Blue()`	RGB(0,0,128)
`Blue(128)`	ARGB(128,0,0,128)

ARGB

ARGB() 用于在表达式中设置或评估图表对象的颜色属性，其中用红色成分 **r**、绿色成分 **g** 和蓝色成分 **b**，以及透明度系数（不透明度）**alpha** 定义颜色。

语法：

```
ARGB(alpha, r, g, b)
```

返回数据类型： 双

参数：

参数

参数	说明
alpha	介于 0 - 255 范围内的透明度值。0 对应完全透明，255 对应完全不透明。
r, g, b	红色，绿色和蓝色成分的值。颜色成分 0 对应无影响，其中一个 255 对应完全影响。

所有参数均必须为表达式，用于解算范围介于 0 至 255 之间的整数。

如果解释数值成分并以十六进制表示法格式化数值，则颜色成分的值比较明显。例如，浅绿色的编号为 4 278 255 360，其十六进制表示法为 FF00FF00。前两位 'FF' (255) 表示 **alpha** 通道。后面两位 '00' 表示**红色**的数量、接下来两位 'FF' 表示**绿色** 的数量，以及最后两位 '00' 表示**蓝色**的数量.

RGB

RGB() 返回与三个参数定义的颜色的颜色代码相对应的整数:红色分量 r、绿色分量 g 和蓝色分量 b。这些分量的整数值必须介于 0 和 255 之间。该函数可在表达式中用于设置或计算图表对象的颜色属性。

语法:

```
RGB (r, g, b)
```

返回数据类型: 双

参数:

参数

参数	说明
r, g, b	红色, 绿色和蓝色成分的值。颜色成分 0 对应无影响, 其中一个 255 对应完全影响。

所有参数均必须为表达式, 用于解算范围介于 0 至 255 之间的整数。

如果解释数值成分并以十六进制表示法格式化数值, 则颜色成分的值比较明显。例如, 浅绿色的编号为 4 278 255 360, 其十六进制表示法为 FF00FF00。前两位 'FF' (255) 表示 **alpha** 通道。在函数 **RGB** 和 **HSL** 中, 这始终为 'FF'(不透明)。后面两位 '00' 表示**红色**的数量、接下来两位 'FF' 表示**绿色**的数量, 以及最后两位 '00' 表示**蓝色**的数量.

示例:图表表达式
此示例将自定义颜色应用于图表:

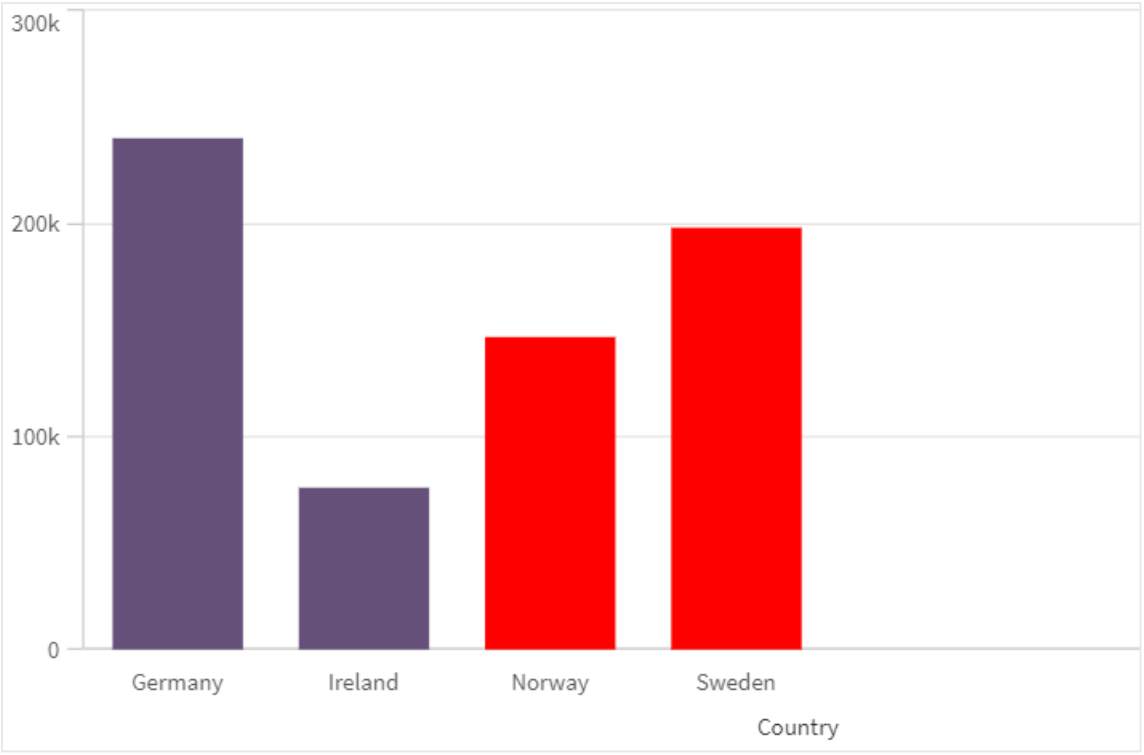
示例中所使用的数据:

```
ProductSales: Load * Inline [Country,Sales,Budget Sweden,100000,50000 Germany, 125000, 175000 Norway, 74850, 68500 Ireland, 45000, 48000 Sweden,98000,50000 Germany, 115000, 175000 Norway, 71850, 68500 Ireland, 31000, 48000 ] (delimiter is ',');
```

在**颜色和图例属性**面板中输入以下表达式:

```
If (Sum(Sales)>Sum(Budget),RGB(255,0,0),RGB(100,80,120))
```

结果:



示例:加载脚本
下面的示例显示十六进制格式值的等效 RGB 值:

```
Load Text(R & G & B) as Text, RGB(R,G,B)      as Color; Load Num#(R,'(HEX)') as R, Num#(G,'(HEX)') as G, Num#(B,'(HEX)') as B Inline [R,G,B 01,02,03 AA,BB,CC];
```

结果:

文本	颜色
010203	RGB(1,2,3)
AABBCC	RGB(170,187,204)

HSL

HSL() 用于在表达式中设置或评估图表对象的颜色属性,其中 **hue**、**saturation** 和 **luminosity** 值介于 0 与 1 之间,用于定义颜色。

语法:

```
HSL (hue, saturation, luminosity)
```

返回数据类型: 双

参数:

参数

参数	说明
hue, saturation, luminosity	范围介于 0 到 1 之间的 hue, saturation 和 luminosity 成分值。

所有参数均必须为表达式，用于解算范围介于 0 至 1 之间的整数。

如果解释数值成分并以十六进制表示法格式化数值，则颜色成分的 RGB 值比较明显。例如，浅绿色的编号为 4 278 255 360，其十六进制表示法为 FF00FF00 和 RGB (0,255,0)。这相当于 HSL (80/240, 240/240, 120/240) - 一个值为 (0.33, 1, 0.5) 的 HSL 值。

5.5 条件函数

全部条件函数一起用于评估条件，然后根据条件值返回不同的答案。所有函数均可用于数据加载脚本和图表表达式。

条件函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

alt

alt 函数用于返回首个具有有效数表示法的参数。如果未找到此类匹配，则将返回最后一个参数。可使用任何数目的参数。

```
alt (expr1[ , expr2 , expr3 , ...] , else)
```

class

class 函数用于将第一个参数赋值给类别间隔。结果是一个 a<=x<b 的双值作为文本值，其中 a 和 b 为 bin 的上限值和下限值，且下界为数值。

```
class (expression, interval [ , label [ , offset ]])
```

coalesce

coalesce 函数用于返回具有有效 non-NULL 表示法的参数中的第一个。可使用任何数目的参数。

```
coalesce(expr1[ , expr2 , expr3 , ...])
```

if

if 函数用于返回一个值，具体取决于函数提供的条件的计算结果是否为 True 或 False。

```
if (condition , then , else)
```

match

match 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数值位置。比较区分大小写。

```
match ( str, expr1 [ , expr2,...exprN ])
```

mixmatch

mixmatch 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数值位置。比较不区分大小写。

```
mixmatch ( str, expr1 [ , expr2,...exprN ])
```

pick

pick 函数用于返回列表中的第 *n* 个表达式。

```
pick (n, expr1[ , expr2,...exprN])
```

wildmatch

wildmatch 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数量。它允许在比较字符串中使用通配字符(***** 和 **?**) 。***** 匹配任何字符序列。**?** 匹配任意单个字符。比较不区分大小写。

```
wildmatch ( str, expr1 [ , expr2,...exprN ])
```

alt

alt 函数用于返回首个具有有效数表示法的参数。如果未找到此类匹配，则将返回最后一个参数。可使用任何数目的参数。

语法：

```
alt(expr1[ , expr2 , expr3 , ...] , else)
```

参数：

参数

参数	说明
expr1	用于检查有效的数字呈现形式的第一个表达式。
expr2	用于检查有效的数字呈现形式的第二个表达式。
expr3	用于检查有效的数字呈现形式的第三个表达式。
else	如果先前的参数都不包含有效的数字呈现形式时返回的值。

alt 函数通常与数字或日期解析函数一起使用。这样，Qlik Sense 就可以以优先顺序测试不同的日期格式。它还可用于处理数字表达式中的 NULL 值。

示例：

示例

示例	结果

`alt( date#( dat , 'YYYY/MM/DD' ),`
`date#( dat , 'MM/DD/YYYY' ),`
`date#( dat , 'MM/DD/YY' ),`
`'No valid date' )`
	此表达式将测试日期字段是否包含三个指定日期格式中的任一日期。如果是这样，它将返回包含原始字符串和有效的日期数字呈现形式的双重值。如果未找到匹配，将返回文本 'No valid date'(无任何有效的数字呈现形式) 。

`alt(Sales,0) + alt(Margin,0)`	此表达式添加了字段 Sales 和 Margin，用于将所有缺失值 (NULL) 替换为 0。

class

class 函数用于将第一个参数赋值给类别间隔。结果是一个 a<=x<b 的双值作为文本值，其中 a 和 b 为 bin 的上限值和下限值，且下界为数值。

语法：

```
class(expression, interval [ , label [ , offset ]])
```

参数：

参数

参数	说明
interval	指定 bin 宽的一个数字。
label	可替换结果文本中的“x”的任意字符串。
offset	可用作分类的默认起始点偏移量的一个数字。默认起始点通常为 0。

示例：

示例

示例	结果
class(var,10) 且 var = 23	返回 '20<=x<30'
class(var,5,'value') 且 var = 23	返回 '20<= value <25'
class(var,10,'x',5) 且 var = 23	返回 '15<=x<25'

示例 - 使用 class 加载脚本

示例：加载脚本

加载脚本

在此例中，我们加载包含人员的姓名和年龄的表格。我们想要添加一个用来根据以十年为时间间隔的年龄组对每位人员进行分类的字段。原始源表如下所示。

结果

Name	Age
John	25
Karen	42
Yoshi	53

要添加年龄组分类字段，您可以使用 **class** 函数添加前置 Load 语句。

在数据加载编辑器中创建一个新选项卡，然后将以下数据作为内联加载加载。在 Qlik Sense 中创建下面的表格以查看结果。

```
 LOAD *, class(Age, 10, 'age') As Agegroup;  LOAD * INLINE [ Age, Name 25, John 42, Karen 53, Yoshi];
```

结果

结果

Name	Age	Agegroup
John	25	20 <= age < 30
Karen	42	40 <= age < 50
Yoshi	53	50 <= age < 60

coalesce

coalesce 函数用于返回具有有效 non-NULL 表示法的参数中的第一个。可使用任何数目的参数。

语法：

```
coalesce(expr1[ , expr2 , expr3 , ...])
```

参数：

参数

参数	说明
expr1	用于检查有效的非空呈现形式的第一个表达式。
expr2	用于检查有效的非空呈现形式的第二个表达式。
expr3	用于检查有效的非空呈现形式的第三个表达式。

示例：

示例

示例	结果
	该表达式将字段的所有 NULL 值更改为 'N/A'。
`Coalesce(ProductDescription, ProductName, ProductCode, 'no description available')`	此表达式将在三个不同的产品描述字段之间进行选择，因为某些字段可能没有产品的值。将按给定的顺序返回具有非空值的第一个字段。如果所有字段都不包含值，则结果将为“无可用描述”。

示例	结果
Coalesce(TextBetween(FileName, '"', '"'), FileName)	此表达式将从字段 *FileName* 中删除可能的括引号。如果给定的 *FileName* 是带引号的，则删除它们，并返回包含的、不带引号的 *FileName*。如果 *TextBetween* 函数没有找到分隔符，它将返回 null，**Coalesce** 将拒绝该值，而是返回原始 *FileName*。

if

if 函数用于返回一个值，具体取决于函数提供的条件的计算结果是否为 True 或 False。

语法：

if(condition , then [, else]**)**

if 函数有三个参数：*condition*、*then* 和 *else*，都是表达式。其他两个（*then* 和 *else*）可为任何类型。

参数

参数	说明
condition	进行逻辑解释的表达式。
then	可为任何类型的表达式。如果 *condition* 是 True，则 if 函数返回 *then* 表达式的值。
else	可为任何类型的表达式。如果 *condition* 是 False，则 if 函数返回 *else* 表达式的值。 该参数为可选。如果 *condition* 为 False，并且未指定 else，则会返回 NULL。

示例

示例	结果
if(Amount>= 0, 'OK', 'Alarm')	此表达式测试数量是否是一个正数（0 或更大），如果是，则返回'OK'。如果数量小于 0，则返回 'Alarm'。

示例 - 使用 if 加载脚本

示例：加载脚本

加载脚本

If 可在加载脚本中与其他方法和对象一起使用，包括变量。例如，如果设置变量 *threshold* 并且希望基于该阈值在数据模型中包括字段，您可以执行以下操作。

在数据加载编辑器中创建一个新选项卡，然后将以下数据作为内联加载加载。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions: Load * Inline [ transaction_id, transaction_date, transaction_amount,
transaction_quantity, customer_id, size, color_code 3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange 3752, 20180916, 5.75, 1, 5646471, S, blue 3753,
20180922, 125.00, 7, 3036491, l, Black 3754, 20180922, 484.21, 13, 049681, xs, Red 3756,
20180922, 59.18, 2, 2038593, M, Blue 3757, 20180923, 177.42, 21, 203521, XL, Black ];   set
```

```
threshold = 100;                                          /* Create new table called Transaction_Buckets  Comp
amount field from Transaction table to threshold of 100. Output results into a new field
called Compared to Threshold                                                          */
If(transaction_amount > $(threshold),'Greater than $(threshold)','Less than $(threshold)') as
[Compared to Threshold] Resident Transactions;
```

结果

Qlik Sense 表显示了在加载脚本中使用 *if* 函数的输出。

transaction_id	与阈值比较
3750	小于 100
3751	大于 100
3752	小于 100
3753	大于 100
3754	大于 100
3756	小于 100
3757	大于 100

示例 - 使用 if 的图表表达式

示例:图表表达式

图表表达式 1

加载脚本

在数据加载编辑器中创建一个新选项卡，然后将以下数据作为内联加载加载。加载数据后，在 Qlik Sense 表格中创建下面的图表表达式示例。

```
MyTable: LOAD * inline [Date, Location, Incidents 1/3/2016, Beijing, 0 1/3/2016, Boston, 12
1/3/2016, Stockholm, 3 1/3/2016, Toronto, 0 1/4/2016, Beijing, 0 1/4/2016, Boston, 8];
```

Qlik Sense 表以图表表达式显示了 *if* 函数的示例。

日期	位置	事件	if(Incidents>=10, 'Critical', 'Ok')	if(Incidents>=10, 'Critical', If(Incidents>=1 and Incidents<10, 'Warning', 'Ok'))
1/3/2016	Beijing	0	Ok	OK
1/3/2016	Boston	12	Critical	Critical
1/3/2016	Stockholm	3	OK	Warning
1/3/2016	Toronto	0	Ok	Ok
1/4/2016	Beijing	0	Ok	Ok
1/4/2016	Boston	8	Ok	警告

图表表达式 2

在新应用程序中，在数据加载编辑器的新选项卡中添加以下脚本，然后加载数据。然后可以使用下面的图表表达式创建表格。

```
SET FirstWeekDay=0; Load  Date(MakeDate(2022)+RecNo()-1) as Date Autogenerate 14;
```

Qlik Sense 表以图表表达式显示了 *if* 函数的示例。

日期	WeekDay(Date)	If(WeekDay (Date)>=5,'WeekEnd','Normal Day')
1/1/2022	Sat	WeekEnd
1/2/2022	Sun	WeekEnd
1/3/2022	Mon	Normal Day
1/4/2022	Tue	Normal Day
1/5/2022	Wed	Normal Day
1/6/2022	Thu	Normal Day
1/7/2022	Fri	Normal Day
1/8/2022	Sat	WeekEnd
1/9/2022	Sun	WeekEnd
1/10/2022	Mon	Normal Day
1/11/2022	Tue	Normal Day
1/12/2022	Wed	Normal Day
1/13/2022	Thu	Normal Day
1/14/2022	Fri	Normal Day

match

match 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数值位置。比较区分大小写。

语法：

```
match( str, expr1 [ , expr2,...exprN ])
```

*如果您要使用不区分大小写的比较，可以使用 **mixmatch** 函数。如果您要使用不区分大小写的比较和通配符，可以使用 **wildmatch** 函数。*

示例:使用 match 加载脚本

示例:加载脚本

加载脚本

您可使用 match 以加载数据的子集。例如,您可为函数中的表达式返回数值。然后您可根据数值限制加载的数据。如果没有匹配,Match 返回 0。在该示例中不匹配的所有表达式将因此返回 0 并且通过 WHERE 语句从数据加载排除。

在数据加载编辑器中创建一个新选项卡,然后将以下数据作为内联加载加载。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions: Load * Inline [ transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size, color_code 3750, 20180830, 23.56, 2, 2038593, L, Red 3751, 20180907, 556.31, 6, 203521, m, orange 3752, 20180916, 5.75, 1, 5646471, S, blue 3753, 20180922, 125.00, 7, 3036491, l, Black 3754, 20180922, 484.21, 13, 049681, xs, Red 3756, 20180922, 59.18, 2, 2038593, M, Blue 3757, 20180923, 177.42, 21, 203521, XL, Black ];   /* Create new table called Transaction_Buckets Create new fields called Customer, and Color code - Blue and Black Load Transactions table. Match returns 1 for 'Blue', 2 for 'Black'.  Does not return a value for 'blue' because match is case sensitive. Only values that returned numeric value greater than 0  are loaded by WHERE statment into Transactions_Buckets table. */ Transaction_Buckets: Load  customer_id,  customer_id as [Customer],   color_code as [Color Code Blue and Black] Resident Transactions Where match(color_code,'Blue','Black') > 0;
```

结果

Qlik Sense 表显示了在加载脚本中使用 match 函数的输出

Color Code Blue and Black	Customer
Black	203521
Black	3036491
Blue	2038593

示例 - 使用 match 的图表表达式

示例:图表表达式

图表表达式 1

加载脚本

在数据加载编辑器中创建一个新选项卡,然后将以下数据作为内联加载加载。加载数据后,在 Qlik Sense 表格中创建下面的图表表达式示例。

```
MyTable: Load * inline [Cities, Count Toronto, 123 Toronto, 234 Toronto, 231 Boston, 32 Boston, 23 Boston, 1341 Beijing, 234 Beijing, 45 Beijing, 235 Stockholm, 938 Stockholm, 39 Stockholm, 189 zurich, 2342 zurich, 9033 zurich, 0039];
```

下面表格中的第一个表达式为 Stockholm 返回 0，因为 'Stockholm' 未包括在 **match** 函数中的表达式列表内。它也为 'Zurich' 返回 0，因为 **match** 比较要区分大小写。

Qlik Sense 表以图表表达式显示了 *match* 函数的示例

Cities	match(Cities,'Toronto','Boston','Beijing','Zurich')	match(Cities,'Toronto','Boston','Beijing','Stockholm','zurich')
Beijing	3	3
Boston	2	2
Stockholm	0	4
Toronto	1	1
zurich	0	5

图表表达式 2

您可使用匹配来为表达式执行自定义排序。

默认情况下，列按数字或字母排序，具体取决于数据。

Qlik Sense 表格示出默认排序顺序的示例

Cities
Beijing
Boston
Stockholm
Toronto
zurich

要更改顺序，执行以下操作：

1. 在**属性**面板中为您的图表打开**排序**部分。
2. 为您要进行自定义排序的列关闭自动排序。
3. 取消选择**按数字排序**以及**按字母排序**。
4. 选择**排序表达式**，然后输入和以下相似的表达式：
   `=match( Cities, 'Toronto','Boston','Beijing','Stockholm','zurich')`
   Cities 列上的排序顺序更改。

Qlik Sense 表格示出使用 *match* 函数更改排序顺序的示例

Cities
Toronto
Boston

Cities
Beijing
Stockholm
zurich

您还可查看返回的数值。

Qlik Sense 表示出从 *match* 函数返回的数值的示例

Cities	Cities & ' - ' & match (Cities, 'Toronto','Boston', 'Beijing','Stockholm','zurich')
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

mixmatch

mixmatch 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数值位置。比较不区分大小写。

语法：

```
mixmatch( str, expr1 [ , expr2,...exprN ])
```

如果您要改为使用区分大小写的比较，可以使用 **match** 函数。如果您要使用不区分大小写的比较和通配符，可以使用 **wildmatch** 函数。

示例 - 使用 mixmatch 加载脚本

示例：加载脚本

加载脚本

您可使用 mixmatch 以加载数据的子集。例如，您可为函数中的表达式返回数值。然后您可根据数值限制加载的数据。如果没有匹配，Mixmatch 返回 0。在该示例中不匹配的所有表达式将因此返回 0 并且通过 WHERE 语句从数据加载排除。

在数据加载编辑器中创建一个新选项卡，然后将以下数据作为内联加载加载。在 Qlik Sense 中创建下面的表格以查看结果。

```
Load * Inline [ transaction_id, transaction_date, transaction_amount, transaction_quantity,
customer_id, size, color_code 3750, 20180830, 23.56, 2, 2038593, L, Red 3751, 20180907,
556.31, 6, 203521, m, orange 3752, 20180916, 5.75, 1, 5646471, S, blue 3753, 20180922, 125.00,
7, 3036491, l, Black 3754, 20180922, 484.21, 13, 049681, xs, Red 3756, 20180922, 59.18, 2,
2038593, M, Blue 3757, 20180923, 177.42, 21, 203521, XL, Black ];  /* Create new table called
Transaction_Buckets Create new fields called Customer, and Color code - Black, Blue, blue Load
```

```
Transactions table. Mixmatch returns 1 for 'Black', 2 for 'Blue'.  Also returns 3 for 'blue'
because mixmatch is not case sensitive. Only values that returned numeric value greater than 0
 are loaded by WHERE statement into Transactions_Buckets table. */  Transaction_Buckets: Load
  customer_id,     customer_id as [Customer],    color_code as [Color Code - Black, Blue,
blue] Resident Transactions Where mixmatch(color_code,'Black','Blue') > 0;
```

结果

Qlik Sense 表显示了在加载脚本中使用 mixmatch 函数的输出。

Color Code Black, Blue, blue	Customer
Black	203521
Black	3036491
Blue	2038593
blue	5646471

示例 - 使用 mixmatch 的图表表达式

示例:图表表达式
在数据加载编辑器中创建一个新选项卡,然后将以下数据作为内联加载加载。加载数据后,在 Qlik Sense 表格中创建下面的图表表达式示例。

图表表达式 1

```
MyTable: Load * inline [Cities, Count Toronto, 123 Toronto, 234 Toronto, 231 Boston, 32
Boston, 23 Boston, 1341 Beijing, 234 Beijing, 45 Beijing, 235 Stockholm, 938 Stockholm, 39
Stockholm, 189 zurich, 2342 zurich, 9033 zurich, 0039];
```

下面表格中的第一个表达式为 Stockholm 返回 0,因为 'Stockholm' 未包括在 **mixmatch** 函数中的表达式列表内。它为 'Zurich' 返回 4,因为 **mixmatch** 比较要区分大小写。

Qlik Sense 表以图表表达式显示了 *mixmatch* 函数的示例

Cities	mixmatch(Cities,'Toronto','Boston','Beijing','Zurich')	mixmatch(Cities,'Toronto','Boston','Beijing','Stockholm','Zurich')
Beijing	3	3
Boston	2	2
Stockholm	0	4
Toronto	1	1
zurich	4	5

图表表达式 2

您可使用 mixmatch 来为表达式执行自定义排序。

默认情况下，列按数字或字母排序，具体取决于数据。

Qlik Sense 表格示出默认排序顺序的示例

Cities
Beijing
Boston
Stockholm
Toronto
zurich

要更改顺序，执行以下操作：

1. 在**属性**面板中为您的图表打开**排序**部分。
2. 为您要进行自定义排序的列关闭自动排序。
3. 取消选择**按数字排序**以及**按字母排序**。
4. 选择**排序表达式**，然后输入以下表达式：
   `=mixmatch( Cities, 'Toronto','Boston','Beijing','Stockholm','Zurich')`
   Cities 列上的排序顺序更改。

Qlik Sense 表格示出使用 *mixmatch* 函数更改排序顺序的示例。

Cities
Toronto
Boston
Beijing
Stockholm
zurich

您还可查看返回的数值。

Qlik Sense 表示出从 *mixmatch* 函数返回的数值的示例。

Cities	Cities & ' - ' & mixmatch (Cities, 'Toronto','Boston', 'Beijing','Stockholm','Zurich')
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

pick

pick 函数用于返回列表中的第 *n* 个表达式。

语法：

```
pick(n, expr1[ , expr2,...exprN])
```

参数：

参数

参数	说明
n	*n* 是介于 1 和 N 之间的整数。

示例：

示例

示例	结果
pick(N, 'A','B',4, 6)	返回 'B', 如果 N = 2 返回 4, 如果 N = 3

wildmatch

wildmatch 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数量。它允许在比较字符串中使用通配字符(***** 和 **?**)。***** 匹配任何字符序列。**?** 匹配任意单个字符。比较不区分大小写。

语法：

```
wildmatch( str, expr1 [ , expr2,...exprN ])
```

如果您想要使用比较而不使用通配符，可以使用 **match** 或 **mixmatch** 函数。

示例：使用 wildmatch 加载脚本

示例：加载脚本

加载脚本

您可使用 wildmatch 以加载数据的子集。例如，您可为函数中的表达式返回数值。然后您可根据数值限制加载的数据。如果没有匹配，Wildmatch 返回 0。在该示例中不匹配的所有表达式将因此返回 0 并且通过 WHERE 语句从数据加载排除。

在数据加载编辑器中创建一个新选项卡，然后将以下数据作为内联加载加载。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions: Load * Inline [ transaction_id, transaction_date, transaction_amount,
transaction_quantity, customer_id, size, color_code 3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange 3752, 20180916, 5.75, 1, 5646471, S, blue 3753,
20180922, 125.00, 7, 3036491, l, Black 3754, 20180922, 484.21, 13, 049681, xs, Red 3756,
20180922, 59.18, 2, 2038593, M, Blue 3757, 20180923, 177.42, 21, 203521, XL, Black ];   /*
Create new table called Transaction_Buckets Create new fields called Customer, and Color code
- Black, Blue, blue, red Load Transactions table. Wildmatch returns 1 for 'Black', 'Blue', and
```

```
'blue', and 2 for 'Red'.  Only values that returned numeric value greater than 0  are loaded
by WHERE statement into Transactions_Buckets table. */   Transaction_Buckets: Load
customer_id,    customer_id as [Customer],    color_code as [Color Code Black, Blue, blue,
Red] Resident Transactions Where wildmatch(color_code,'Bl*','R??') > 0;
```

结果

Qlik Sense 表显示了在加载脚本中使用 *wildmatch* 函数的输出

Color Code Black, Blue, blue, Red	Customer
Black	203521
Black	3036491
Blue	2038593
blue	5646471
Red	049681
Red	2038593

示例:使用 wildmatch 的图表表达式

示例:图表表达式

图表表达式 1

在数据加载编辑器中创建一个新选项卡,然后将以下数据作为内联加载加载。加载数据后,在 Qlik Sense 表格中创建下面的图表表达式示例。

```
MyTable: Load * inline [Cities, Count Toronto, 123 Toronto, 234 Toronto, 231 Boston, 32
Boston, 23 Boston, 1341 Beijing, 234 Beijing, 45 Beijing, 235 Stockholm, 938 Stockholm, 39
Stockholm, 189 zurich, 2342 zurich, 9033 zurich, 0039];
```

下面表格中的第一个表达式为 Stockholm 返回 0,因为 'Stockholm' 未包括在 **wildmatch** 函数中的表达式列表内。它还为 'Boston' 返回 0,因为 ? 仅在单个字符上匹配。

Qlik Sense 表以图表表达式显示了 *wildmatch* 函数的示例

Cities	wildmatch(Cities,'Tor*','?ton','Beijing','*urich')	wildmatch(Cities,'Tor*','???ton','Beijing','Stockholm','*urich')
Beijing	3	3
Boston	0	2
Stockholm	0	4
Toronto	1	1
zurich	4	5

图表表达式 2

您可使用 wildmatch 来为表达式执行自定义排序。

默认情况下，列按数字或字母排序，具体取决于数据。

Qlik Sense 表格示出默认排序顺序的示例

Cities
Beijing
Boston
Stockholm
Toronto
zurich

要更改顺序，执行以下操作：

1. 在**属性**面板中为您的图表打开**排序**部分。
2. 为您要进行自定义排序的列关闭自动排序。
3. 取消选择**按数字排序**以及**按字母排序**。
4. 选择**排序表达式**，然后输入和以下相似的表达式：
   `=wildmatch( Cities, 'Tor*','???ton','Beijing','Stockholm','*urich')`
   Cities 列上的排序顺序更改。

Qlik Sense 表格示出使用 *wildmatch* 函数更改排序顺序的示例。

Cities
Toronto
Boston
Beijing
Stockholm
zurich

您还可查看返回的数值。

Qlik Sense 表示出从 *wildmatch* 函数返回的数值的示例

Cities	Cities & ' - ' & wildmatch (Cities, 'Tor*','???ton','Beijing','Stockholm','*urich')
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

5.6 计数函数

本部分介绍数据加载脚本中与 **LOAD** 语句评估期间的记录计数器相关的函数。唯一可用于图表表达式的函数是 **RowNo()**。

某些计数器函数没有任何参数，但是它后面的括号不能省略。

计数函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

autonumber

此脚本函数用于返回在脚本执行期间 *expression* 遇到的每个特殊计算值的整数值。此函数可用于创建复合键的紧凑记忆呈现形式。

```
autonumber (expression[ , AutoID])
```

autonumberhash128

此脚本函数用于计算合并输入表达式值的 128 位散列，并返回脚本执行期间遇到的每个特殊散列值的唯一整数值。例如，此函数可用于创建复合键的紧凑记忆呈现形式。

```
autonumberhash128 (expression {, expression})
```

autonumberhash256

此脚本函数用于计算合并输入表达式值的 256 位散列，并返回脚本执行期间遇到的每个特殊散列值的唯一整数值。此函数可用于创建复合键的紧凑记忆呈现形式。

```
autonumberhash256 (expression {, expression})
```

IterNo

此脚本函数用于返回一个整数，表明何时计算带 **while** 子句的 **LOAD** 语句中的单个记录的值。第一次迭代的编号为 1。**IterNo** 函数只有与 **while** 子句结合使用才有意义。

```
IterNo ( )
```

RecNo

此脚本函数用于返回当前表格的当前读取行数。第一个记录为编号 1。

```
RecNo ( )
```

RowNo - script function

此函数用于返回结果 Qlik Sense 内部表格中当前行位置的整数。第一行为编号 1。

```
RowNo ( )
```

RowNo - chart function

RowNo() 用于返回表格中当前列段数据的当前行数。对于位图图表，**RowNo()** 用于返回图表的等效垂直表内的当前行数。

```
RowNo - 图表函数([TOTAL])
```

autonumber

此脚本函数用于返回在脚本执行期间 *expression* 遇到的每个特殊计算值的整数值。此函数可用于创建复合键的紧凑记忆呈现形式。

> *您只能连接已在同一数据加载过程中生成的 **autonumber** 关键字段，作为根据读取表格的顺序所生成的整数。如果您需要使用在两次数据加载过程中保持一致且独立于源数据排序的关键字段，应使用 **hash128**、**hash160** 或 **hash256** 函数。*

语法：

```
autonumber(expression[ , AutoID])
```

参数：

参数	说明
AutoID	为创建多个计数实例，如果 **autonumber** 函数用于脚本内不同的字段，则可选参数 *AutoID* 将用于命名每个计数。

示例：创建合成钥段

在此例中，我们使用 **autonumber** 函数创建合成钥段以节省内存。尽管此例主要用于演示目的，但对于包含大量行的表格将非常有意义。

示例数据

Region	Year	Month	Sales
North	2014	May	245
North	2014	May	347
North	2014	June	127
South	2014	June	645
South	2013	May	367
South	2013	May	221

使用内联数据加载源数据。然后，我们添加前置 Load，用来从 Region、Year 和 Month 字段创建合成钥段。

```
RegionSales:
LOAD *,
AutoNumber(Region&Year&Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Sales
North, 2014,   May,    245
North, 2014,   May,    347
```

```
North, 2014,   June,   127
South, 2014,   June,   645
South, 2013,   May, 367
South, 2013,   May,    221
];
```

最终生成的表格如下所示：

结果表

Region	Year	Month	Sales	RYMkey
North	2014	May	245	1
North	2014	May	347	1
North	2014	June	127	2
South	2014	June	645	3
South	2013	May	367	4
South	2013	May	221	4

在此例中，如果您需要链接到其他表格，可以引用 RYMkey(例如 1)，而不是字符串 “North2014May”。

现在，我们可以通过类似方式加载成本的源表格。在前置 Load 中已排除 Region、Year 和 Month 字段以避免创建合成关键字段，因为我们已经使用 **autonumber** 函数创建复合关键字段用于链接表格。

```
RegionCosts:
LOAD Costs,
AutoNumber(Region&Year&Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013,   May,    167
North, 2014,   May,    56
North, 2014,   June,   199
South, 2014,   June,   64
South, 2013,   May, 172
South, 2013,   May,    126
];
```

现在，我们可以将表格可视化添加到工作表，并添加 Region、Year 和 Month 字段，以及销售额和成本的 Sum 度量。表格将如下所示：

结果表

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals	-	-	1952	784

Region	Year	Month	Sum([Sales])	Sum([Costs])
North	2014	June	127	199
North	2014	May	592	56
South	2014	June	645	64
South	2013	May	588	465

autonumberhash128

此脚本函数用于计算合并输入表达式值的 128 位散列，并返回脚本执行期间遇到的每个特殊散列值的唯一整数值。例如，此函数可用于创建复合键的紧凑记忆呈现形式。

*您只能连接已在同一数据加载过程中生成的 **autonumberhash128** 关键字段，作为根据读取表格的顺序所生成的整数。如果您需要使用在两次数据加载过程中保持一致且独立于源数据排序的关键字段，应使用 **hash128**、**hash160** 或 **hash256** 函数。*

语法：

```
autonumberhash128(expression {, expression})
```

示例：创建合成钥段

在此例中，我们使用 **autonumberhash128** 函数创建合成钥段以节省内存。尽管此例主要用于演示目的，但对于包含大量行的表格将非常有意义。

示例数据

Region	Year	Month	Sales
North	2014	May	245
North	2014	May	347
North	2014	June	127
South	2014	June	645
South	2013	May	367
South	2013	May	221

使用内联数据加载源数据。然后，我们添加前置 Load，用来从 Region、Year 和 Month 字段创建合成钥段。

```
RegionSales:
LOAD *,
AutoNumberHash128(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Sales
North, 2014,   May,    245
North, 2014,   May,    347
```

```
North, 2014,   June,   127
South, 2014,   June,   645
South, 2013,   May, 367
South, 2013,   May,    221
];
```

最终生成的表格如下所示：

结果表

Region	Year	Month	Sales	RYMkey
North	2014	May	245	1
North	2014	May	347	1
North	2014	June	127	2
South	2014	June	645	3
South	2013	May	367	4
South	2013	May	221	4

在此例中，如果您需要链接到其他表格，可以引用 RYMkey(例如 1)，而不是字符串 “North2014May”。

现在，我们可以通过类似方式加载成本的源表格。在前置 Load 中已排除 Region、Year 和 Month 字段以避免创建合成关键字段，因为我们已经使用 **autonumberhash128** 函数创建复合关键字段用于链接表格。

```
RegionCosts:
LOAD Costs,
AutoNumberHash128(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013,   May,    167
North, 2014,   May,    56
North, 2014,   June,   199
South, 2014,   June,   64
South, 2013,   May, 172
South, 2013,   May,    126
];
```

现在，我们可以将表格可视化添加到工作表，并添加 Region、Year 和 Month 字段，以及销售额和成本的 Sum 度量。表格将如下所示：

结果表

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals	-	-	1952	784

Region	Year	Month	Sum([Sales])	Sum([Costs])
North	2014	June	127	199
North	2014	May	592	56
South	2014	June	645	64
South	2013	May	588	465

autonumberhash256

此脚本函数用于计算合并输入表达式值的 256 位散列，并返回脚本执行期间遇到的每个特殊散列值的唯一整数值。此函数可用于创建复合键的紧凑记忆呈现形式。

*您只能连接已在同一数据加载过程中生成的 **autonumberhash256** 关键字段，作为根据读取表格的顺序所生成的整数。如果您需要使用在两次数据加载过程中保持一致且独立于源数据排序的关键字段，应使用 **hash128**、**hash160** 或 **hash256** 函数。*

语法：

```
autonumberhash256(expression {, expression})
```

示例：创建合成钥段

在此例中，我们使用 **autonumberhash256** 函数创建合成钥段以节省内存。尽管此例主要用于演示目的，但对于包含大量行的表格将非常有意义。

示例表格

Region	Year	Month	Sales
North	2014	May	245
North	2014	May	347
North	2014	June	127
South	2014	June	645
South	2013	May	367
South	2013	May	221

使用内联数据加载源数据。然后，我们添加前置 Load，用来从 Region、Year 和 Month 字段创建合成钥段。

```
RegionSales:
LOAD *,
AutoNumberHash256(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Sales
North, 2014,   May,    245
```

```
North, 2014,   May,    347
North, 2014,   June,   127
South, 2014,   June,   645
South, 2013,   May, 367
South, 2013,   May,    221
];
```

最终生成的表格如下所示：

结果表

Region	Year	Month	Sales	RYMkey
North	2014	May	245	1
North	2014	May	347	1
North	2014	June	127	2
South	2014	June	645	3
South	2013	May	367	4
South	2013	May	221	4

在此例中，如果您需要链接到其他表格，可以引用 RYMkey(例如 1)，而不是字符串 “North2014May”。

现在，我们可以通过类似方式加载成本的源表格。在前置 Load 中已排除 Region、Year 和 Month 字段以避免创建合成关键字段，因为我们已经使用 **autonumberhash256** 函数创建复合关键字段用于链接表格。

```
RegionCosts:
LOAD Costs,
AutoNumberHash256(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013,   May,    167
North, 2014,   May,    56
North, 2014,   June,   199
South, 2014,   June,   64
South, 2013,   May, 172
South, 2013,   May,    126
];
```

现在，我们可以将表格可视化添加到工作表，并添加 Region、Year 和 Month 字段，以及销售额和成本的 Sum 度量。表格将如下所示：

结果表

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals	-	-	1952	784

Region	Year	Month	Sum([Sales])	Sum([Costs])
North	2014	June	127	199
North	2014	May	592	56
South	2014	June	645	64
South	2013	May	588	465

IterNo

此脚本函数用于返回一个整数，表明何时计算带 **while** 子句的 **LOAD** 语句中的单个记录的值。第一次迭代的编号为 1。**IterNo** 函数只有与 **while** 子句结合使用才有意义。

语法：

```
IterNo( )
```

示例和结果：

示例：

```
LOAD
   IterNo() as Day,
   Date( StartDate + IterNo() - 1 ) as Date
   While StartDate + IterNo() - 1 <= EndDate;

LOAD * INLINE
[StartDate, EndDate
2014-01-22, 2014-01-26
];
```

该 **LOAD** 语句将为 **StartDate** 和 **EndDate** 定义的范围之间的每个日期生成一条记录。

在此例中，最终生成的表格如下所示：

结果表

Day	Date
1	2014-01-22
2	2014-01-23
3	2014-01-24
4	2014-01-25
5	2014-01-26

RecNo

此脚本函数用于返回当前表格的当前读取行数。第一个记录为编号 1。

语法：

```
RecNo( )
```

与 **RowNo( )** 相比，此函数计算生成的 Qlik Sense 表格中的行数，而 **RecNo( )** 函数计算原始数据表格中的记录数，并且会在将原始数据表格串联至其他表格时进行重置。

示例：数据加载脚本

原始数据表格加载：

```
Tab1:
LOAD * INLINE
[A, B
1, aa
2,cc
3,ee];

Tab2:
LOAD * INLINE
[C, D
5, xx
4,yy
6,zz];
```

加载选定行的记录数和行数：

```
QTab:
LOAD *,
RecNo( ),
RowNo( )
resident Tab1 where A<>2;

LOAD
C as A,
D as B,
RecNo( ),
RowNo( )
resident Tab2 where A<>5;

//We don't need the source tables anymore, so we drop them
Drop tables Tab1, Tab2;
```

所生成的 Qlik Sense 内部表格：

结果表

A	B	RecNo( )	RowNo( )
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

RowNo

此函数用于返回结果 Qlik Sense 内部表格中当前行位置的整数。第一行为编号 1。

语法：

```
RowNo( [TOTAL])
```

与对原始数据表格中记录进行计数的 **RecNo( )** 相反，**RowNo( )** 函数不会对 **where** 子句排除的记录进行计数，并且在原始数据表格串联到其他表格时，该函数不会重置。

*如果使用前置 Load，即从同一表格读取的叠加的 **LOAD** 语句数，则只能在顶部的 **RowNo( )** 语句中使用 **LOAD**。如果在后续的 **LOAD** 语句中使用 **RowNo( )**，则将返回 0。*

示例：数据加载脚本

原始数据表格加载：

```
Tab1:
LOAD * INLINE
[A, B
1, aa
2,cc
3,ee];

Tab2:
LOAD * INLINE
[C, D
5, xx
4,yy
6,zz];
```

加载选定行的记录数和行数：

```
QTab:
LOAD *,
RecNo( ),
RowNo( )
resident Tab1 where A<>2;

LOAD
C as A,
D as B,
RecNo( ),
RowNo( )
resident Tab2 where A<>5;

//We don't need the source tables anymore, so we drop them
Drop tables Tab1, Tab2;
```

所生成的 Qlik Sense 内部表格：

结果表

A	B	RecNo( )	RowNo( )
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

RowNo - 图表函数

RowNo() 用于返回表格中当前列段数据的当前行数。对于位图图表，**RowNo()** 用于返回图表的等效垂直表内的当前行数。

如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

列段数据

*当任何图表表达式使用了 **RowNo()** 时，就不允许在图表中依据 Y 值排序，也不允许在表格中依据表达式数据列排序。因此，这些排序替代项会自动禁用。*

语法：

```
RowNo([TOTAL])
```

返回数据类型： 整数

参数：

参数	说明
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

示例：使用 RowNo 的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
Temp:
 LOAD * inline [ Customer|Product|OrderNumber|UnitSales|UnitPrice Astrida|AA|1|4|16
Astrida|AA|7|10|15 Astrida|BB|4|9|9 Betacab|CC|6|5|10 Betacab|AA|5|2|20 Betacab|BB|1|25| 25
Canutility|AA|3|8|15 Canutility|CC|5|4|19 Divadip|CC|2|4|16 Divadip|DD|3|1|25 ] (delimiter is
'|');
```

图表表达式

在 Qlik Sense 工作表中创建表可视化，以 **Customer** 和 **UnitSales** 为维度。添加 RowNo( ) 和 RowNo (TOTAL) 为度量，分别标记为**段中的行**以及 **Row Number**。在表格中添加以下表达式作为度量：

```
If( RowNo( )=1, 0, UnitSales / Above( UnitSales ))
```

结果

Customer	UnitSales	Row in Segment	Row Number	If(RowNo( )=1, 0, UnitSales / Above(UnitSales))
Astrida	4	1	1	0
Astrida	9	2	2	2.25
Astrida	10	3	3	1.1111111111111
Betacab	2	1	4	0
Betacab	5	2	5	2.5
Betacab	25	3	6	5
Canutility	4	1	7	0
Canutility	8	2	8	2
Divadip	1	1	9	0
Divadip	4	2	10	4

解释

Row in Segment 列显示列段数据的结果 1、2 和 3，该列段数据包含客户 Astrida 的 UnitSales 值。然后，再次从 1 开始为下一个列段数据的行进行编号，即 Betacab。

Row Number 列由于 RowNo() 的参数 TOTAL 而忽略维度，并对表中的行进行计数。

此表达式会为每个列段数据中的第一行返回 0，因此列会显示：

0、2.25、1.1111111、0、2.5、5、0、2、0 和 4。

另请参见：

- *Above - 图表函数 (page 596)*

5.7 日期和时间函数

Qlik Sense 日期和时间函数用于变换和转换日期和时间值。所有函数均可用于数据加载脚本和图表表达式。

这些函数基于日期-时间序列号(等于从 1899 年 12 月 30 日开始的天数)。整数部分表示天数,分数部分表示一天的时间。

Qlik Sense 使用该参数的数值,所以数字即使没有格式化为日期或时间,也可以作为参数的有效值。例如,如果参数与数值不对应(因为它是一个字符串),则 Qlik Sense 会尝试根据日期和时间环境变量解释此字符串。

如果在参数中使用的时间格式与环境变量设置不一致,Qlik Sense 将不会做出正确的解释。要解决此问题,可更改设置或使用解释功能。

在每个函数的示例中,假设默认的时间和日期格式为 hh:mm:ss 和 YYYY-MM-DD (ISO 8601)。

在处理具有日期或时间函数的时间戳时,Qlik Sense 会忽略所有夏令时参数,除非日期或时间函数包含地理位置。

例如,`ConvertToLocalTime( filetime('Time.qvd'), 'Paris')` 将使用夏令时参数,而 `ConvertToLocalTime(filetime('Time.qvd'), 'GMT-01:00')` 将不使用夏令时参数。

日期和时间函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

时间的整数表达式

second

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示秒的整数。

```
second (expression)
```

minute

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示分钟的整数。

```
minute (expression)
```

hour

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示小时的整数。

```
hour (expression)
```

day

此函数用于根据标准数字解释当 **expression** 小数部分被解释为日期时返回一个表示某天的整数。

```
day (expression)
```

week

此函数用于返回根据 ISO 8601 表示周数的整数。周数根据标准数字解释通过表达式的日期解释进行计算。

```
week (expression)
```

month

此函数用于返回包含在环境变量 **MonthNames** 中定义的月份名称的对偶值和一个介于 1 至 12 的整数。月份根据标准数字解释通过表达式的日期解释进行计算。

```
month (expression)
```

year

此函数用于根据标准数字解释当 **expression** 被解释为日期时返回一个表示年份的整数。

```
year (expression)
```

weekyear

此函数用于返回根据 ISO 8601 周数所属的年份。星期数范围在 1 和大约 52 之间。

```
weekyear (expression)
```

weekday

此函数用于返回包含以下名称的对偶值：在环境变量 **DayNames** 中定义的日期名称。介于 0-6 之间的整数对应于一周 (0-6) 的标定天。

```
weekday (date)
```

Timestamp 函数

now

此函数用于返回系统时钟的当前时间的时间戳。默认值为 1。

```
now ([ timer_mode])
```

today

此函数用于返回系统时钟的当前日期。

```
today ([timer_mode])
```

LocalTime

此函数用于返回指定时区的系统时钟的当前时间戳。

```
localtime ([timezone [, ignoreDST ]])
```

Make 函数

makedate

此函数用于返回根据年份 **YYYY**、月份 **MM** 和日期 **DD** 计算的日期。

```
makedate (YYYY [ , MM [ , DD ] ])
```

makeweekdate

此函数用于返回根据年份 **YYYY**、星期 **WW** 和星期几 **D** 计算的日期。

```
makeweekdate (YYYY [ , WW [ , D ] ])
```

maketime

此函数用于返回根据小时 **hh**、分钟 **mm** 和秒 **ss** 计算的时间。

```
maketime (hh [ , mm [ , ss [ .fff ] ] ])
```

其他日期函数

AddMonths

此函数用于返回在 **startdate** 后 **n** 个月内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 个月内发生的日期。

```
addmonths (startdate, n , [ , mode])
```

AddYears

此函数用于返回在 **startdate** 后 **n** 年内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 年内发生的日期。

```
addyears (startdate, n)
```

yeartodate

此函数用于判断输入时间戳是否在最后加载脚本的日期的年份以内，并返回 True(如果在) 或返回 False(如果不在) 。

```
yeartodate (date [ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

Timezone 函数

timezone

此函数用于返回当前时区的名称，如 Windows 所定义。

```
timezone ( )
```

GMT

此函数用于返回来自系统时钟的当前 Greenwich Mean Time 和 Windows 时间设置。

```
GMT ( )
```

UTC

用于返回当前 Coordinated Universal Time。

```
UTC ( )
```

daylightsaving

用于返回如 Windows 所定义的当下为日间省时的调整。

```
daylightsaving ( )
```

converttolocaltime

将 UTC 或 GMT 时间戳转换为本地时间作为对偶值。此地方可为全世界任何一个城市，地方和时区。

```
converttolocaltime (timestamp [, place [, ignore_dst=false]])
```

设置时间函数

setdateyear

此函数用于输入 **timestamp** 和 **year**，并使用在输入中指定的 **year** 更新 **timestamp**。

```
setdateyear  (timestamp, year)
```

setdateyearmonth

此函数用于输入 **timestamp**、**month** 和 **year**，并使用在输入中指定的 **year** 和 **month** 更新 **timestamp**。

```
setdateyearmonth  (timestamp, year, month)
```

In... 函数

inyear

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的年份以内。

```
inyear (date, basedate , shift [, first_month_of_year = 1])
```

inyeartodate

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 为止以及包括 **base_date** 最后毫秒的年份部分以内。

```
inyeartodate  (date, basedate , shift [, first_month_of_year = 1])
```

inquarter

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的季度以内。

```
inquarter  (date, basedate , shift [, first_month_of_year = 1])
```

inquartertodate

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 为止以及包括 **base_date** 最后毫秒的季度部分以内。

```
inquartertodate  (date, basedate , shift [, first_month_of_year = 1])
```

inmonth

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的月份以内。

```
inmonth (date, basedate , shift)
```

inmonthtodate

用于返回 True，如果 **date** 位于包含 **basedate** 为止以及包括 **basedate** 最后毫秒的月份部分以内。

```
inmonthtodate  (date, basedate , shift)
```

inmonths

此函数用于判断时间戳是否位于作为基准日期的同一个月、两个月、季度、四个月或半年以内。另外，也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

```
inmonths (n, date, basedate , shift [, first_month_of_year = 1])
```

inmonthstodate

此函数用于判断时间戳是否位于截止以及包括 **base_date** 的最后毫秒的某个月、两个月、季度、四个月或半年周期的一部分以内。另外，它也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

```
inmonthstodate  (n, date, basedate , shift [, first_month_of_year = 1])
```

inweek

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的星期以内。

```
inweek (date, basedate , shift [, weekstart])
```

inweektodate

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 为止以及包括 **base_date** 最后毫秒的星期部分以内。

```
inweektodate  (date, basedate , shift [, weekstart])
```

inlunarweek

此函数用于判断 **timestamp** 是否位于包含 **base_date** 的阴历周以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
inlunarweek  (date, basedate , shift [, weekstart])
```

inlunarweektodate

此函数用于判断 **timestamp** 是否位于截止以及包括 **base_date** 最后毫秒的阴历周的某部分以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
inlunarweektodate  (date, basedate , shift [, weekstart])
```

inday

此函数用于返回 True，如果 **timestamp** 位于包含 **base_timestamp** 的一天以内。

```
inday (timestamp, basetimestamp , shift [, daystart])
```

indaytotime

此函数用于返回 True，如果 **timestamp** 位于包含 **base_timestamp** 为止以及包括 **base_timestamp** 精确毫秒的日子部分以内。

```
indaytotime (timestamp, basetimestamp , shift [, daystart])
```

Start ... end 函数

yearstart

此函数用于返回与包含 **date** 的年份的第一天的开始时间对应的时间戳。默认的输出格式为在脚本中所设置的 **DateFormat**。

```
yearstart ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearend

此函数用于返回与包含 **date** 的年份的最后一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

```
yearend ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearname

此函数用于返回一个四位数年份的显示值，带有与包含 **date** 的年份的第一天的第一毫秒时间戳对应的基本数值。

```
yearname (date [, shift = 0 [, first_month_of_year = 1]] )
```

quarterstart

此函数用于返回与包含 **date** 的季度的第一毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

```
quarterstart (date [, shift = 0 [, first_month_of_year = 1]])
```

quarterend

此函数用于返回与包含 **date** 的季度的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

```
quarterend (date [, shift = 0 [, first_month_of_year = 1]])
```

quartername

此函数用于返回一个显示值，该值显示季度的月(根据 **MonthNames** 脚本变量的格式)以及年，伴随一个与该季度第一天第一毫秒的时间戳对应的基础数值。

```
quartername (date [, shift = 0 [, first_month_of_year = 1]])
```

monthstart

此函数用于返回与包含 **date** 的月份的第一天的第一毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

```
monthstart (date [, shift = 0])
```

monthend

此函数用于返回与包含 **date** 的月份的最后一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

```
monthend (date [, shift = 0])
```

monthname

此函数用于返回一个显示值，该值显示该月(根据 **MonthNames** 脚本变量的格式)以及年，伴随一个与该月第一天第一毫秒的时间戳对应的基础数值。

```
monthname (date [, shift = 0])
```

monthsstart

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

```
monthsstart (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsend

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的最后毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

```
monthsend (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsname

此函数用于返回一个显示值，表示时段各月份(根据 **MonthNames** 脚本变量的格式)和年的范围。基础数值与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应。

```
monthsname (n, date [, shift = 0 [, first_month_of_year = 1]])
```

weekstart

此函数用于返回与包含 **date** 的日历周的第一天(周一)的第一毫秒时间戳对应的值。默认输出格式是在脚本中设置的 **DateFormat**。

```
weekstart (date [, shift = 0 [,weekoffset = 0]])
```

weekend

此函数用于返回一个与包含 **date** 的日历周的最后一日(周日)最后一毫秒时间戳对应的值。默认输出格式为在脚本中设置的 **DateFormat**。

```
weekend (date [, shift = 0 [,weekoffset = 0]])
```

weekname

此函数用于返回一个值，显示带有与包含 **date** 的周的第一天的第一毫秒时间戳对应的基本数值对应的年份和周数。

```
weekname (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekstart

此函数用于返回与包含 **date** 的阴历周的第一毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
lunarweekstart (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekend

此函数用于返回与包含 **date** 的阴历周的最后毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
lunarweekend (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekname

此函数用于返回一个显示值，显示与包含 **date** 的阴历周的第一天的第一毫秒时间戳对应的年份和阴历周数。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
lunarweekname (date [, shift = 0 [,weekoffset = 0]])
```

daystart

此函数用于返回与 **time** 参数中包含的一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **TimestampFormat**。

```
daystart (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayend

此函数用于返回与 **time** 中包含的一天的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **TimestampFormat**。

```
dayend (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayname

此函数用于返回一个值，显示与包含 **time** 当天第一毫秒的时间戳对应的基本数值的日期。

```
dayname (timestamp [, shift = 0 [, dayoffset = 0]])
```

Day numbering 函数

age

age 函数用于返回某人在 **date_of_birth** 出生的 **timestamp**(完整年份) 时的年龄。

```
age (timestamp, date_of_birth)
```

networkdays

networkdays 函数用于返回工作日的编号 (周一至周五) ，在 **start_date** 和 **end_date** 之间，并将任何列出的可选 **holiday**考虑在内。

```
networkdays  (start:date, end_date {, holiday})
```

firstworkdate

firstworkdate 函数用于返回最近的起始日以获得 **no_of_workdays**(周一至周五) ，将任何列出的可选节假日考虑在内，不迟于 **end_date**。**end_date** 和 **holiday** 应为有效的日期或时间戳。

```
firstworkdate (end_date, no_of_workdays {, holiday} )
```

lastworkdate

lastworkdate 函数用于返回最早的结束日以获得 **no_of_workdays**(周一至周五) ，如果在 **start_date** 开始考虑任何列出的可选 **holiday**。**start_date** 和 **holiday** 应是有效的日期或时间戳。

```
lastworkdate (start_date, no_of_workdays {, holiday})
```

daynumberofyear

此函数用于计算时间戳所属的年份的天数。从该年度的第一天的第一毫秒开始计算，但可以偏移第一个月。

```
daynumberofyear (date[,firstmonth])
```

daynumberofquarter

此函数用于计算时间戳所属的季度的天数。

```
daynumberofquarter (date[,firstmonth])
```

addmonths

此函数用于返回在 **startdate** 后 **n** 个月内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 个月内发生的日期。

语法：

```
AddMonths(startdate, n , [ , mode])
```

返回数据类型： 双

AddMonths 函数返回同时包含字符串和数字值的双重值。该函数会获取输入表达式的数值，然后生成一个表示此数字的字符串。字符串用于显示，而数值则用于所有数值计算和排序。

参数：

参数

参数	说明
startdate	作为时间戳的开始日期，例如“2012-10-12”。
n	作为正整数或负整数的月份数量。
mode	指定是相对每月的开头还是相对每月的结尾添加月份。对于相对于月份开头的添加，默认模式为 0。为相对于月份末尾的添加将模式设置为 1。当模式设置为 1 时并且输入日期为 28 号或以上时，该功能检查要从开始日期达到月末还剩余多少天。到达月末的相同天数设置在返回的日期上。

示例和结果：

脚本示例

示例	结果
addmonths ('2003-01-29',3)	返回“2003-04-29”
addmonths ('2003-01-29',3,0)	返回“2003-04-29”
addmonths ('2003-01-29',3,1)	返回“2003-04-28”
addmonths ('2003-01-29',1,0)	返回“2003-02-28”

示例	结果
addmonths ('2003-01-29',1,1)	返回“2003-02-26”
addmonths ('2003-02-28',1,0)	返回“2003-03-28”
addmonths ('2003-02-28',1,1)	返回“2003-03-31”
addmonths ('2003-01-29',-3)	返回“2002-10-29”

addyears

此函数用于返回在 **startdate** 后 **n** 年内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 年内发生的日期。

语法：

```
AddYears(startdate, n)
```

返回数据类型：双

参数：

参数

参数	说明
startdate	作为时间戳的开始日期，例如“2012-10-12”。
n	作为正整数或负整数的年份数量。

示例和结果：

脚本示例

示例	结果
addyears ('2010-01-29',3)	返回“2013-01-29”
addyears ('2010-01-29',-1)	返回“2009-01-29”

age

age 函数用于返回某人在 **date_of_birth** 出生的 **timestamp**（完整年份）时的年龄。

语法：

```
age(timestamp, date_of_birth)
```

可以是表达式。

返回数据类型：数字

参数：

参数

参数	说明
timestamp	时间戳或用于对时间戳求值的表达式都可用于计算完成的年份数量。
date_of_birth	正在计算其年龄的人的出生日期。可以是表达式。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
age('25/01/2014', '29/10/2012')	返回 1。
age('29/10/2014', '29/10/2012')	返回 2。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Employees:
LOAD * INLINE [
Member|DateOfBirth
John|28/03/1989
Linda|10/12/1990
Steve|5/2/1992
Birg|31/3/1993
Raj|19/5/1994
Prita|15/9/1994
Su|11/12/1994
Goran|2/3/1995
Sunny|14/5/1996
Ajoa|13/6/1996
Daphne|7/7/1998
Biffy|4/8/2000
] (delimiter is |);
AgeTable:
Load *,
age('20/08/2015', DateOfBirth) As Age
Resident Employees;
Drop table Employees;
```

结果列表显示了为表格中的每条记录返回的 age 值。

结果表

Member	DateOfBirth	Age
John	28/03/1989	26
Linda	10/12/1990	24
Steve	5/2/1992	23
Birg	31/3/1993	22
Raj	19/5/1994	21
Prita	15/9/1994	20
Su	11/12/1994	20
Goran	2/3/1995	20
Sunny	14/5/1996	19
Ajoa	13/6/1996	19
Daphne	7/7/1998	17
Biffy	4/8/2000	15

converttolocaltime

将 UTC 或 GMT 时间戳转换为本地时间作为对偶值。此地方可为全世界任何一个城市，地方和时区。

语法：

```
ConvertToLocalTime(timestamp [, place [, ignore_dst=false]])
```

返回数据类型：双

参数：

参数

参数	说明
timestamp	时间戳或对时间戳求值的表达式都可用于转换。

参数	说明
place	下面的有效地方和时区表格中的地方或时区。或者，可以使用 GMT 或 UTC 定义本地时间。以下值和时间偏移量范围有效： • GMT • GMT-12:00 - GMT-01:00 • GMT+01:00 - GMT+14:00 • UTC • UTC-12:00 - UTC-01:00 • UTC+01:00 - UTC+14:00 *只能使用标准时间偏移量。不能使用任意时间偏移量，例如，GMT-04:27。*
ignore_dst	如果要忽略 DST(夏令时)，则设置为 True。

结果时间调整为夏令时，除非将 **ignore_dst** 设置为 True。

有效的地方和时区

A-C	D-K	L-R	S-Z
Abu Dhabi	Darwin	La Paz	Samoa
Adelaide	Dhaka	Lima	Santiago
Alaska	Eastern Time (US & Canada)	Lisbon	Sapporo
Amsterdam	Edinburgh	Ljubljana	Sarajevo
Arizona	Ekaterinburg	London	Saskatchewan
Astana	Fiji	Madrid	Seoul
Athens	Georgetown	Magadan	Singapore
Atlantic Time (Canada)	Greenland	Mazatlan	Skopje
Auckland	Greenwich Mean Time : Dublin	Melbourne	Sofia
Azores	Guadalajara	Mexico City	Solomon Is.
Baghdad	Guam	Mid-Atlantic	Sri Jayawardenepura
Baku	Hanoi	Minsk	St. Petersburg
Bangkok	Harare	Monrovia	Stockholm
Beijing	Hawaii	Monterrey	Sydney

A-C	D-K	L-R	S-Z
Belgrade	Helsinki	Moscow	Taipei
Berlin	Hobart	Mountain Time (US & Canada)	Tallinn
Bern	Hong Kong	Mumbai	Tashkent
Bogota	Indiana (East)	Muscat	Tbilisi
Brasilia	International Date Line West	Nairobi	Tehran
Bratislava	Irkutsk	New Caledonia	Tokyo
Brisbane	Islamabad	New Delhi	Urumqi
Brussels	Istanbul	Newfoundland	Warsaw
Bucharest	Jakarta	Novosibirsk	Wellington
Budapest	Jerusalem	Nuku'alofa	West Central Africa
Buenos Aires	Kabul	Osaka	Vienna
Cairo	Kamchatka	Pacific Time (US & Canada)	Vilnius
Canberra	Karachi	Paris	Vladivostok
Cape Verde Is.	Kathmandu	Perth	Volgograd
Caracas	Kolkata	Port Moresby	Yakutsk
Casablanca	Krasnoyarsk	Prague	Yerevan
Central America	Kuala Lumpur	Pretoria	Zagreb
Central Time (US & Canada)	Kuwait	Quito	-
Chennai	Kyiv	Riga	-
Chihuahua	-	Riyadh	-
Chongqing	-	Rome	-
Copenhagen	-	-	-

示例和结果：

脚本示例

示例	结果
`ConvertToLocalTime('2007-11-10 23:59:00','Paris')`	返回“2007-11-11 00:59:00”以及相应的内部时间戳表示。

示例	结果
ConvertToLocalTime(UTC(), 'GMT-05:00')	返回北美东海岸的时间，例如纽约。
ConvertToLocalTime(UTC(), 'GMT-05:00', True)	返回北美东海岸的时间，例如纽约，而不调整日间节省时间。

day

此函数用于根据标准数字解释当 **expression** 小数部分被解释为日期时返回一个表示某天的整数。

函数返回特定月份的日期。它通常用于将日期字段作为日历维度的一部分导出。

语法：

```
day(expression)
```

返回数据类型： 整数

函数示例

示例	结果
day('1971-10-12')	返回 12
day('35648')	返回 6，因为 35648 = 1997-08-06

dayend

此函数用于返回与 **time** 中包含的一天的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **TimestampFormat**。

语法：

```
DayEnd(time[, [period_no[, day_start]])
```

返回数据类型： 双

参数：

参数

参数	说明
time	要求值的时间戳。
period_no	**period_no** 为整数，或解算为整数的表达式，其中值 0 表示该天包含 **time**。**period_no** 为负数表示前几天，正数表示随后的几天。
day_start	要指定不想从某一日的午夜开始工作，可指定一个偏移作为某日内时间的小数 **day_start**。例如，0.125 表示上午 3 点。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
dayend('25/01/2013 16:45:00')	返回 25/01/2013 23:59:59。
dayend('25/01/2013 16:45:00', -1)	返回 24/01/2013 23:59:59。
dayend('25/01/2013 16:45:00', 0, 0.5)	返回 26/01/2013 11:59:59。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找标记每个发票日期之后当天结束的时间戳。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
DayEnd(InvDate, 1) AS DEnd
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 dayend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	DEnd
28/03/2012	29/03/2012 23:59:59
10/12/2012	11/12/2012 23:59:59

InvDate	DEnd
5/2/2013	07/02/2013 23:59:59
31/3/2013	01/04/2013 23:59:59
19/5/2013	20/05/2013 23:59:59
15/9/2013	16/09/2013 23:59:59
11/12/2013	12/12/2013 23:59:59
2/3/2014	03/03/2014 23:59:59
14/5/2014	15/05/2014 23:59:59
13/6/2014	14/06/2014 23:59:59
7/7/2014	08/07/2014 23:59:59
4/8/2014	05/08/2014 23:59:59

daylightsaving

用于返回如 Windows 所定义的当下为日间省时的调整。

语法：

```
DaylightSaving( )
```

返回数据类型：双

示例：

```
daylightsaving( )
```

dayname

此函数用于返回一个值，显示与包含 **time** 当天第一毫秒的时间戳对应的基本数值的日期。

语法：

```
DayName(time[, period_no [, day_start]])
```

返回数据类型：双

参数：

参数

参数	说明
time	要求值的时间戳。

参数	说明
period_no	**period_no** 为整数，或解算为整数的表达式，其中值 0 表示该天包含 **time**。**period_no** 为负数表示前几天，正数表示随后的几天。
day_start	要指定不想从某一日的午夜开始工作，可指定一个偏移作为某日内时间的小数 **day_start**。例如，0.125 表示上午 3 点。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`dayname('25/01/2013 16:45:00')`	返回 25/01/2013。
`dayname('25/01/2013 16:45:00', -1)`	返回 24/01/2013。
`dayname('25/01/2013 16:45:00', 0, 0.5 )`	返回 25/01/2013。 显示与 '25/01/2013 12:00:00.000. 对应的基础数值的完整时间戳

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

在此例中，已根据标记表格中每个发票日期之后当天开始的时间戳创建日期名称。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
DayName(InvDate, 1) AS DName
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 dayname() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	DName
28/03/2012	29/03/2012 00:00:00
10/12/2012	11/12/2012 00:00:00
5/2/2013	07/02/2013 00:00:00
31/3/2013	01/04/2013 00:00:00
19/5/2013	20/05/2013 00:00:00
15/9/2013	16/09/2013 00:00:00
11/12/2013	12/12/2013 00:00:00
2/3/2014	03/03/2014 00:00:00
14/5/2014	15/05/2014 00:00:00
13/6/2014	14/06/2014 00:00:00
7/7/2014	08/07/2014 00:00:00
4/8/2014	05/08/2014 00:00:00

daynumberofquarter

此函数用于计算时间戳所属的季度的天数。

语法：

```
DayNumberOfQuarter(timestamp[,start_month])
```

返回数据类型：整数

此函数使用的是基于 366 天的年份。

参数：

参数

参数	说明
timestamp	要求值的日期。
start_month	通过在 2 和 12 之间(如果省略，则为 1) 指定 **start_month**，年初可移动到任何一个月的第一天。例如，如果您想要从 3 月 1 日开始的财政年工作，请指定 **start_month** = 3。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
DayNumberOfQuarter('12/09/2014')	返回 74，当前季度的天数。
DayNumberOfQuarter ('12/09/2014',3)	返回 12，当前季度的天数。 在此例中，第一个季度从三月份开始（因为已将 start_month 指定为 3）。这意味着当前季度为第三个季度，从 9 月 1 日开始。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
ProjectTable:
LOAD recno() as InvID, * INLINE [
StartDate
28/03/2014
10/12/2014
5/2/2015
31/3/2015
19/5/2015
15/9/2015
] ;
NrDays:
Load *,
DayNumberOfQuarter(StartDate,4) As DayNrQtr
Resident ProjectTable;
Drop table ProjectTable;
```

结果列表显示了为表格中的每条记录返回的 DayNumberOfQuarter 值。

结果表

InvID	StartDate	DayNrQtr
1	28/03/2014	88
2	10/12/2014	71
3	5/2/2015	36
4	31/3/2015	91
5	19/5/2015	49
6	15/9/2015	77

daynumberofyear

此函数用于计算时间戳所属的年份的天数。从该年度的第一天的第一毫秒开始计算，但可以偏移第一个月。

语法：

```
DayNumberOfYear(timestamp[,start_month])
```

返回数据类型：整数

此函数使用的是基于 366 天的年份。

参数：

参数

参数	说明
timestamp	要求值的日期。
start_month	通过在 2 和 12 之间(如果省略，则为 1)指定 **start_month**，年初可移动到任何一个月的第一天。例如，如果您想要从 3 月 1 日开始的财政年工作，请指定 **start_month** = 3。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
DayNumberOfYear('12/09/2014')	返回 256，从第一年开始算起的天数。
DayNumberOfYear('12/09/2014',3)	返回 196，从第一个三月开始算起的天数。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
ProjectTable:
LOAD recno() as InvID, * INLINE [
StartDate
28/03/2014
10/12/2014
5/2/2015
31/3/2015
19/5/2015
15/9/2015
] ;
NrDays:
Load *,
```

```
DayNumberOfYear(StartDate,4) As DayNrYear
Resident ProjectTable;
Drop table ProjectTable;
```

结果列表显示了为表格中的每条记录返回的 DayNumberOfYear 值。

结果表

InvID	StartDate	DayNrYear
1	28/03/2014	363
2	10/12/2014	254
3	5/2/2015	311
4	31/3/2015	366
5	19/5/2015	49
6	15/9/2015	168

daystart

此函数用于返回与 **time** 参数中包含的一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **TimestampFormat**。

语法：

```
DayStart(time[, [period_no[, day_start]])
```

返回数据类型：双

参数：

参数

参数	说明
time	要求值的时间戳。
period_no	**period_no** 为整数，或解算为整数的表达式，其中值 0 表示该天包含 **time**。**period_no** 为负数表示前几天，正数表示随后的几天。
day_start	要指定不想从某一日的午夜开始工作，可指定一个偏移作为某日内时间的小数 **day_start**。例如，0.125 表示上午 3 点。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
daystart('25/01/2013 16:45:00')	返回 25/01/2013 00:00:00。
daystart('25/01/2013 16:45:00', -1)	返回 24/01/2013 00:00:00。
daystart('25/01/2013 16:45:00', 0, 0.5)	返回 25/01/2013 12:00:00。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找标记每个发票日期之后当天开始的时间戳。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
DayStart(InvDate, 1) AS DStart
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 daystart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	DStart
28/03/2012	29/03/2012 00:00:00
10/12/2012	11/12/2012 00:00:00
5/2/2013	07/02/2013 00:00:00
31/3/2013	01/04/2013 00:00:00
19/5/2013	20/05/2013 00:00:00

InvDate	DStart
15/9/2013	16/09/2013 00:00:00
11/12/2013	12/12/2013 00:00:00
2/3/2014	03/03/2014 00:00:00
14/5/2014	15/05/2014 00:00:00
13/6/2014	14/06/2014 00:00:00
7/7/2014	08/07/2014 00:00:00
4/8/2014	05/08/2014 00:00:00

firstworkdate

firstworkdate 函数用于返回最近的起始日以获得 **no_of_workdays**(周一至周五) , 将任何列出的可选节假日考虑在内 , 不迟于 **end_date**。**end_date** 和 **holiday** 应为有效的日期或时间戳。

语法:

```
firstworkdate(end_date, no_of_workdays {, holiday} )
```

返回数据类型: 整数

参数:

参数

参数	说明
end_date	要求值的结束日期的时间戳。
no_of_workdays	要实现的工作日天数。
holiday	从工作日排除假期。假期可表述为开始日期和结束日期 , 以逗号分隔。 **示例: '25/12/2013', '26/12/2013'** 您可以指定多个假期 , 以逗号分隔。 **示例: '25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014'**

示例和结果:
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
firstworkdate ('29/12/2014', 9)	返回 17/12/2014。
firstworkdate ('29/12/2014', 9, '25/12/2014', '26/12/2014')	返回 15/12/2014, 因为已将两天假期考虑在内。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
ProjectTable:
LOAD *, recno() as InvID, INLINE [
EndDate
28/03/2015
10/12/2015
5/2/2016
31/3/2016
19/5/2016
15/9/2016
] ;
NrDays:
Load *,
FirstWorkDate(EndDate,120) As StartDate
Resident ProjectTable;
Drop table ProjectTable;
```

结果列表显示了为表格中的每条记录返回的 FirstWorkDate 值。

结果表

InvID	EndDate	StartDate
1	28/03/2015	13/10/2014
2	10/12/2015	26/06/2015
3	5/2/2016	24/08/2015
4	31/3/2016	16/10/2015
5	19/5/2016	04/12/2015
6	15/9/2016	01/04/2016

GMT

此函数用于返回来自系统时钟的当前 Greenwich Mean Time 和 Windows 时间设置。

语法：

```
GMT( )
```

返回数据类型：双

示例：

```
gmt( )
```

hour

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示小时的整数。

语法：

```
hour(expression)
```

返回数据类型：整数

示例和结果：

脚本示例

示例	结果
`hour( '09:14:36' )`	返回 9
`hour( '0.5555' )`	返回 13(因为 0.5555 = 13:19:55)

inday

此函数用于返回 True，如果 **timestamp** 位于包含 **base_timestamp** 的一天以内。

语法：

```
InDay (timestamp, base_timestamp, period_no[, day_start])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_timestamp** 进行比较的日期和时间。
base_ timestamp	日期和时间用于计算时间戳的值。
period_no	该天可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该天包含 **base_ timestamp**。**period_no** 为负数表示前几天，正数表示随后的几天。
day_start	如果不想从每一日的午夜开始处理，可指定一个偏移作为某日内时间的小数 **day_start**。例如，0.125 表示上午 3 点。

示例和结果：

脚本示例

示例	结果
inday ('12/01/2006 12:23:00', '12/01/2006 00:00:00', 0)	返回 True
inday ('12/01/2006 12:23:00', '13/01/2006 00:00', 0)	返回 False
inday ('12/01/2006 12:23:00', '12/01/2006 00:00:00', -1)	返回 False
inday ('11/01/2006 12:23:00', '12/01/2006 00:00:00', -1)	返回 True
inday ('12/01/2006 12:23:00', '12/01/2006 00:00:00', 0, 0.5)	返回 False
inday ('12/01/2006 11:23:00', '12/01/2006 00:00:00', 0, 0.5)	返回 True

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例检查发票日期是否在以 base_timestamp 开始的当天的任何时间内。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvTime
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InDay(InvTime, '28/03/2012 00:00:00', 0) AS InDayEx
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inday() 函数的返回值的列。

结果表

InvTime	InDayEx
28/03/2012	-1 (True)
10/12/2012	0 (False)

InvTime	InDayEx
5/2/2013	0 (False)
31/3/2013	0 (False)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

indaytotime

此函数用于返回 True，如果 **timestamp** 位于包含 **base_timestamp** 为止以及包括 **base_timestamp** 精确毫秒的日子部分以内。

语法：

```
InDayToTime (timestamp, base_timestamp, period_no[, day_start])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_timestamp** 进行比较的日期和时间。
base_timestamp	日期和时间用于计算时间戳的值。
period_no	该天可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该天包含 **base_timestamp**。**period_no** 为负数表示前几天，正数表示随后的几天。
day_start	(可选) 如果不想从每一日的午夜开始处理，可指定一个偏移作为某日内时间的小数 **day_start**。例如，0.125 表示上午 3 点。

示例和结果：

脚本示例

示例	结果
indaytotime ('12/01/2006 12:23:00', '12/01/2006 23:59:00', 0)	返回 True
indaytotime ('12/01/2006 12:23:00', '12/01/2006 00:00:00', 0)	返回 False
indaytotime ('11/01/2006 12:23:00', '12/01/2006 23:59:00', -1)	返回 True

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例检查发票时间戳是否在以 base_timestamp 开始的当天的 17:00:00 之前。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvTime
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InDayToTime(InvTime, '28/03/2012 17:00:00', 0) AS InDayExTT
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 indaytotime() 函数的返回值的列。

结果表

InvTime	InDayExTT
28/03/2012	-1 (True)
10/12/2012	0 (False)
5/2/2013	0 (False)
31/3/2013	0 (False)
19/5/2013	0 (False)

InvTime	InDayExTT
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inlunarweek

此函数用于判断 **timestamp** 是否位于包含 **base_date** 的阴历周以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

语法：

```
InLunarWeek (timestamp, base_date, period_no[, first_week_day])
```

返回数据类型： 布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算阴历周的值。
period_no	阴历周可通过 **period_no** 偏移。period_no 为整数，其中值 0 表示该阴历周包含 **base_date**。**period_no** 为负数表示前几个阴历星期，为正数表示随后的几个阴历星期。
first_week_day	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

示例和结果：

脚本示例

示例	结果
`inlunarweek ('12/01/2013', '14/01/2013', 0)`	返回 True。由于 timestamp 的值，12/01/2013 属于 08/01/2013 至 14/01/2013 的周之内。

示例	结果
`inlunarweek ('12/01/2013', '07/01/2013', 0)`	返回 False。由于 base_date, 将属于阴历周的 07/01/2013 定义为 01/01/2013 至 07/01/2013。
`inlunarweek ('12/01/2013', '14/01/2013', -1)`	返回 False。由于将 period_no 的值指定为 -1, 表示将周移动到前一周, 即 01/01/2013 到 07/01/2013。
`inlunarweek ('07/01/2013', '14/01/2013', -1)`	返回 True。与前面的示例相比, 时间戳是在考虑到向后移动后的周内。
`inlunarweek ('11/01/2006', '08/01/2006', 0, 3)`	返回 False。由于将 first_week_day 的值指定为 3, 表示从 04/01/2013 开始计算的一年, 因此 base_date 的值在第一周内, 并且 timestamp 的值属于 11/01/2013 至 17/01/2013 周内。

示例:

将示例脚本添加到应用程序并运行。要查看结果, 将结果列中列出的字段添加到应用程序中的工作表。

以下示例检查发票日期是否在按四周从 base_date 值开始移动的周内。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InLunarWeek(InvDate, '11/01/2013', 4) AS InLWeekPlus4
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inlunarweek() 函数的返回值的列。

对于 InvDate5/2/2013 的值, 此函数返回 True, 因为 base_date 的值, 11/01/2013 按四周移动, 因此它应属于 5/02/2013 至 11/02/2013 的周内。

结果表

InvDate	InLWeekPlus4
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	-1 (True)
31/3/2013	0 (False)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inlunarweektodate

此函数用于判断 **timestamp** 是否位于截止以及包括 **base_date** 最后毫秒的阴历周的某部分以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

语法：

```
InLunarWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

返回数据类型： 布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算阴历周的值。
period_no	阴历周可通过 **period_no** 偏移。period_no 为整数，其中值 0 表示该阴历周包含 **base_date**。**period_no** 为负数表示前几个阴历星期，为正数表示随后的几个阴历星期。
first_week_day	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

示例和结果：

脚本示例

示例	结果
inlunarweektodate ('12/01/2013', '13/01/2013', 0)	返回 True。由于 timestamp 的值，12/01/2013 属于 08/01/2013 至 13/01/2013 的周的一部分。
inlunarweektodate ('12/01/2013', '11/01/2013', 0)	返回 False。由于 timestamp 的值晚于 base_date 的值，尽管这两个日期都属于 12/01/2012 之前的同一阴历周。
inlunarweektodate ('12/01/2006', '05/01/2006', 1)	返回 True。将 period_no 的值指定为 1，表示 base_date 向前移动一周，因此 timestamp 的值属于阴历周的一部分。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例检查发票日期是否属于按四周从 base_date 值开始移动的周的一部分。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InLunarWeekToDate(InvDate, '07/01/2013', 4) AS InLWeek2DPlus4
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inlunarweek() 函数的返回值的列。

对于 InvDate5/2/2013 的值，此函数返回 True，因为 base_date 的值，11/01/2013 按四周移动，因此它属于 5/02/2013 至 07/02/2013 的周的一部分。

结果表

InvDate	InLWeek2DPlus4
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	-1 (True)
31/3/2013	0 (False)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inmonth

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的月份以内。

语法：

```
InMonth (timestamp, base_date, period_no[, first_month_of_year])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算月份的值。
period_no	该月份可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该月份包含 **base_date**。**period_no** 为负数表示前几月，为正数则表示随后的几月。
first_month_of_year	**first_month_of_year** 参数被禁用并且被保留以供未来使用。

示例和结果：

脚本示例

示例	结果
inmonth ('25/01/2013', '01/01/2013', 0)	返回 True
inmonth('25/01/2013', '01/04/2013', 0)	返回 False
inmonth ('25/01/2013', '01/01/2013', -1)	返回 False
inmonth ('25/12/2012', '01/01/2013', -1)	返回 True

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例检查发票日期是否属于 base_date 中的月份之后第四个月的任何时间内（通过将 period_no 指定为 4）。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InMonth(InvDate, '31/01/2013', 4) AS InMthPlus4
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inmonth() 函数的返回值的列。

结果表

InvDate	InMthPlus4
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	0 (False)

InvDate	InMthPlus4
31/3/2013	0 (False)
19/5/2013	-1 (True)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inmonths

此函数用于判断时间戳是否位于作为基准日期的同一个月、两个月、季度、四个月或半年以内。另外，也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

语法：

```
InMonths(n_months, timestamp, base_date, period_no [, first_month_of_year])
```

返回数据类型： 布尔值

参数：

参数

参数	说明
n_months	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1（相当于 inmonth() 函数）、2（两个月）、3（相当于 inquarter() 函数）、4（四个月）或 6（半年）。
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算周期的值。
period_no	该周期可通过 **period_no** 偏移，其为整数，或解算为整数的表达式，其中值 0 表示该周期包含 **base_date**。**period_no** 为负数表示前几个时段，为正数则表示随后的几个时段。
first_month_of_year	如果您不想从一月开始处理（财政）年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`inmonths(4, '25/01/2013', '25/04/2013', 0)`	返回 True。由于 timestamp 的值，25/01/2013 属于 01/01/2013 至 30/04/2013 的四个月周期内，其中 base_date 的值在 25/04/2013 内。
`inmonths(4, '25/05/2013', '25/04/2013', 0)`	返回 False。由于 25/05/2013 在与前面示例相同的周期之外。
`inmonths(4, '25/11/2012', '01/02/2013', -1 )`	返回 True。由于 period_no 的值，-1 表示将搜索周期向后移动四个月中的其中一个周期(n-months 的值)，这可以使搜索周期介于 01/09/2012 至 31/12/2012. 之间
`inmonths( 4, '25/05/2006', '01/03/2006', 0, 3)`	返回 True。由于将 first_month_of_year 的值设置为 3，这使得搜索周期介于 01/03/2006 至 30/07/2006 之内，而不是 01/01/2006 至 30/04/2006。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中检查发票日期是否在 base_date 按一个两个月周期向前移动的两个月周期内(通过将 period_no 指定为 1)。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InMonths(2, InvDate, '11/02/2013', 1) AS InMthsPlus1
Resident TempTable;
```

```
Drop table TempTable;
```

结果列表包含原始日期和包括 InMonths() 函数的返回值的列。

搜索周期介于 01/03/2013 至 30/04/2013 之间，因为 base_date 的值从函数中的值 (11/02/2013) 开始向前移动两个月。

结果表

InvDate	InMthsPlus1
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	0 (False)
31/3/2013	-1 (True)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inmonthstodate

此函数用于判断时间戳是否位于截止以及包括 **base_date** 的最后毫秒的某个月、两个月、季度、四个月或半年周期的一部分以内。另外，它也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

语法：

```
InMonths (n_months, timestamp, base_date, period_no[, first_month_of_year ])
```

返回数据类型： 布尔值

参数：

参数

参数	说明
n_months	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月) 或 6(半年)。
timestamp	想要用来与 **base_date** 进行比较的日期。

参数	说明
base_date	日期用于计算周期的值。
period_no	该周期可通过 **period_no** 偏移，其为整数，或解算为整数的表达式，其中值 0 表示该周期包含 **base_date**。**period_no** 为负数表示前几个时段，为正数则表示随后的几个时段。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`inmonthstodate(4, '25/01/2013', '25/04/2013', 0)`	返回 True。由于 timestamp 的值，25/01/2013 属于 01/01/2013 至 25/04/2013 结束的四个月周期内，其中 base_date 的值在 25/04/2013 内。
`inmonthstodate(4, '26/04/2013', '25/04/2006', 0)`	返回 False。由于 26/04/2013 在与前面示例相同的周期之外。
`inmonthstodate(4, '25/09/2005', '01/02/2006', -1)`	返回 True。由于 period_no 的值，-1 表示将搜索周期向后移动四个月中的其中一个周期(n-months 的值)，这可以使搜索周期介于 01/09/2005 至 01/02/2006. 之间
`inmonthstodate(4, '25/04/2006', '01/06/2006', 0, 3)`	返回 True。由于将 first_month_of_year 的值设置为 3，这使得搜索周期介于 01/03/2006 至 01/06/2006 之内，而不是 01/05/2006 至 01/06/2006。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中检查发票日期是否在两个月周期截止并包括 base_date 按两个月周期向前移动的一部分内(通过将 period_no 指定为 4)。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
```

```
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InMonthsToDate(2, InvDate, '15/02/2013', 4) AS InMths2DPlus4
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 InMonths() 函数的返回值的列。

搜索周期介于 01/09/2013 至 15/10/2013 之间，因为 base_date 的值从函数中的值 (15/02/2013) 开始向前移动八个月。

结果表

InvDate	InMths2DPlus4
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	0 (False)
31/3/2013	0 (False)
19/5/2013	0 (False)
15/9/2013	-1 (True)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inmonthtodate

用于返回 True，如果 **date** 位于包含 **basedate** 为止以及包括 **basedate** 最后毫秒的月份部分以内。

语法：

```
InMonthToDate (timestamp, base_date, period_no)
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算月份的值。
period_no	该月份可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该月份包含 **base_date**。**period_no** 为负数表示前几月，为正数则表示随后的几月。

示例和结果：

脚本示例

示例	结果
`inmonthtodate ('25/01/2013', '25/01/2013', 0)`	返回 True
`inmonthtodate ('25/01/2013', '24/01/2013', 0)`	返回 False
`inmonthtodate ('25/01/2013', '28/02/2013', -1)`	返回 True

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

通过将 period_no 指定为 4，以下示例检查发票日期是否在 base_date 中的月份之后，但在 base_date 中指定的当天结束之前的第四个月中。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InMonthToDate(InvDate, '31/01/2013', 4) AS InMthPlus42D
Resident TempTable;
```

```
Drop table TempTable;
```

结果列表包含原始日期和包括 inmonthtodate() 函数的返回值的列。

结果表

InvDate	InMthPlus42D
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	0 (False)
31/3/2013	0 (False)
19/5/2013	-1 (True)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inquarter

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的季度以内。

语法：

```
InQuarter (timestamp, base_date, period_no[, first_month_of_year])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算季度的值。
period_no	该季度可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该季度包含 **base_date**。**period_no** 为负数表示前几季，为正数则表示随后的几季。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：

脚本示例

示例	结果
inquarter ('25/01/2013', '01/01/2013', 0)	返回 True
inquarter ('25/01/2013', '01/04/2013', 0)	返回 False
inquarter ('25/01/2013', '01/01/2013', -1)	返回 False
inquarter ('25/12/2012', '01/01/2013', -1)	返回 True
inquarter ('25/01/2013', '01/03/2013', 0, 3)	返回 False
inquarter ('25/03/2013', '01/03/2013', 0, 3)	返回 True

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

通过将 first_month_of_year 的值设置为 4，并包括 base_date 31/01/2013，以下示例检查发票日期是否在指定财政年的第四个季度内。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InQuarter(InvDate, '31/01/2013', 0, 4) AS Qtr4FinYr1213
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inquarter() 函数的返回值的列。

结果表

InvDate	Qtr4Fin1213
28/03/2012	0 (False)

InvDate	Qtr4Fin1213
10/12/2012	0 (False)
5/2/2013	-1 (True)
31/3/2013	-1 (True)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inquartertodate

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 为止以及包括 **base_date** 最后毫秒的季度部分以内。

语法：

```
InQuarterToDate (timestamp, base_date, period_no [, first_month_of_year])
```

返回数据类型： 布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算季度的值。
period_no	该季度可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该季度包含 **base_date**。**period_no** 为负数表示前几季，为正数则表示随后的几季。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：

脚本示例

示例	结果
inquartertodate ('25/01/2013', '25/01/2013', 0)	返回 True
inquartertodate (25/01/2013', '24/01/2013', 0)	返回 False
inquartertodate ('25/01/2012', '01/02/2013', -1)	返回 True

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

通过将 first_month_of_year 的值设置为 4，以下示例检查发票日期是否在指定财政年的第四个季度，并在 28/02/2013 结束之前。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InQuarterToDate(InvDate, '28/02/2013', 0, 4) AS Qtr42Date
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inquartertodate() 函数的返回值的列。

结果表

InvDate	Qtr42Date
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	-1 (True)
31/3/2013	0 (False)

InvDate	Qtr42Date
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inweek

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的星期以内。

语法：

```
InWeek (timestamp, base_date, period_no[, first_week_day])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算星期的值。
period_no	该星期可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该星期包含 **base_date**。**period_no** 为负数表示前几个星期，正数表示随后的几个星期。
first_week_day	默认情况下，一周的第一天为星期一，从星期天与星期一之间的午夜开始。要指示一周从另外一天开始，应在 **first_week_day** 中指定偏移。这需要指定一个表示天数和/或天的分位数的整数。

示例和结果：

脚本示例

示例	结果
`inweek ('12/01/2006', '14/01/2006', 0)`	返回 True
`inweek ('12/01/2006', '20/01/2006', 0 )`	返回 False

示例	结果
inweek ('12/01/2006', '14/01/2006', -1)	返回 False
inweek ('07/01/2006', '14/01/2006', -1)	返回 True
inweek ('12/01/2006', '09/01/2006', 0, 3)	返回 False 由于将 first_week_day 指定为 3(星期四) , 这使得本周的第一天 12/01/2006 之后一周包含 09/01/2006。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例检查发票日期是否属于 base_date 中的周之后第四周的任何时间内(通过将 period_no 指定为 4) 。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InWeek(InvDate, '11/01/2013', 4) AS InWeekPlus4
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inweek() 函数的返回值的列。

InvDate5/2/2013 属于 base_date 之后四周的某一周中：11/1/2013.

结果表

InvDate	InWeekPlus4
28/03/2012	0 (False)
10/12/2012	0 (False)

InvDate	InWeekPlus4
5/2/2013	-1 (True)
31/3/2013	0 (False)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inweektodate

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 为止以及包括 **base_date** 最后毫秒的星期部分以内。

语法：

```
InWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算星期的值。
period_no	该星期可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该星期包含 **base_date**。**period_no** 为负数表示前几个星期，正数表示随后的几个星期。
first_week_day	默认情况下，一周的第一天为星期一，从星期天与星期一之间的午夜开始。要指示一周从另外一天开始，应在 **first_week_day** 中指定偏移。这需要指定一个表示天数和/或天的分位数的整数。

示例和结果：

脚本示例

示例	结果
inweektodate ('12/01/2006', '12/01/2006', 0)	返回 True
inweektodate ('12/01/2006', '11/01/2006', 0)	返回 False
inweektodate ('12/01/2006', '18/01/2006', -1)	返回 False 由于将 period_no 指定为 -1，作为衡量 timestamp 的依据的有效日期为 11/01/2006。
inweektodate ('11/01/2006', '12/01/2006', 0, 3)	返回 False 由于将 first_week_day 指定为 3(星期四)，这使得本周的第一天 12/01/2006 之后一周包含 12/01/2006。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例检查发票日期是否在 base_date 中的周之后第四周期间，但在 base_date 的值之前(通过将 period_no 指定为 4)。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InWeekToDate(InvDate, '11/01/2013', 4) AS InWeek2DPlus4
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inweek() 函数的返回值的列。

结果表

InvDate	InWeek2DPlus4
28/03/2012	0 (False)
10/12/2012	0 (False)
5/2/2013	-1 (True)
31/3/2013	0 (False)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inyear

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的年份以内。

语法：

```
InYear (timestamp, base_date, period_no [, first_month_of_year])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算年份的值。
period_no	该年份可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该年份包含 **base_date**。**period_no** 为负数表示前几年，为正数表示随后几年。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
inyear ('25/01/2013', '01/01/2013', 0)	返回 True
inyear ('25/01/2012', '01/01/2013', 0)	返回 False
inyear ('25/01/2013', '01/01/2013', -1)	返回 False
inyear ('25/01/2012', '01/01/2013', -1)	返回 True
inyear ('25/01/2013', '01/01/2013', 0, 3)	返回 True base_date 和 first_month_of_year 的值指定 timestamp 必须位于 01/03/2012 和 28/02/2013 之间
inyear ('25/03/2013', '01/07/2013', 0, 3)	返回 True

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

通过将 first_month_of_year 的值设置为 4，并包括介于 1/4/2012 和 31/03/2013 之间 base_date，以下示例检查发票日期是否在指定财政年内。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];
```

测试 InvDate 是否在 1/04/2012 至 31/03/2013 财政年内：

```
InvoiceData:
LOAD *,
InYear(InvDate, '31/01/2013', 0, 4) AS FinYr1213
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inyear() 函数的返回值的列。

结果表

InvDate	FinYr1213
28/03/2012	0 (False)
10/12/2012	-1 (True)
5/2/2013	-1 (True)
31/3/2013	-1 (True)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

inyeartodate

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 为止以及包括 **base_date** 最后毫秒的年份部分以内。

语法：

```
InYearToDate (timestamp, base_date, period_no[, first_month_of_year])
```

返回数据类型：布尔值

参数：

参数

参数	说明
timestamp	想要用来与 **base_date** 进行比较的日期。
base_date	日期用于计算年份的值。
period_no	该年份可通过 **period_no** 偏移。**period_no** 为整数，其中值 0 表示该年份包含 **base_date**。**period_no** 为负数表示前几年，为正数表示随后几年。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：

脚本示例

示例	结果
inyeartodate ('2013/01/25', '2013/02/01', 0)	返回 True
inyeartodate ('2012/01/25', '2013/01/01', 0)	返回 False
inyeartodate ('2012/01/25', '2013/02/01', -1)	返回 True
inyeartodate ('2012/11/25', '2013/01/31', 0, 4)	返回 True timestamp 的值在第四个月中开始的财政年内，并在 base_date 的值之前。
inyeartodate ('2013/3/31', '2013/01/31', 0, 4)	返回 False 与前面的示例相比，timestamp 的值仍在财政年内，但它在 base_date 的值之后，因此它属于一年的某一部分以外。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

通过将 first_month_of_year 的值设置为 4，以下示例检查发票日期是否在指定财政年，并在 31/01/2013 结束之前一年的某一部分内。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
InYearToDate(InvDate, '31/01/2013', 0, 4) AS FinYr2Date
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 inyeartodate() 函数的返回值的列。

结果表

InvDate	FinYr2Date
28/03/2012	0 (False)
10/12/2012	-1 (True)
5/2/2013	0 (False)
31/3/2013	0 (False)
19/5/2013	0 (False)
15/9/2013	0 (False)
11/12/2013	0 (False)
2/3/2014	0 (False)
14/5/2014	0 (False)
13/6/2014	0 (False)
7/7/2014	0 (False)
4/8/2014	0 (False)

lastworkdate

lastworkdate 函数用于返回最早的结束日以获得 **no_of_workdays**(周一至周五) , 如果在 **start_date** 开始考虑任何列出的可选 **holiday**。**start_date** 和 **holiday** 应是有效的日期或时间戳。

语法：

```
lastworkdate(start_date, no_of_workdays {, holiday})
```

返回数据类型： 双

参数：

参数

参数	说明
start_date	评估的开始日期。
no_of_workdays	要实现的工作日天数。
holiday	从工作日排除假期。假期可表述为开始日期和结束日期, 以逗号分隔。 **示例： '25/12/2013', '26/12/2013'** 您可以指定多个假期, 以逗号分隔。 **示例： '25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014'**

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`lastworkdate ('19/12/2014', 9)`	返回 '31/12/2014'
`lastworkdate ('19/12/2014', 9, '2014-12-25', '2014-12-26')`	返回 02/01/2015，因为已将两天假期考虑在内。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
ProjectTable:
LOAD *, recno() as InvID, INLINE [
StartDate
28/03/2014
10/12/2014
5/2/2015
31/3/2015
19/5/2015
15/9/2015
] ;
NrDays:
Load *,
LastWorkDate(StartDate,120) As EndDate
Resident ProjectTable;
Drop table ProjectTable;
```

结果列表显示了为表格中的每条记录返回的 LastWorkDate 值。

结果表

InvID	StartDate	EndDate
1	28/03/2014	11/09/2014
2	10/12/2014	26/05/2015
3	5/2/2015	27/07/2015
4	31/3/2015	14/09/2015
5	19/5/2015	02/11/2015
6	15/9/2015	29/02/2016

localtime

此函数用于返回指定时区的系统时钟的当前时间戳。

语法：

```
LocalTime([timezone [, ignoreDST ]])
```

返回数据类型：双

参数：

参数

参数	说明
timezone	将 **timezone** 指定为一个字符串，其中包含在 **Windows 控制面板**的**时区**下专为**日期和时间**列出的任何一个地理位置，或指定为“GMT+hh:mm”格式的字符串。 如果未指定时区，则将返回本地时间。
ignoreDST	如果 **ignoreDST** 为 -1 (True)，则将忽略夏令时。

示例和结果：

以下示例基于在 2014-10-22 12:54:47 本地时间调用的函数，本地时区为 GMT+01:00。

脚本示例

示例	结果
`localtime ()`	返回本地时间 2014-10-22 12:54:47。
`localtime ('London')`	返回伦敦本地时间 2014-10-22 11:54:47。
`localtime ('GMT+02:00')`	返回 GMT+02:00 时区的本地时间 2014-10-22 13:54:47。
`localtime ('Paris','-1')`	返回巴黎本地时间 2014-10-22 11:54:47，忽略夏令时。

lunarweekend

此函数用于返回与包含 **date** 的阴历周的最后毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

语法：

```
LunarweekEnd(date[, period_no[, first_week_day]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。

参数	说明
period_no	**period_no** 为整数，或解算为整数的表达式，其中值 0 表示该阴历周包含 **date**。**period_no** 为负数表示前几个阴历星期，为正数表示随后的几个阴历星期。
first_week_day	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`lunarweekend('12/01/2013')`	返回 14/01/2013 23:59:59。
`lunarweekend('12/01/2013', -1)`	返回 7/01/2013 23:59:59。
`lunarweekend('12/01/2013', 0, 1)`	返回 15/01/2013 23:59:59。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找每个发票日期的阴历周的最后一天，其中 date 按一周移动（通过将 period_no 指定为 1）。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
LunarWeekEnd(InvDate, 1) AS LWkEnd
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 lunarweekend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	LWkEnd
28/03/2012	07/04/2012
10/12/2012	22/12/2012
5/2/2013	18/02/2013
31/3/2013	08/04/2013
19/5/2013	27/05/2013
15/9/2013	23/09/2013
11/12/2013	23/12/2013
2/3/2014	11/03/2014
14/5/2014	27/05/2014
13/6/2014	24/06/2014
7/7/2014	15/07/2014
4/8/2014	12/08/2014

lunarweekname

此函数用于返回一个显示值，显示与包含 **date** 的阴历周的第一天的第一毫秒时间戳对应的年份和阴历周数。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

语法：

```
LunarWeekName(date [, period_no[, first_week_day]])
```

返回数据类型： 双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，或解算为整数的表达式，其中值 0 表示该阴历周包含 **date**。**period_no** 为负数表示前几个阴历星期，为正数表示随后的几个阴历星期。
first_week_day	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

示例和结果：

脚本示例

示例	结果
lunarweekname('12/01/2013')	返回 2006/02。
lunarweekname('12/01/2013', -1)	返回 2006/01。
lunarweekname('12/01/2013', 0, 1)	返回 2006/02。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

在此例中，对于表格中的每个发票日期，根据周所在的年度创建阴历周名称，并将其与阴历周编号进行相关联(通过将 period_no 指定为 1 按一周移动)。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
LunarWeekName(InvDate, 1) AS LWkName
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 lunarweekname() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	LWkName
28/03/2012	2012/14
10/12/2012	2012/51
5/2/2013	2013/07
31/3/2013	2013/14

InvDate	LWkName
19/5/2013	2013/21
15/9/2013	2013/38
11/12/2013	2013/51
2/3/2014	2014/10
14/5/2014	2014/21
13/6/2014	2014/25
7/7/2014	2014/28
4/8/2014	2014/32

lunarweekstart

此函数用于返回与包含 **date** 的阴历周的第一毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

语法：

```
LunarweekStart(date[, period_no[, first_week_day]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，或解算为整数的表达式，其中值 0 表示该阴历周包含 **date**。**period_no** 为负数表示前几个阴历星期，为正数表示随后的几个阴历星期。
first_week_day	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`lunarweekstart ('12/01/2013')`	返回 08/01/2013。

示例	结果
lunarweekstart ('12/01/2013', -1)	返回 01/01/2013。
lunarweekstart ('12/01/2013', 0, 1)	返回 09/01/2013。 由于是通过将 first_week_day 设置为 1 来指定偏移量，这意味着已将一年的开始更改为 02/01/2013。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找每个发票日期的阴历周的第一天，其中 date 按一周移动（通过将 period_no 指定为 1）。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
LunarWeekStart(InvDate, 1) AS LWkStart
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 lunarweekstart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	LWkStart
28/03/2012	01/04/2012
10/12/2012	16/12/2012
5/2/2013	12/02/2013
31/3/2013	02/04/2013
19/5/2013	21/05/2013

InvDate	LWkStart
15/9/2013	17/09/2013
11/12/2013	17/12/2013
2/3/2014	05/03/2014
14/5/2014	21/05/2014
13/6/2014	18/06/2014
7/7/2014	09/07/2014
4/8/2014	06/08/2014

makedate

此函数用于返回根据年份 **YYYY**、月份 **MM** 和日期 **DD** 计算的日期。

语法：

```
MakeDate(YYYY [ , MM [ , DD ] ])
```

返回数据类型：双

参数：

参数

参数	说明
YYYY	作为整数的年份。
MM	作为整数的月份。如果未指定月份，则假定为 1(一月)。
DD	作为整数的天。如果未指定日期，则假定为 1(第 1 天)。

示例：图表表达式

图表表达式示例

示例	结果
`makedate(2012)`	返回 2012-01-01
`makedate(12)`	返回 0012-01-01
`makedate(2012,12)`	返回 2012-12-01
`makedate(2012,2,14)`	返回 2012-02-14

示例：加载脚本

makedate 可用在数据脚本中以将来自不同字段的日期数据组合成一个新的日期字段。在下面的示例，来自字段 transaction_year、transaction_month 和 transaction_day 的日数据被组合成名为 Transaction Date 的一个新字段。

在**数据加载编辑器**中，创建新的部分，然后添加示例脚本并运行它。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

加载脚本

```
SET DateFormat='DD/MM/YYYY';
SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT';
SET FirstWeekDay=0;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';

Transactions:
Load
*,
MakeDate(transaction_year, transaction_month, transaction_day) as "Transaction Date",
;

Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_day, transaction_amount,
transaction_quantity, discount, customer_id, size, color_code
3750, 2018, 08, 30, 12423.56, 23, 0,2038593, L, Red
3751, 2018, 09, 07, 5356.31, 6, 0.1, 203521, m, orange
3752, 2018, 09, 16, 15.75, 1, 0.22, 5646471, S, blue
3753, 2018, 09, 22, 1251, 7, 0, 3036491, l, Black
3754, 2018, 09, 22, 21484.21, 1356, 75, 049681, xs, Red
3756, 2018, 09, 22, -59.18, 2, 0.333333333333333, 2038593, M, Blue
3757, 2018, 09, 23, 3177.4, 21, .14, 203521, XL, Black
];
```

结果

Qlik Sense table showing results of the makedate function being used in the load script.

transaction_id	Transaction Date
3750	30/08/2018
3751	07/09/2018
3752	16/09/2018
3753	22/09/2018
3754	22/09/2018
3756	22/09/2018
3757	23/09/2018

maketime

此函数用于返回根据小时 **hh**、分钟 **mm** 和秒 **ss** 计算的时间。

语法：

```
MakeTime(hh [ , mm [ , ss ] ])
```

返回数据类型：双

参数：

参数

参数	说明
hh	作为整数的小时。
mm	作为整数的分钟。 如果未指定分钟，则假定为 00。
ss	作为整数的秒。 如果未指定秒，则假定为 00。

示例和结果：

脚本示例

示例	结果
maketime(22)	返回 22:00:00
maketime(22, 17)	返回 22:17:00
maketime(22, 17, 52)	返回 22:17:52

makeweekdate

此函数用于返回根据年份 **YYYY**、星期 **WW** 和星期几 **D** 计算的日期。

语法：

```
MakeWeekDate(YYYY [ , WW [ , D ] ])
```

返回数据类型：双

参数：

参数

参数	说明
YYYY	作为整数的年份。
WW	作为整数的周。
D	作为整数的星期几。 如果未指定星期几，则假定为 0(星期一)。

示例和结果：

脚本示例

示例	结果
makeweekdate(2014,6,6)	返回 2014-02-09
makeweekdate(2014,6,1)	返回 2014-02-04
makeweekdate(2014,6)	返回 2014-02-03(假定普通日为 0)

minute

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示分钟的整数。

语法：

```
minute(expression)
```

返回数据类型： 整数

示例和结果：

脚本示例

示例	结果
minute ('09:14:36')	返回 14
minute ('0.5555')	返回 19(因为 0.5555 = 13:19:55)

month

此函数用于返回包含在环境变量 **MonthNames** 中定义的月份名称的对偶值和一个介于 1 至 12 的整数。月份根据标准数字解释通过表达式的日期解释进行计算。

函数以 MonthName 系统变量的格式返回特定日期的月份名称。它通常用于在主日历中创建日期字段作为维度。

语法：

```
month(expression)
```

返回数据类型： 整数

函数示例

示例	结果
month('2012-10-12')	返回 Oct
month('35648')	返回 Aug, 因为 35648 = 1997-08-06

monthend

此函数用于返回与包含 **date** 的月份的最后一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
MonthEnd(date[, period_no])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，如果为 0 或忽略，表示该月包含 **date**。**period_no** 为负数表示前几月，为正数则表示随后的几月。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`monthend('19/02/2012')`	返回 29/02/2012 23:59:59。
`monthend('19/02/2001', -1)`	返回 31/01/2001 23:59:59。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例用于查找表格中每个发票日期的当月的最后一天，其中基准日期按四个月移动(通过将 *period_no* 指定为 4)。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
```

```
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
MonthEnd(InvDate, 4) AS MthEnd
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 monthend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	MthEnd
28/03/2012	31/07/2012
10/12/2012	30/04/2013
5/2/2013	30/06/2013
31/3/2013	31/07/2013
19/5/2013	30/09/2013
15/9/2013	31/01//2014
11/12/2013	30/04//2014
2/3/2014	31/07//2014
14/5/2014	30/09/2014
13/6/2014	31/10/2014
7/7/2014	30/11/2014
4/8/2014	31/12/2014

monthname

此函数用于返回一个显示值，该值显示该月（根据 **MonthNames** 脚本变量的格式）以及年，伴随一个与该月第一天第一毫秒的时间戳对应的基础数值。

语法：

```
MonthName(date[, period_no])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，如果为 0 或忽略，表示该月包含 **date**。**period_no** 为负数表示前几月，为正数则表示随后的几月。

示例：图表表达式
该示例使用日期格式 **DD/MM/YYYY**，在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。**SET Monthnames** 语句设置为 Jan;Feb;Mar，诸如此类。

图表表达式示例

示例	结果
`monthname('19/10/2013')`	返回 Oct 2013
`monthname('19/10/2013', -1)`	返回 Sep 2013

示例：加载脚本
在此例中，对于表格中的每个发票日期，根据从该年度的 base_date 移动四个月的月份名称创建月份名称。

在**数据加载编辑器**中，创建新的部分，然后添加示例脚本并运行它。然后，至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

加载脚本

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
MonthName(InvDate, 4) AS MthName
```

```
Resident TempTable;
Drop table TempTable;
```

结果

结果列表包含原始日期和包括 monthname() 函数的返回值的列。

InvDate	MthName
28/03/2012	Jul 2012
10/12/2012	Apr 2013
5/2/2013	Jun 2013
31/3/2013	Jul 2013
19/5/2013	Sep 2013
15/9/2013	Jan 2014
11/12/2013	Apr 2014
2/3/2014	Jul 2014
14/5/2014	Sep 2014
13/6/2014	Oct 2014
7/7/2014	Nov 2014
4/8/2014	Dec 2014

示例:加载脚本
在该示例中,对于表格中的每个 transaction_date,创建了值 Returnable_Until。Returnable_Until 值的计算是通过将 transaction_date 的月份移动至一月之后来进行。

在**数据加载编辑器**中,创建新的部分,然后添加示例脚本并运行它。然后,至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

加载脚本

```
SET DateFormat='YYYYMMDD';
SET TimestampFormat='YYYYMMDD h:mm:ss[.fff] TT';
SET FirstMonthOfYear=1;
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
SET
LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;D
ecember';

Transactions:
Load
*,
MonthName(Date#(transaction_date,'YYYYMMDD'), 1) as Returnable_Until,
;
```

```
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 20180830, 12423.56, 23, 0,2038593, L, Red
3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180916, 15.75, 1, 0.22, 5646471, S, blue
3753, 20180922, 1251, 7, 0, 3036491, l, Black
3754, 20180922, 21484.21, 1356, 75, 049681, xs, Red
3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, Blue
3757, 20180923, 3177.4, 21, .14, 203521, XL, Black
];
```

结果

Qlik Sense table showing results of the monthname function being used in the load script.

transaction_id	transaction_date	Returnable_Until
3750	20180830	Sep 2018
3751	20180907	Oct 2018
3752	20180916	Oct 2018
3753	20180922	Oct 2018
3754	20180922	Oct 2018
3756	20180922	Oct 2018
3757	20180923	Oct 2018

monthsend

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的最后毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

语法：

```
MonthsEnd(n_months, date[, period_no [, first_month_of_year]])
```

返回数据类型：双

参数：

参数

参数	说明
n_months	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月)或 6(半年)。
date	要求值的日期。

参数	说明
period_no	该周期可通过 **period_no** 偏移，其为整数，或解算为整数的表达式，其中值 0 表示该周期包含 **base_date**。**period_no** 为负数表示前几个时段，为正数则表示随后的几个时段。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`monthsend(4, '19/07/2013')`	返回 31/08/2013。
`monthsend(4, '19/10/2013', -1)`	返回 31/08/2013。
`monthsend(4, '19/10/2013', 0, 2)`	返回 31/01/2014。 因为该年度的开始变成 2 月。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例用于查找每个发票日期的两个月周期最后一天的结束，按一个两个月周期向前移动。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
MonthsEnd(2, InvDate, 1) AS BiMthsEnd
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 MonthsEnd() 函数的返回值的列。

结果表

InvDate	BiMthsEnd
28/03/2012	30/06/2012
10/12/2012	28/02/2013
5/2/2013	30/04/2013
31/3/2013	30/04/2013
19/5/2013	31/08/2013
15/9/2013	31/12/2013
11/12/2013	28/02/2014
2/3/2014	30/06/2014
14/5/2014	31/08/2014
13/6/2014	31/08/2014
7/7/2014	31/10/2014
4/8/2014	31/10/2014

monthsname

此函数用于返回一个显示值，表示时段各月份（根据 **MonthNames** 脚本变量的格式）和年的范围。基础数值与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应。

语法：

```
MonthsName(n_months, date[, period_no[, first_month_of_year]])
```

返回数据类型： 双

参数：

参数

参数	说明
n_months	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1（相当于 inmonth() 函数）、2（两个月）、3（相当于 inquarter() 函数）、4（四个月）或 6（半年）。
date	要求值的日期。
period_no	该周期可通过 **period_no** 偏移，其为整数，或解算为整数的表达式，其中值 0 表示该周期包含 **base_date**。**period_no** 为负数表示前几个时段，为正数则表示随后的几个时段。

参数	说明
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 first_month_of_year 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 DD/MM/YYYY。日期格式已经在数据加载脚本顶部的 SET DateFormat 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
monthsname(4, '19/10/2013')	返回 Sep-Dec 2013。 因为在此例和其他示例中，已将 SET Monthnames 语句设置为 Jan;Feb;Mar，以此类推。
monthsname(4, '19/10/2013', -1)	返回 May-Aug 2013。
monthsname(4, '19/10/2013', 0, 2)	返回 Oct-Jan 2014。 因为已将该年度指定为从 2 月开始，因此四个月周期在下一年度的第一个月结束。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

在此例中，对于表格中的每个发票日期，根据从该年度的两个月周期中的月份范围创建月份名称。范围为 4x2 个月的偏移(通过将 period_no 指定为 4)。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
MonthsName(2, InvDate, 4) AS MthsName
Resident TempTable;
```

```
Drop table TempTable;
```

结果列表包含原始日期和包括 monthsname() 函数的返回值的列。

结果表

InvDate	MthsName
28/03/2012	Nov-Dec 2012
10/12/2012	Jul-Aug 2013
5/2/2013	Sep-Oct 2013
31/3/2013	Nov-Dec2013
19/5/2013	Jan-Feb 2014
15/9/2013	May-Jun 2014
11/12/2013	Jul-Aug 2014
2/3/2014	Nov-Dec 2014
14/5/2014	Jan-Feb 2015
13/6/2014	Jan-Feb 2015
7/7/2014	Mar-Apr 2015
4/8/2014	Mar-Apr 2015

monthsstart

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

语法：

```
MonthsStart(n_months, date[, period_no [, first_month_of_year]])
```

返回数据类型： 双

参数：

参数

参数	说明
n_months	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月) 或 6(半年)。
date	要求值的日期。
period_no	该周期可通过 **period_no** 偏移，其为整数，或解算为整数的表达式，其中值 0 表示该周期包含 **base_date**。**period_no** 为负数表示前几个时段，为正数则表示随后的几个时段。

参数	说明
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 first_month_of_year 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 DD/MM/YYYY。日期格式已经在数据加载脚本顶部的 SET DateFormat 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
monthsstart(4, '19/10/2013')	返回 1/09/2013。
monthsstart(4, '19/10/2013, -1)	返回 01/05/2013。
monthsstart(4, '19/10/2013', 0, 2)	返回 01/10/2013。 因为该年度的开始变成 2 月。

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例用于查找每个发票日期的两个月周期的第一天，按一个两个月周期向前移动。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
MonthsStart(2, InvDate, 1) AS BiMthsStart
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 MonthsStart() 函数的返回值的列。

结果表

InvDate	BiMthsStart
28/03/2012	01/05/2012
10/12/2012	01/01/2013
5/2/2013	01/03/2013
31/3/2013	01/05/2013
19/5/2013	01/07/2013
15/9/2013	01/11/2013
11/12/2013	01/01/2014
2/3/2014	01/05/2014
14/5/2014	01/07/2014
13/6/2014	01/07/2014
7/7/2014	01/09/2014
4/8/2014	01/09/2014

monthstart

此函数用于返回与包含 **date** 的月份的第一天的第一毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
MonthStart(date[, period_no])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，如果为 0 或忽略，表示该月包含 **date**。**period_no** 为负数表示前几月，为正数则表示随后的几月。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
monthstart('19/10/2001')	返回 01/10/2001。
monthstart('19/10/2001', -1)	返回 01/09/2001。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例用于查找表格中每个发票日期的当月的第一天，其中 base_date 按四个月移动（通过将 period_no 指定为 4）。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
MonthStart(InvDate, 4) AS MthStart
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 monthstart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	MthStart
28/03/2012	01/07/2012
10/12/2012	01/04/2013
5/2/2013	01/06/2013
31/3/2013	01/07/2013
19/5/2013	01/09/2013

InvDate	MthStart
15/9/2013	01/01/2014
11/12/2013	01/04/2014
2/3/2014	01/07/2014
14/5/2014	01/09/2014
13/6/2014	01/10/2014
7/7/2014	01/11/2014
4/8/2014	01/12/2014

networkdays

networkdays 函数用于返回工作日的编号(周一至周五),在 **start_date** 和 **end_date** 之间,并将任何列出的可选 **holiday**考虑在内。

语法:

```
networkdays (start_date, end_date [, holiday])
```

返回数据类型: 整数

参数:

参数

参数	说明
start_date	评估的开始日期。
end_date	评估的结束日期。
holiday	从工作日排除假期。假期可表述为开始日期和结束日期,以逗号分隔。 **示例:** `'25/12/2013', '26/12/2013'` 您可以指定多个假期,以逗号分隔。 **示例:** `'25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014'`

示例和结果:
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
networkdays ('19/12/2013', '07/01/2014')	返回 14。以下示例没有将假期考虑在内。
networkdays ('19/12/2013', '07/01/2014', '25/12/2013', '26/12/2013')	返回 12。以下示例将 25/12/2013 至 26/12/2013 的假期考虑在内。
networkdays ('19/12/2013', '07/01/2014', '25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014')	返回 10。以下示例将两个假期考虑在内。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
PayTable:
LOAD recno() as InvID, * INLINE [
InvRec|InvPaid
28/03/2012|28/04/2012
10/12/2012|01/01/2013
5/2/2013|5/3/2013
31/3/2013|01/5/2013
19/5/2013|12/6/2013
15/9/2013|6/10/2013
11/12/2013|12/01/2014
2/3/2014|2/4/2014
14/5/2014|14/6/2014
13/6/2014|14/7/2014
7/7/2014|14/8/2014
4/8/2014|4/9/2014
] (delimiter is '|');
NrDays:
Load *,
NetWorkDays(InvRec,InvPaid) As PaidDays
Resident PayTable;
Drop table PayTable;
```

结果列表显示了为表格中的每条记录返回的 NetworkDays 值。

结果表

InvID	InvRec	InvPaid	PaidDays
1	28/03/2012	28/04/2012	23
2	10/12/2012	01/01/2013	17
3	5/2/2013	5/3/2013	21
4	31/3/2013	01/5/2013	23
5	19/5/2013	12/6/2013	18

InvID	InvRec	InvPaid	PaidDays
6	15/9/2013	6/10/2013	15
7	11/12/2013	12/01/2014	23
8	2/3/2014	2/4/2014	23
9	14/5/2014	14/6/2014	23
10	13/6/2014	14/7/2014	22
11	7/7/2014	14/8/2014	29
12	4/8/2014	4/9/2014	24

now

此函数用于返回系统时钟的当前时间的时间戳。默认值为 1。

语法：

```
now([ timer_mode])
```

返回数据类型：双

参数：

参数

参数	说明
timer_mode	可以具有以下值： 0(最后完成的数据加载的时间) 1(函数调用时的时间) 2(应用程序打开的时间) *如果在数据加载脚本中使用此函数，则 **timer_mode=0** 将会生成最后完成数据加载的时间，而 **timer_mode=1** 将会提供当前数据加载的函数调用时间。*

示例和结果：

脚本示例

示例	结果
now(0)	返回最后数据加载完成的时间。
now(1)	当在可视化表达式中使用时，此函数返回函数调用的时间。 当在数据加载脚本中使用时，此函数返回当前数据加载的函数调用时间。
now(2)	返回应用程序打开的时间。

quarterend

此函数用于返回与包含 **date** 的季度的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
QuarterEnd(date[, period_no[, first_month_of_year]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，其中值 0 表示该季度包含 **date**。**period_no** 为负数表示前几季，为正数则表示随后的几季。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`quarterend('29/10/2005')`	返回 31/12/2005 23:59:59。
`quarterend('29/10/2005', -1)`	返回 30/09/2005 23:59:59。
`quarterend('29/10/2005', 0, 3)`	返回 30/11/2005 23:59:59。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例用于查找表格中每个发票日期的季度的最后一天，其中将该年度的第一个月指定为 3 月。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
```

```
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
QuarterEnd(InvDate, 0, 3) AS QtrEnd
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 quarterend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	QtrEnd
28/03/2012	31/05/2012
10/12/2012	28/02/2013
5/2/2013	28/02/2013
31/3/2013	31/05/2013
19/5/2013	31/05/2013
15/9/2013	30/11/2013
11/12/2013	28/02/2014
2/3/2014	31/05/2014
14/5/2014	31/05/2014
13/6/2014	31/08/2014
7/7/2014	31/08/2014
4/8/2014	31/08/2014

quartername

此函数用于返回一个显示值，该值显示季度的月（根据 **MonthNames** 脚本变量的格式）以及年，伴随一个与该季度第一天第一毫秒的时间戳对应的基础数值。

语法：

```
QuarterName(date[, period_no[, first_month_of_year]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，其中值 0 表示该季度包含 **date**。**period_no** 为负数表示前几季，为正数则表示随后的几季。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：

脚本示例

示例	结果
`quartername('29/10/2013')`	返回 Oct-Dec 2013。
`quartername('29/10/2013', -1)`	返回 Jul-Sep 2013。
`quartername('29/10/2013', 0, 3)`	返回 Sep-Nov 2013。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

在此例中，对于表格中的每个发票日期，根据包含 *InvID* 的季度创建季度名称。将该年度的第一个月指定为 4 月。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
QuarterName(InvDate, 0, 4) AS QtrName
```

```
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 quartername() 函数的返回值的列。

结果表

InvDate	QtrName
28/03/2012	Jan-Mar 2011
10/12/2012	Oct-Dec 2012
5/2/2013	Jan-Mar 2012
31/3/2013	Jan-Mar 2012
19/5/2013	Apr-Jun 2013
15/9/2013	Jul-Sep 2013
11/12/2013	Oct-Dec 2013
2/3/2014	Jan-Mar 2013
14/5/2014	Apr-Jun 2014
13/6/2014	Apr-Jun 2014
7/7/2014	Jul-Sep 2014
4/8/2014	Jul-Sep 2014

quarterstart

此函数用于返回与包含 **date** 的季度的第一毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
QuarterStart(date[, period_no[, first_month_of_year]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，其中值 0 表示该季度包含 **date**。**period_no** 为负数表示前几季，为正数则表示随后的几季。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
quarterstart('29/10/2005')	返回 01/10/2005。
quarterstart('29/10/2005', -1)	返回 01/07/2005。
quarterstart('29/10/2005', 0, 3)	返回 01/09/2005。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例用于查找表格中每个发票日期的季度的第一天，其中将该年度的第一个月指定为 3 月。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
QuarterStart(InvDate, 0, 3) AS QtrStart
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 quarterstart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	QtrStart
28/03/2012	01/03/2012
10/12/2012	01/12/2012

InvDate	QtrStart
5/2/2013	01/12/2012
31/3/2013	01/03/2013
19/5/2013	01/03/2013
15/9/2013	01/09/2013
11/12/2013	01/12/2013
2/3/2014	01/03/2014
14/5/2014	01/03/2014
13/6/2014	01/06/2014
7/7/2014	01/06/2014
4/8/2014	01/06/2014

second

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示秒的整数。

语法：

```
second (expression)
```

返回数据类型： 整数

示例和结果：

脚本示例

示例	结果
`second( '09:14:36' )`	返回 36
`second( '0.5555' )`	返回 55(因为 0.5555 = 13:19:55)

setdateyear

此函数用于输入 **timestamp** 和 **year** , 并使用在输入中指定的 **year** 更新 **timestamp** 。

语法：

```
setdateyear (timestamp, year)
```

返回数据类型：双

参数：

参数

参数	说明
timestamp	标准的 Qlik Sense 时间戳(通常只是一个日期)。
year	一个四位数年份。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`setdateyear ('29/10/2005', 2013)`	返回“29/10/2013”
`setdateyear ('29/10/2005 04:26:14', 2013)`	返回“29/10/2013 04:26:14” 要在可视化中查看时间戳的时间部分，您必须将数字格式设置为日期，并选择用于显示时间值的格式的值。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
SetYear:
Load *,
SetDateYear(testdates, 2013) as NewYear
Inline [
testdates
1/11/2012
10/12/2012
1/5/2013
2/1/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];
```

结果列表包含原始日期和在其中已将年份设置为 2013 的列。

结果表

testdates	NewYear
1/11/2012	1/11/2013
10/12/2012	10/12/2013
2/1/2012	2/1/2013
1/5/2013	1/5/2013
19/5/2013	19/5/2013
15/9/2013	15/9/2013
11/12/2013	11/12/2013
2/3/2014	2/3/2013
14/5/2014	14/5/2013
13/6/2014	13/6/2013
7/7/2014	7/7/2013
4/8/2014	4/8/2013

setdateyearmonth

此函数用于输入 **timestamp**、**month** 和 **year**，并使用在输入中指定的 **year** 和 **month** 更新 **timestamp**。.

语法：

```
SetDateYearMonth (timestamp, year, month)
```

返回数据类型：双

参数：

参数

参数	说明
timestamp	标准的 Qlik Sense 时间戳(通常只是一个日期)。
year	一个四位数年份。
month	一位或两位数月份。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
setdateyearmonth ('29/10/2005', 2013, 3)	返回“29/03/2013”
setdateyearmonth ('29/10/2005 04:26:14', 2013, 3)	返回“29/03/2013 04:26:14” 要在可视化中查看时间戳的时间部分，您必须将数字格式设置为日期，并选择用于显示时间值的格式的值。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
SetYearMonth:
Load *,
SetDateYearMonth(testdates, 2013,3) as NewYearMonth
Inline [
testdates
1/11/2012
10/12/2012
2/1/2013
19/5/2013
15/9/2013
11/12/2013
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];
```

结果列表包含原始日期和在其中已将年份设置为 2013 的列。

结果表

testdates	NewYearMonth
1/11/2012	1/3/2013
10/12/2012	10/3/2013
2/1/2012	2/3/2013
19/5/2013	19/3/2013
15/9/2013	15/3/2013
11/12/2013	11/3/2013
14/5/2014	14/3/2013
13/6/2014	13/3/2013
7/7/2014	7/3/2013
4/8/2014	4/3/2013

timezone

此函数用于返回当前时区的名称，如 Windows 所定义。

语法：

```
TimeZone( )
```

返回数据类型：字符串

示例：

```
timezone( )
```

today

此函数用于返回系统时钟的当前日期。

语法：

```
today([ timer_mode])
```

返回数据类型：双

参数：

参数

参数	说明
timer_mode	可以具有以下值： 0(最后完成的数据加载的日子) 1(函数调用的日子) 2(应用程序打开的日子) *如果在数据加载脚本中使用此函数，则 **timer_mode=0** 将会生成最后完成数据加载的日期，而 **timer_mode=1** 将会提供当前数据加载的日期。*

示例和结果：

脚本示例

示例	结果
Today(0)	返回最后完成数据加载的日期。
Today(1)	当在可视化表达式中使用时，此函数返回函数调用的日期。 当在数据加载脚本中使用时，此函数返回当前数据加载开始时的日期。
Today(2)	返回应用程序打开的日期。

UTC

用于返回当前 Coordinated Universal Time。

语法：

```
UTC( )
```

返回数据类型： 双

示例：

```
utc( )
```

week

此函数用于返回根据 ISO 8601 表示周数的整数。周数根据标准数字解释通过表达式的日期解释进行计算。

语法：

```
week(timestamp [, first_week_day [, broken_weeks [, reference_day]]])
```

返回数据类型： 整数

参数

参数	说明
timestamp	作为时间戳或解析时间戳的表达式进行评估以转换的日期，例如 '2012-10-12'。
first_week_day	如果不指定 **first_week_day**，则变量 **FirstWeekDay** 的值将用作一周的第一天。 如果要使用其他天作为一周的第一天，请将 **first_week_day** 设置为： • 0，表示周一 • 1，表示周二 • 2，表示周三 • 3，表示周四 • 4，表示周五 • 5，表示周六 • 6，表示周日 此函数返回的整数现在将使用您使用 **first_week_day** 设置的一周的第一天。

参数	说明
broken_weeks	如果不指定 **broken_weeks**，则变量 **BrokenWeeks** 的值将用于定义周是否已中断。 默认情况下，Qlik Sense 函数使用连续的周。这意味着： • 在某些年份中，第 1 周在 12 月开始，而在其他年份中，第 52 或 53 周延续到 1 月。 • 在 1 月中，第 1 周始终至少有 4 天。 替代方法是使用不连续的周。 • 第 52 或 53 周不延续到 1 月。 • 第 1 周在 1 月 1 日开始，因此在大部分情况下不是完整的一周。 可以使用以下值： • 0（表示使用连续周） • 1（表示使用不连续周）
reference_day	如果不指定 **reference_day**，则变量 **ReferenceDay** 的值将用于定义将一月的哪一天设置为定义第 1 周的参考日。默认设置下，Qlik Sense 函数使用 4 作为参考日。这意味着第 1 周必须包含 1 月 4 日，换句话说，第 1 周始终至少具有 1 月份的前 4 天。 以下值可用于设置不同参考日： • 1（表示 1 月 1 日） • 2（表示 2 月 1 日） • 3（表示 3 月 1 日） • 4（表示 4 月 1 日） • 5（表示 5 月 1 日） • 6（表示 6 月 1 日） • 7（表示 7 月 1 日）

示例和结果：

脚本示例

示例	结果
`week( '2012-10-12' )`	返回 41。
`week( '35648' )`	返回 32，因为 35648 = 1997-08-06
`week('2012-10-12', 0, 1)`	返回 42

weekday

此函数用于返回包含以下名称的对偶值：

- 在环境变量 **DayNames** 中定义的日期名称。
- 介于 0-6 之间的整数对应于一周 (0-6) 的标定天。

语法：

```
weekday(date [,first_week_day=0])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
first_week_day	如果不指定 **first_week_day**，则变量 **FirstWeekDay** 的值将用作一周的第一天。 如果要使用其他天作为一周的第一天，请将 **first_week_day** 设置为： • 0，表示周一 • 1，表示周二 • 2，表示周三 • 3，表示周四 • 4，表示周五 • 5，表示周六 • 6，表示周日 此函数返回的整数现在将使用您使用 **first_week_day** 设置的一周的第一天作为基数 (0)。 *FirstWeekDay (page 141)*

示例：图表表达式
在以下示例中，**FirstWeekDay** 设置为 0(除非另有说明)。

脚本示例

示例	结果
`weekday( '1971-10-12' )`	返回“Tue”和 1
`weekday( '1971-10-12' , 6)`	返回“Tue”和 2。 在此示例中，我们使用 Sunday (6) 作为一周的第一天。

`SET FirstWeekDay = 6;`
 	
`...`
`weekday( '1971-10-12')`
	返回“Tue”和 2。

示例:加载脚本

加载脚本

*weekday*可用在加载脚本中以返回字符串以及表示星期几的数字,即使已经在脚本中设置了*FirstWeekDay*和*ReferenceDay*。下面的加载脚本包括特定的*FirstWeekDay*和*ReferenceDay*值,然后使用*weekday*通过*transaction_date*列中的数据返回表示星期几的字符串和数字。

在示出的结果中,*Day*列包含返回的字符串,而*Numeric value of Day*和*Numeric value of week starting from Sunday*包含返回的数值。在加载脚本中,*weekday*乘以 1,这是确保返回的数据类型为数值的简便方式。

在**数据加载编辑器**中,创建新的部分,然后添加示例脚本并运行它。然后,至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

```
SET DateFormat='DD/MM/YYYY';
SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT';
SET FirstWeekDay=0;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';

Transactions:
Load
*,
WeekDay(transaction_date) as [Day],
1*WeekDay(transaction_date) as [Numeric value of Day]
1*WeekDay(transaction_date, 6) as [Numeric value of a week starting from Sunday],
;
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 20180830, 12423.56, 23, 0,2038593, L, Red
3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180916, 15.75, 1, 0.22, 5646471, S, blue
3753, 20180922, 1251, 7, 0, 3036491, l, Black
3754, 20180922, 21484.21, 1356, 75, 049681, xs, Red
3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, Blue
3757, 20180923, 3177.4, 21, .14, 203521, XL, Black
];
```

结果

Qlik Sense表格示出在加载脚本中使用的weekday函数的结果。

transaction_id	transaction_date	日	日的数值	从周日开始的周的数值
3750	20180830	Thu	3	4
3751	20180907	周四	3	4

transaction_id	transaction_date	日	日的数值	从周日开始的周的数值
3752	20180916	Sat	5	6
3753	20180922	周五	4	5
3754	20180922	周五	4	5
3756	20180922	周五	4	5
3757	20180923	周六	5	6

weekend

此函数用于返回一个与包含 **date** 的日历周的最后一日(周日)最后一毫秒时间戳对应的值。默认输出格式为在脚本中设置的 **DateFormat**。

语法:

```
WeekEnd(date [, period_no[, first_week_day]])
```

返回数据类型: 双

参数:

参数

参数	说明
date	要求值的日期。
period_no	**shift** 为整数,其中值 0 表示该星期包含 **date**。shift 为负数,表示前几星期,正数表示随后的几星期。
first_week_day	指定一周的开始日期。如果忽略,使用 **FirstWeekDay** 变量的值。 可能的 **first_week_day** 值为: • 0, 表示周一 • 1, 表示周二 • 2, 表示周三 • 3, 表示周四 • 4, 表示周五 • 5, 表示周六 • 6, 表示周日 *FirstWeekDay (page 141)*

示例和结果:
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
weekend('10/01/2013')	返回 12/01/2013 23:59:59。
weekend('10/01/2013', -1)	返回 06/01/2013 23:59:59.
weekend('10/01/2013', 0, 1)	返回 14/01/2013 23:59:59。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找后跟每个发票日期的周的周中的最后一天。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
WeekEnd(InvDate, 1) AS WkEnd
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 weekend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	WkEnd
28/03/2012	08/04/2012
10/12/2012	23/12/2012
5/2/2013	17/02/2013
31/3/2013	07/04/2013
19/5/2013	26/05/2013

15/9/2013	22/09/2013
11/12/2013	22/12/2013
2/3/2014	09/03/2014
14/5/2014	25/05/2014
13/6/2014	22/06/2014
7/7/2014	20/07/2014
4/8/2014	17/08/2014

weekname

此函数用于返回一个值，显示带有与包含 **date** 的周的第一天的第一毫秒时间戳对应的基本数值对应的年份和周数。

语法：

```
WeekName(date[, period_no[,first_week_day]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**shift** 为整数，其中值 0 表示该星期包含 **date**。shift 为负数，表示前几星期，正数表示随后的几星期。
first_week_day	指定一周的开始日期。如果忽略，使用 **FirstWeekDay** 变量的值。 可能的 **first_week_day** 值为： • 0，表示周一 • 1，表示周二 • 2，表示周三 • 3，表示周四 • 4，表示周五 • 5，表示周六 • 6，表示周日 *FirstWeekDay (page 141)*

示例和结果：

示例	结果
weekname('12/01/2013')	返回 2013/02。
weekname('12/01/2013', -1)	返回 2013/01。
weekname('12/01/2013', 0, 1)	返回 2013/02。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

在此例中，对于表格中的每个发票日期，根据周所在的年度创建周名称，并将其与周编号进行相关联（通过将 period_no 指定为 1 按一周移动）。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
WeekName(InvDate, 1) AS WkName
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 weekname() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	WkName
28/03/2012	2012/14
10/12/2012	2012/51
5/2/2013	2013/07
31/3/2013	2013/14
19/5/2013	2013/21

InvDate	WkName
15/9/2013	2013/38
11/12/2013	2013/51
2/3/2014	2014/10
14/5/2014	2014/21
13/6/2014	2014/25
7/7/2014	2014/29
4/8/2014	2014/33

weekstart

此函数用于返回与包含 **date** 的日历周的第一天(周一)的第一毫秒时间戳对应的值。默认输出格式是在脚本中设置的 **DateFormat**。

语法:

```
WeekStart(date [, period_no[, first_week_day]])
```

返回数据类型: 双

参数:

参数

参数	说明
date	要求值的日期。
period_no	**shift** 为整数,其中值 0 表示该星期包含 **date**。shift 为负数,表示前几星期,正数表示随后的几星期。
first_week_day	指定一周的开始日期。如果忽略,使用 **FirstWeekDay** 变量的值。 可能的 **first_week_day** 值为: • 0, 表示周一 • 1, 表示周二 • 2, 表示周三 • 3, 表示周四 • 4, 表示周五 • 5, 表示周六 • 6, 表示周日 *FirstWeekDay (page 141)*

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
weekstart('12/01/2013')	返回 07/01/2013。
weekstart('12/01/2013', -1)	返回 31/11/2012。
weekstart('12/01/2013', 0, 1)	返回 08/01/2013。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找后跟每个发票日期的周的第一天。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
WeekStart(InvDate, 1) AS WkStart
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 weekstart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	WkStart
28/03/2012	02/04/2012
10/12/2012	17/12/2012
5/2/2013	11/02/2013

InvDate	WkStart
31/3/2013	01/04/2013
19/5/2013	20/05/2013
15/9/2013	16/09/2013
11/12/2013	16/12/2013
2/3/2014	03/03/2014
14/5/2014	19/05/2014
13/6/2014	16/06/2014
7/7/2014	14/07/2014
4/8/2014	11/08/2014

weekyear

此函数用于返回根据 ISO 8601 周数所属的年份。星期数范围在 1 和大约 52 之间。

语法：

```
weekyear(expression)
```

返回数据类型： 整数

示例和结果：

脚本示例

示例	结果
weekyear('1996-12-30')	返回 1997，因为 1997 的第 1 周从 1996/12/30 星期一 开始
weekyear('1997-01-02')	返回 1997
weekyear('1997-12-28')	返回 1997
weekyear('1997-12-30')	返回 1998，因为 1998 的第 1 周从 1997/12/29 星期一 开始
weekyear('1999-01-02')	返回 1998，因为 1998 的第 53 周从 1999-01-03 开始

限制：

部分年份第 1 周始于十二月，例如，1997 年 12 月。其他年份始于上一年的第 53 周，例如 1999 年 1 月。对于少数日子，相应星期序数可能属于其他年份，此时函数 **year** 和 **weekyear** 将返回不同的值。

year

此函数用于根据标准数字解释当 **expression** 被解释为日期时返回一个表示年份的整数。

语法：

```
year(expression)
```

返回数据类型：整数

示例和结果：

脚本示例

示例	结果
year('2012-10-12')	返回 2012
year('35648')	返回 1997, 因为 35648 = 1997-08-06

yearend

此函数用于返回与包含 **date** 的年份的最后一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
YearEnd( date[, period_no[, first_month_of_year = 1]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，其中值 0 表示该年份包含 **date**。**period_no** 为负数表示前几年，为正数表示随后几年。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
yearend ('19/10/2001')	返回 31/12/2001 23:59:59。
yearend ('19/10/2001', -1)	返回 31/12/2000 23:59:59。
yearend ('19/10/2001', 0, 4)	返回 31/03/2002 23:59:59。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找每个发票日期的年份的最后一天，其中将该年度的第一个月指定为 4 月。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
YearEnd(InvDate, 0, 4) AS YrEnd
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 yearend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	YrEnd
28/03/2012	31/03/2011
10/12/2012	31/03/2012
5/2/2013	31/03/2013
31/3/2013	31/03/2013
19/5/2013	31/03/2014
15/9/2013	31/03/2014
11/12/2013	31/03/2014
2/3/2014	31/03/2014
14/5/2014	31/03/2015

13/6/2014	31/03/2015
7/7/2014	31/03/2015
4/8/2014	31/03/2015

yearname

此函数用于返回一个四位数年份的显示值，带有与包含 **date** 的年份的第一天的第一毫秒时间戳对应的基本数值。

语法：

```
YearName(date[, period_no[, first_month_of_year]] )
```

返回数据类型：双

参数：

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，其中值 0 表示该年份包含 **date**。**period_no** 为负数表示前几年，为正数表示随后几年。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。该显示值将为一个字符串，表示两年。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`yearname ( '19/10/2001')`	返回 2001。
`yearname ( '19/10/2001', -1 )`	返回 2000。
`yearname ( '19/10/2001', 0, 4)`	返回 2001-2002。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找每个发票日期的年份的第一天，其中将该年度的第一个月指定为 4 月。

以下示例为在表格中找到的每个发票日期的年份创建四加四位数的名称。这是因为已将该年度的第一个月指定为 4 月。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
YearName(InvDate, 0, 4) AS YrName
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 yearname() 函数的返回值的列。

结果表

InvDate	YrName
28/03/2012	2011-2012
10/12/2012	2012-2013
5/2/2013	2012-2013
31/3/2013	2012-2013
19/5/2013	2013-2014
15/9/2013	2013-2014
11/12/2013	2013-2014
2/3/2014	2013-2014
14/5/2014	2014-2015
13/6/2014	2014-2015
7/7/2014	2014-2015
4/8/2014	2014-2015

yearstart

此函数用于返回与包含 **date** 的年份的第一天的开始时间对应的时间戳。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
YearStart(date[, period_no[, first_month_of_year]])
```

返回数据类型：双

参数：

参数

参数	说明
date	要求值的日期。
period_no	**period_no** 为整数，其中值 0 表示该年份包含 **date**。**period_no** 为负数表示前几年，为正数表示随后几年。
first_month_of_year	如果您不想从一月开始处理(财政)年，可在 **first_month_of_year** 中指定一个介于 2 和 12 之间的值。

示例和结果：
以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

脚本示例

示例	结果
`yearstart ('19/10/2001')`	返回 01/01/2001。
`yearstart ('19/10/2001', -1)`	返回 01/01/2000。
`yearstart ('19/10/2001', 0, 4)`	返回 01/04/2001。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

以下示例在表格中查找每个发票日期的年份的第一天，其中将该年度的第一个月指定为 4 月。

```
TempTable:
LOAD RecNo() as InvID, * Inline [
InvDate
28/03/2012
10/12/2012
5/2/2013
31/3/2013
19/5/2013
```

```
15/9/2013
11/12/2013
2/3/2014
14/5/2014
13/6/2014
7/7/2014
4/8/2014
];

InvoiceData:
LOAD *,
YearStart(InvDate, 0, 4) AS YrStart
Resident TempTable;
Drop table TempTable;
```

结果列表包含原始日期和包括 yearstart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。

结果表

InvDate	YrStart
28/03/2012	01/04/2011
10/12/2012	01/04/2012
5/2/2013	01/04/2012
31/3/2013	01/04/2012
19/5/2013	01/04/2013
15/9/2013	01/04/2013
11/12/2013	01/04/2013
2/3/2014	01/04/2013
14/5/2014	01/04/2014
13/6/2014	01/04/2014
7/7/2014	01/04/2014
4/8/2014	01/04/2014

yeartodate

此函数用于判断输入时间戳是否在最后加载脚本的日期的年份以内，并返回 True(如果在) 或返回 False(如果不在) 。

语法：

```
YearToDate(timestamp[ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

返回数据类型： 布尔值

如果未使用可选参数，年初至今指日历年中 1 月 1 日以后任何一天，包括最近一次脚本执行日期。

参数：

参数

参数	说明
timestamp	评估的时间戳，如“2012-10-12”。
yearoffset	通过指定 **yearoffset**，**yeartodate** 对于其他年份的同一时期返回 True。**yearoffset** 为负表示上一年，偏移量为正表示下一年。通过指定 yearoffset = -1 获得至今的最近一年。如果忽略，则假设为 0。
firstmonth	通过在 1 和 12 之间（如果省略，则为 1）指定 **firstmonth**，年初可移动到任何一个月的第一天。例如，如果您想要从 5 月 1 日开始的财政年工作，请指定 **firstmonth** = 5。
todaydate	通过指定一个 **todaydate**（如果忽略执行上次脚本时间戳），这可作为该时期的上限移动该日。

示例和结果：
下例假设上次重新加载时间 = 2011-11-18

脚本示例

示例	结果
`yeartodate( '2010-11-18')`	返回 False
`yeartodate( '2011-02-01')`	返回 True
`yeartodate( '2011-11-18')`	返回 True
`yeartodate( '2011-11-19')`	返回 False
`yeartodate( '2011-11-19', 0, 1, '2011-12-31')`	返回 True
`yeartodate( '2010-11-18', -1)`	返回 True
`yeartodate( '2011-11-18', -1)`	返回 False
`yeartodate( '2011-04-30', 0, 5)`	返回 False
`yeartodate( '2011-05-01', 0, 5)`	返回 True

5.8 指数和对数函数

本部分介绍与指数计算和对数计算相关的函数。所有函数均可用于数据加载脚本和图表表达式。

在以下函数中，参数为表达式，其中 **x** 和 **y** 应解释为实值数。

exp

自然指数函数 e^x，使用自然对数 **e** 作为底数。结果为正数。

```
exp(x )
```

示例和结果：

exp(3) 返回 20.085。

log

x 的自然对数。仅在 **x**> 0 时才可定义此函数。结果为数字。

```
log(x )
```

示例和结果：

log(3) 返回 1.0986

log10

x 的常用对数(以 10 为底)。仅在 **x**> 0 时才可定义此函数。结果为数字。

```
log10(x )
```

示例和结果：

log10(3) 返回 0.4771

pow

返回 **x** 的 **y** 次幂。结果为数字。

```
pow(x,y )
```

示例和结果：

pow(3, 3) 返回 27

sqr

x 平方(**x** 的 2 次幂)。结果为数字。

```
sqr (x )
```

示例和结果：

sqr(3) 返回 9

sqrt

x 的平方根。仅在 **x** >= 0 时才可定义此函数。结果为正数。

```
sqrt(x )
```

示例和结果：

sqrt(3) 返回 1.732

5.9 字段函数

这些函数只可用于图表表达式中。

字段函数可返回整数或字符串，以便确定不同的字段选择项情况。

计数函数

GetAlternativeCount

GetAlternativeCount()用于查找标识字段中可能(浅灰色)值的数量。

```
GetAlternativeCount - 图表函数 (field_name)
```

GetExcludedCount

GetExcludedCount() 用于查找标识字段中排除相异值的数量。排除的值包括替代项(浅灰)、排除项(深灰)以及所选排除项(带复选标记的深灰)字段。

```
GetExcludedCount - 图表函数 (page 522)(field_name)
```

GetNotSelectedCount

此图表函数用于返回名为 **fieldname** 的字段中非选定值的数量。字段必须处于“和”模式以使此函数相关。

```
GetNotSelectedCount - 图表函数(fieldname [, includeexcluded=false])
```

GetPossibleCount

GetPossibleCount() 用于查找标识字段中可能值的数量。如果标识字段包括选择项，则计算选定(绿色)值的数量。否则，计算相关(白色)值的数量。

```
GetPossibleCount - 图表函数(field_name)
```

GetSelectedCount

GetSelectedCount() 用于查找字段中选定(绿色)值的数量。

```
GetSelectedCount - 图表函数 (field_name [, include_excluded])
```

字段和选择项函数

GetCurrentSelections

GetCurrentSelections() 返回应用程序中的当前选择列表。如果改为在搜索框中使用搜索字符串进行选择，则 **GetCurrentSelections()** 返回字符串。

```
GetCurrentSelections - 图表函数([record_sep [,tag_sep [,value_sep [,max_
values]]]])
```

GetFieldSelections

GetFieldSelections() 用于返回包含字段内当前选择项的**字符串**。

```
GetFieldSelections - 图表函数 ( field_name [, value_sep [, max_values]])
```

GetObjectDimension

GetObjectDimension() 返回维度的名称。**Index**(索引) 是一个可选整数 , 表明应返回的维度。

```
GetObjectDimension - 图表函数 ([index])
```

GetObjectField

GetObjectField() 返回维度的名称。**Index**(索引) 是一个可选整数 , 表明应返回的维度。

```
GetObjectField - 图表函数 ([index])
```

GetObjectMeasure

GetObjectMeasure() 返回度量的名称。**Index**(索引) 是一个可选整数 , 表明应返回的度量。

```
GetObjectMeasure - 图表函数 ([index])
```

GetAlternativeCount - 图表函数

GetAlternativeCount()用于查找标识字段中可能 (浅灰色) 值的数量。

语法 :

```
GetAlternativeCount (field_name)
```

返回数据类型 : 整数

参数 :

参数

参数	说明
field_name	包含要度量的数据范围的字段。

示例和结果 :

以下示例使用加载到筛选器窗格的 **First name** 字段。

示例和结果

示例	结果
假定选择 **John**(在 **First name** 中) 。 `GetAlternativeCount ([First name])`	4, 因为 **First name** 中有 4 个唯一的排除 (灰色) 值。
假定已选择 **John** 和 **Peter**。 `GetAlternativeCount ([First name])`	3, 因为 **First name** 中有 3 个唯一的排除 (灰色) 值。
假定未在 **First name** 中选择任何值。 `GetAlternativeCount ([First name])`	0, 因为没有选择项。

示例中所使用的数据 :

```
Names:
LOAD * inline [
```

```
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetCurrentSelections - 图表函数

GetCurrentSelections() 返回应用程序中的当前选择列表。如果改为在搜索框中使用搜索字符串进行选择，则 **GetCurrentSelections()** 返回字符串。

如果使用选项，您需要指定 record_sep。要指定新行，请将 **record_sep** 设置为 **chr(13)&chr(10)**。

如果选择除两个值以外的所有值，或除一个值以外的所有值，则分别使用格式“NOT x,y”或“NOT y”。如果选择全部值，并且全部值的计数大于 max_values，将返回文本 ALL。

语法：

```
GetCurrentSelections ([record_sep [, tag_sep [, value_sep [, max_values [, state_name]]]]])
```

返回数据类型： 字符串

参数：

参数

参数	说明
record_sep	要置于两个字段记录之间的分隔符。默认分隔符为 <CR><LF>，表示新行。
tag_sep	要置于字段名标记和字段值之间的分隔符。默认分隔符为“: ”。
value_sep	置于字段值之间的分隔符。默认分隔符为“,”。
max_values	将会单独列出字段值的最大数字。当选择更大的字段值数量时，会改用“x 个值，共 y 个”格式。默认值为 6。
state_name	已为特定可视化选择的备用状态的名称。如果已使用 **state_name** 参数，则将只考虑与指定状态名称关联的选择。

示例和结果：

以下示例使用两个加载到不同筛选器窗格的字段，一个用于 **First name** 名称，另一个用于 **Initials**。

示例和结果

示例	结果
假定选择 **John**(在 **First name** 中)。 `GetCurrentSelections ()`	'First name: John'
假定已在 **First name** 中选择 **John** 和 **Peter**。 `GetCurrentSelections ()`	'First name: John, Peter'
假定在 **First name** 中选择 **John** 和 **Peter**, 并在 **Initials** 中选择 **JA**。 `GetCurrentSelections ()`	'First name: John, Peter Initials: JA'
假定在 **First name** 中选择 **John**, 并在 **Initials** 中选择 **JA**。 `GetCurrentSelections ( chr(13)&chr(10) , ' = ' )`	'First name = John Initials = JA'
假定已在 **First name** 中选择除 Sue 以外的所有名称, 并且在 **Initials** 中没有选择项。 `GetCurrentSelections (chr(13)&chr(10), '=', ',' ,3)`	'First name=NOT Sue'

示例中所使用的数据:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetExcludedCount - 图表函数

GetExcludedCount() 用于查找标识字段中排除相异值的数量。排除的值包括替代项(浅灰)、排除项(深灰)以及所选排除项(带复选标记的深灰)字段。

语法:

```
GetExcludedCount (field_name)
```

返回数据类型: 字符串

参数:

参数

参数	说明
field_name	包含要度量的数据范围的字段。

示例和结果：

以下示例使用三个加载到不同筛选器窗格的字段，一个用于 **First name**，一个用于 **Last name**，另一个用于 **Initials**。

示例和结果

示例	结果
假定未在 **First name** 中选择任何值。	`GetExcludedCount (Initials) = 0` 不存在选择项。
如果在 **First name** 中选择了 **John**。	`GetExcludedCount (Initials) = 5` 在**姓名中的大写字母**中有 5 个深灰色的排除值。第六个单元格 (JA) 为白色，因为它与 **First name** 中的选择项 John 关联。
如果选择了 **John** 和 **Peter**。	`GetExcludedCount (Initials) = 3` 在 **Initials** 中，John 与值 1 关联并且 Peter 与值 2 关联。
如果在 **First name** 中选择了 **John** 和 **Peter**，然后在 **Last name** 中选择了 **Franc**。	`GetExcludedCount ([First name]) = 4` 在**名字**中有深灰色的 4 个排除值。**GetExcludedCount()** 对带排除值的字段进行运算，包括替代项和选择的排除的字段。
如果在 **First name** 中选择了 **John** 和 **Peter**，然后在 **Last name** 中选择了 **Franc** 和 **Anderson**。	`GetExcludedCount (Initials) = 4` 在**姓名中的大写字母**中有 4 个深灰色的排除值。其他两个单元(JA 和 PF) 将为白色，因为它们与选择项 John 和 **First name** 中的 Peter 关联。
如果在 **First name** 中选择了 **John** 和 **Peter**，然后在 **Last name** 中选择了 **Franc** 和 **Anderson**。	`GetExcludedCount ([Last name]) = 4` 在**姓名中的大写字母**中有 4 个排除值。Devonshire 具有浅灰色，而 Brown、Carr 和 Elliot 具有深灰色。

示例中所使用的数据：

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetFieldSelections - 图表函数

GetFieldSelections() 用于返回包含字段内当前选择项的**字符串**。

如果选择除两个值以外的所有值，或除一个值以外的所有值，则分别使用格式“NOT x,y”或“NOT y”。如果选择全部值，并且全部值的计数大于 max_values，将返回文本 ALL。

语法：

```
GetFieldSelections ( field_name [, value_sep [, max_values [, state_name]]])
```

返回数据类型： 字符串

返回字符串格式

格式	描述
'a, b, c'	如果选定值的数量等于或小于 max_values，则返回的字符串是选定值的列表。 这些值用 value_sep 作为分隔符分隔。
'NOT a, b, c'	如果未选定值的数量等于或小于 max_values，则返回的字符串是未选定值的列表，以 NOT 作为前缀。 这些值用 value_sep 作为分隔符分隔。
'x of y'	x = 选定值的数目 y = 值的总数 这将在 max_values < x < (y - max_values) 返回。
'ALL'	在选定了所有值时返回。
'-'	在未选定值时返回。
<search string>	如果选择了使用搜索，则返回搜索字符串。

参数：

参数

参数	说明
field_name	包含要度量的数据范围的字段。
value_sep	置于字段值之间的分隔符。默认分隔符为“,”。
max_values	将会单独列出字段值的最大数字。当选择更大的字段值数量时，会改用“x 个值，共 y 个”格式。默认值为 6。
state_name	已为特定可视化选择的备用状态的名称。如果已使用 **state_name** 参数，则将只考虑与指定状态名称关联的选择。

示例和结果：

以下示例使用加载到筛选器窗格的 **First name** 字段。

示例和结果

示例	结果
假定选择 **John**(在 **First name** 中) 。 `GetFieldSelections ([First name])`	“John”
假定已选择 **John** 和 **Peter**。 `GetFieldSelections ([First name])`	“John,Peter”
假定已选择 **John** 和 **Peter**。 `GetFieldSelections ([First name],'; ')`	“John; Peter”
假定已在 **First name** 中选择 **John**、**Sue**、**Mark**。 `GetFieldSelections ([First name],';',2)`	“NOT Jane;Peter”, 因为值 2 被表述为 max_values 参数的值。否则, 结果将为 John; Sue; Mark.

示例中所使用的数据:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetNotSelectedCount - 图表函数

此图表函数用于返回名为 **fieldname** 的字段中非选定值的数量。字段必须处于“和”模式以使此函数相关。

语法:

```
GetNotSelectedCount(fieldname [, includeexcluded=false])
```

参数:

参数

参数	说明
fieldname	要求值的字段的名称。
includeexcluded	如果 **includeexcluded** 表述为 True, 计数将包括在一个其他字段中被选择项排除的选定值。

示例：

```
GetNotSelectedCount( Country )
GetNotSelectedCount( Country, true )
```

GetObjectDimension - 图表函数

GetObjectDimension() 返回维度的名称。**Index**(索引) 是一个可选整数，表明应返回的维度。

您无法在以下位置使用该函数：标题、副标题、页脚、基准线表达式。

您无法在使用 Object ID 的另一对象中引用维度或度量的名称。

语法：

```
GetObjectDimension ([index])
```

示例：

```
GetObjectDimension(1)
```

示例：图表表达式

Qlik Sense 表以图表表达式显示了 GetObjectDimension 函数的示例

transaction_date	Customer ID	transaction_quantity	=GetObjectDimension ()	=GetObjectDimension (0)	=GetObjectDimension (1)
2018/08/30	049681	13	transaction_date	transaction_date	CustomerID
2018/08/30	203521	6	transaction_date	transaction_date	CustomerID
2018/08/30	203521	21	transaction_date	transaction_date	CustomerID

如果您希望返回度量的名称，请改为使用 **GetObjectMeasure** 函数。

GetObjectField - 图表函数

GetObjectField() 返回维度的名称。**Index**(索引) 是一个可选整数，表明应返回的维度。

您无法在以下位置使用该函数：标题、副标题、页脚、基准线表达式。

您无法在使用 Object ID 的另一对象中引用维度或度量的名称。

语法:

```
GetObjectField ([index])
```

示例:

```
GetObjectField(1)
```

示例:图表表达式

Qlik Sense 表以图表表达式显示了 GetObjectField 函数的示例。

transaction_date	CustomerID	transaction_quantity	GetObjectField	=GetObjectField(0)	=GetObjectField(1)
2018/08/30	049681	13	transaction_date	transaction_date	CustomerID
2018/08/30	203521	6	transaction_date	transaction_date	CustomerID
2018/08/30	203521	21	transaction_date	transaction_date	CustomerID

如果您希望返回度量的名称,请改为使用 **GetObjectMeasure** 函数。

GetObjectMeasure - 图表函数

GetObjectMeasure() 返回度量的名称。**Index**(索引) 是一个可选整数,表明应返回的度量。

您无法在以下位置使用该函数:标题、副标题、页脚、基准线表达式。

您无法在使用 Object ID 的另一对象中引用维度或度量的名称。

语法:

```
GetObjectMeasure ([index])
```

示例:

```
GetObjectMeasure(1)
```

示例:图表表达式

Qlik Sense 表以图表表达式显示了 GetObjectMeasure 函数的示例

Customer ID	sum (transaction_quantity)	Avg (transaction_quantity)	=GetObjectMeasure ()	=GetObjectMeasure(0)	=GetObjectMeasure(1)
49681	13	13	sum(transaction_quantity)	sum(transaction_quantity)	Avg(transaction_quantity)
203521	27	13.5	sum(transaction_quantity)	sum(transaction_quantity)	Avg(transaction_quantity)

如果希望返回维度的名称，可改为使用 **GetObjectField** 函数。

GetPossibleCount - 图表函数

GetPossibleCount() 用于查找标识字段中可能值的数量。如果标识字段包括选择项，则计算选定(绿色)值的数量。否则，计算相关(白色)值的数量。.

对于有选择项的字段，**GetPossibleCount()** 将返回所选(绿色)字段的数量。

返回数据类型： 整数

语法：

```
GetPossibleCount (field_name)
```

参数：

参数

参数	说明
field_name	包含要度量的数据范围的字段。

示例和结果：

以下示例使用两个加载到不同筛选器窗格的字段，一个用于 **First name** 名称，另一个用于 **Initials**。

示例和结果

示例	结果
假定选择 **John**(在 **First name** 中)。 `GetPossibleCount ([Initials])`	1，因为 Initials 中有 1 个值与 **First name** 中的选择项 **John** 关联。
假定选择 **John**(在 **First name** 中)。 `GetPossibleCount ([First name])`	1，因为 **First name** 中有 1 个选择项 **John**。
假定选择 **Peter**(在 **First name** 中)。 `GetPossibleCount ([Initials])`	2，因为 Peter 与 **Initials** 中的 2 个值关联。

示例	结果
假定未在 **First name** 中选择任何值。 `GetPossibleCount ([First name])`	5, 因为没有选择项, 并且 **First name** 中有 5 个唯一的值。
假定未在 **First name** 中选择任何值。 `GetPossibleCount ([Initials])`	6, 因为没有选择项, 并且 **Initials** 中有 6 个唯一的值。

示例中所使用的数据:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetSelectedCount - 图表函数

GetSelectedCount() 用于查找字段中选定(绿色)值的数量。

语法:

```
GetSelectedCount (field_name [, include_excluded [, state_name]])
```

返回数据类型: 整数

参数:

参数

参数	说明
field_name	包含要度量的数据范围的字段。
include_ excluded	如果设置为 **True()**, 则计数会包括所选值, 尽管这些值当前排除在其他字段选择项之外。如果为 False 或省略, 则这些值不会包括在内。
state_ name	已为特定可视化选择的备用状态的名称。如果已使用 **state_name** 参数, 则将只考虑与指定状态名称关联的选择。

示例和结果:

以下示例使用三个加载到不同筛选器窗格的字段, 一个用于 **First name** 名称, 一个用于 **Initials**, 另一个用于 **Has cellphone**。

示例和结果

示例	结果
假定选择 **John**(在 **First name** 中) 。 `GetSelectedCount ([First name])`	1, 因为已在 **First name** 中选择一个值。
假定选择 **John**(在 **First name** 中) 。 `GetSelectedCount ([Initials])`	0, 因为未在 **Initials** 中选择任何值。
在 **First name** 中没有选择任何选择项, 在 **Initials** 中选择所有值, 之后在 **Has cellphone** 中选择值 **Yes**。 `GetSelectedCount ([Initials], True())`	6. 虽然带有 **Initials**MC 和 PD 的选择项的 **Has cellphone** 设置为 **No**, 但结果仍是 6, 因为参数 include_excluded 已设置为 True()。

示例中所使用的数据:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

5.10 文件函数

文件函数(只在脚本表达式中可用) 返回有关当前阅读的表格文件的信息。这些函数对所有数据源来说都返回 NULL, 除了表格文件(例外 : **ConnectString( )**) 。

文件函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Attribute

此脚本函数用于返回不同媒体文件的元标签的值作为文本。支持以下格式 : MP3、WMA、WMV、PNG 和 JPG。如文件 **filename** 不存在, 则它不是一种支持的文件格式或不包含一个称为 **attributename** 的元标签, 将会返回 NULL 值。

```
Attribute (filename, attributename)
```

ConnectString

ConnectString() 函数用于返回 ODBC 或 OLE DB 连接的活动数据连接的名称。此函数用于返回在没有执行 **connect** 语句时或在执行 **disconnect** 语句后返回空字符串。

```
ConnectString ()
```

FileBaseName

FileBaseName 函数返回一个字符串，其中包含当前正在读取的表格文件的名称，没有路径或扩展名。

```
FileBaseName ()
```

FileDir

FileDir 函数用于返回一个包含至当前阅读表格文件目录的路径。

```
FileDir ()
```

FileExtension

FileExtension 函数用于返回一个包含至当前阅读表格文件扩展名的字符串。

```
FileExtension ()
```

FileName

FileName 函数返回一个字符串，其中包含当前正在读取的表格文件的名称，没有路径或扩展名。

```
FileName ()
```

FilePath

FilePath 函数用于返回一个包含至当前阅读表格文件的完整路径的字符串。

```
FilePath ()
```

FileSize

FileSize 函数用于返回一个包含文件 filename 字节大小的整数，或如果未指定 filename，则返回一个包含当前阅读的表格文件字节大小的整数。

```
FileSize ()
```

FileTime

FileTime 函数用于返回文件 filename 的上一次修改日期和时间的时间戳。如果未指定 filename，则此函数将参考当前阅读的表格文件。

```
FileTime ([ filename ])
```

GetFolderPath

GetFolderPath 函数用于返回 Microsoft Windows *SHGetFolderPath* 函数的值。此函数用于输入 Microsoft Windows 文件夹的名称，并返回该文件夹的完整路径。

```
GetFolderPath ()
```

QvdCreateTime

此脚本函数用于返回 QVD 文件的 XML-标题时间戳(如果有)，否则返回 NULL 值。

```
QvdCreateTime (filename)
```

QvdFieldName

此脚本函数返回 QVD 文件中的字段编号 **fieldno**。如果字段不存在，则返回 NULL。

```
QvdFieldName (filename , fieldno)
```

QvdNoOfFields

此脚本函数用于返回 QVD 文件中的字段数。

```
QvdNoOfFields (filename)
```

QvdNoOfRecords

此脚本函数用于返回 QVD 文件中的当前记录数。

```
QvdNoOfRecords (filename)
```

QvdTableName

此脚本函数用于返回存储在 QVD 文件中的表格名称。

```
QvdTableName (filename)
```

Attribute

此脚本函数用于返回不同媒体文件的元标签的值作为文本。支持以下格式：MP3、WMA、WMV、PNG 和 JPG。如文件 **filename** 不存在，则它不是一种支持的文件格式或不包含一个称为 **attributename** 的元标签，将会返回 NULL 值。

语法：

```
Attribute(filename, attributename)
```

可以读取大多数元标签。本主题中的示例显示了可以为相应的支持文件类型读取的标签。

*根据相关规范，您只能读取保存在文件中的元标签，例如 ID2v3（MP3 文件）或 EXIF（JPG 文件），而不能读取 **Windows File Explorer** 中的元信息。*

参数：

参数

参数	说明
filename	如有必要，媒体文件名称包括路径作为文件夹数据连接。 **示例：** ***'lib://Table Files/'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** ***c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** ***data***
attributename	元标签的名称。

以下示例使用 **GetFolderPath** 函数查找指向媒体文件的路径。因为在旧模式下仅支持 **GetFolderPath**，当您在标准模式或 Qlik Sense SaaS 中使用该功能时，您需要将对 **GetFolderPath** 的引用替换为 lib:// 数据连接路径。

文件系统访问限制 (page 759)

Example 1: MP3 文件

此脚本读取文件夹 *MyMusic* 中所有可能的 MP3 元标签。

```
// Script to read MP3 meta tags for each vExt in 'mp3' for each vFoundFile in filelist(
GetFolderPath('MyMusic') & '\*.'& vExt ) FileList: LOAD FileLongName,     subfield
(FileLongName,'\',-1) as FileShortName,     num(FileSize(FileLongName),'# ### ### ###',',','
') as FileSize,     FileTime(FileLongName) as FileTime,      // ID3v1.0 and ID3v1.1 tags
Attribute(FileLongName, 'Title') as Title,     Attribute(FileLongName, 'Artist') as Artist,
 Attribute(FileLongName, 'Album') as Album,     Attribute(FileLongName, 'Year') as Year,
Attribute(FileLongName, 'Comment') as Comment,     Attribute(FileLongName, 'Track') as Track,
   Attribute(FileLongName, 'Genre') as Genre,

     // ID3v2.3 tags     Attribute(FileLongName, 'AENC') as AENC, // Audio encryption
Attribute(FileLongName, 'APIC') as APIC, // Attached picture     Attribute(FileLongName,
'COMM') as COMM, // Comments     Attribute(FileLongName, 'COMR') as COMR, // Commercial frame
   Attribute(FileLongName, 'ENCR') as ENCR, // Encryption method registration     Attribute
(FileLongName, 'EQUA') as EQUA, // Equalization     Attribute(FileLongName, 'ETCO') as ETCO,
// Event timing codes     Attribute(FileLongName, 'GEOB') as GEOB, // General encapsulated
object     Attribute(FileLongName, 'GRID') as GRID, // Group identification registration
Attribute(FileLongName, 'IPLS') as IPLS, // Involved people list     Attribute(FileLongName,
'LINK') as LINK, // Linked information     Attribute(FileLongName, 'MCDI') as MCDI, // Music
CD identifier     Attribute(FileLongName, 'MLLT') as MLLT, // MPEG location lookup table
Attribute(FileLongName, 'OWNE') as OWNE, // Ownership frame     Attribute(FileLongName,
```

```
'PRIV') as PRIV, // Private frame     Attribute(FileLongName, 'PCNT') as PCNT, // Play counter
    Attribute(FileLongName, 'POPM') as POPM, // Popularimeter

     Attribute(FileLongName, 'POSS') as POSS, // Position synchronisation frame     Attribute
(FileLongName, 'RBUF') as RBUF, // Recommended buffer size     Attribute(FileLongName, 'RVAD')
as RVAD, // Relative volume adjustment     Attribute(FileLongName, 'RVRB') as RVRB, // Reverb
   Attribute(FileLongName, 'SYLT') as SYLT, // Synchronized lyric/text     Attribute
(FileLongName, 'SYTC') as SYTC, // Synchronized tempo codes     Attribute(FileLongName,
'TALB') as TALB, // Album/Movie/Show title     Attribute(FileLongName, 'TBPM') as TBPM, // BPM
(beats per minute)     Attribute(FileLongName, 'TCOM') as TCOM, // Composer     Attribute
(FileLongName, 'TCON') as TCON, // Content type     Attribute(FileLongName, 'TCOP') as TCOP,
// Copyright message     Attribute(FileLongName, 'TDAT') as TDAT, // Date     Attribute
(FileLongName, 'TDLY') as TDLY, // Playlist delay

     Attribute(FileLongName, 'TENC') as TENC, // Encoded by     Attribute(FileLongName,
'TEXT') as TEXT, // Lyricist/Text writer     Attribute(FileLongName, 'TFLT') as TFLT, // File
type     Attribute(FileLongName, 'TIME') as TIME, // Time     Attribute(FileLongName, 'TIT1')
as TIT1, // Content group description     Attribute(FileLongName, 'TIT2') as TIT2, //
Title/songname/content description     Attribute(FileLongName, 'TIT3') as TIT3, //
Subtitle/Description refinement     Attribute(FileLongName, 'TKEY') as TKEY, // Initial key
 Attribute(FileLongName, 'TLAN') as TLAN, // Language(s)     Attribute(FileLongName, 'TLEN')
as TLEN, // Length     Attribute(FileLongName, 'TMED') as TMED, // Media type

     Attribute(FileLongName, 'TOAL') as TOAL, // Original album/movie/show title     Attribute
(FileLongName, 'TOFN') as TOFN, // Original filename     Attribute(FileLongName, 'TOLY') as
TOLY, // Original lyricist(s)/text writer(s)     Attribute(FileLongName, 'TOPE') as TOPE, //
Original artist(s)/performer(s)     Attribute(FileLongName, 'TORY') as TORY, // Original
release year     Attribute(FileLongName, 'TOWN') as TOWN, // File owner/licensee     Attribute
(FileLongName, 'TPE1') as TPE1, // Lead performer(s)/Soloist(s)     Attribute(FileLongName,
'TPE2') as TPE2, // Band/orchestra/accompaniment

     Attribute(FileLongName, 'TPE3') as TPE3, // Conductor/performer refinement     Attribute
(FileLongName, 'TPE4') as TPE4, // Interpreted, remixed, or otherwise modified by
Attribute(FileLongName, 'TPOS') as TPOS, // Part of a set     Attribute(FileLongName, 'TPUB')
as TPUB, // Publisher     Attribute(FileLongName, 'TRCK') as TRCK, // Track number/Position in
set     Attribute(FileLongName, 'TRDA') as TRDA, // Recording dates     Attribute
(FileLongName, 'TRSN') as TRSN, // Internet radio station name     Attribute(FileLongName,
'TRSO') as TRSO, // Internet radio station owner

     Attribute(FileLongName, 'TSIZ') as TSIZ, // Size     Attribute(FileLongName, 'TSRC') as
TSRC, // ISRC (international standard recording code)     Attribute(FileLongName, 'TSSE') as
TSSE, // Software/Hardware and settings used for encoding     Attribute(FileLongName, 'TYER')
as TYER, // Year     Attribute(FileLongName, 'TXXX') as TXXX, // User defined text information
frame     Attribute(FileLongName, 'UFID') as UFID, // Unique file identifier     Attribute
(FileLongName, 'USER') as USER, // Terms of use     Attribute(FileLongName, 'USLT') as USLT,
// Unsychronized lyric/text transcription     Attribute(FileLongName, 'WCOM') as WCOM, //
Commercial information     Attribute(FileLongName, 'WCOP') as WCOP, // Copyright/Legal
information

     Attribute(FileLongName, 'WOAF') as WOAF, // Official audio file webpage     Attribute
(FileLongName, 'WOAR') as WOAR, // Official artist/performer webpage     Attribute
(FileLongName, 'WOAS') as WOAS, // Official audio source webpage     Attribute(FileLongName,
'WORS') as WORS, // Official internet radio station homepage     Attribute(FileLongName,
'WPAY') as WPAY, // Payment     Attribute(FileLongName, 'WPUB') as WPUB, // Publishers
```

```
official webpage     Attribute(FileLongName, 'WXXX') as WXXX; // User defined URL link frame
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels); Next vFoundFile Next vExt
```

Example 2: JPEG

此脚本从文件夹 *MyPictures* 中的 JPG 文件读取所有可能的 EXIF 元标签。

```
// Script to read Jpeg Exif meta tags for each vExt in 'jpg', 'jpeg', 'jpe', 'jfif', 'jif',
'jfi' for each vFoundFile in filelist( GetFolderPath('MyPictures') & '\*.'& vExt )

 FileList: LOAD FileLongName,     subfield(FileLongName,'\',-1) as FileShortName,     num
(FileSize(FileLongName),'# ### ### ###',',',' ') as FileSize,     FileTime(FileLongName) as
FileTime,      // ************   Exif Main (IFD0) Attributes   ************     Attribute
(FileLongName, 'ImageWidth') as ImageWidth,     Attribute(FileLongName, 'ImageLength') as
ImageLength,     Attribute(FileLongName, 'BitsPerSample') as BitsPerSample,     Attribute
(FileLongName, 'Compression') as Compression,

    //  examples: 1=uncompressed, 2=CCITT, 3=CCITT 3, 4=CCITT 4,

    //5=LZW, 6=JPEG (old style), 7=JPEG, 8=Deflate, 32773=PackBits RLE,      Attribute
(FileLongName, 'PhotometricInterpretation') as PhotometricInterpretation,

    //  examples: 0=WhiteIsZero, 1=BlackIsZero, 2=RGB, 3=Palette, 5=CMYK, 6=YCbCr,
Attribute(FileLongName, 'ImageDescription') as ImageDescription,     Attribute(FileLongName,
'Make') as Make,     Attribute(FileLongName, 'Model') as Model,     Attribute(FileLongName,
'StripOffsets') as StripOffsets,     Attribute(FileLongName, 'Orientation') as Orientation,

    //  examples: 1=TopLeft, 2=TopRight, 3=BottomRight, 4=BottomLeft,

    // 5=LeftTop, 6=RightTop, 7=RightBottom, 8=LeftBottom,      Attribute(FileLongName,
'SamplesPerPixel') as SamplesPerPixel,     Attribute(FileLongName, 'RowsPerStrip') as
RowsPerStrip,     Attribute(FileLongName, 'StripByteCounts') as StripByteCounts,     Attribute
(FileLongName, 'XResolution') as XResolution,     Attribute(FileLongName, 'YResolution') as
YResolution,     Attribute(FileLongName, 'PlanarConfiguration') as PlanarConfiguration,

    //  examples: 1=chunky format, 2=planar format,      Attribute(FileLongName,
'ResolutionUnit') as ResolutionUnit,

    //  examples: 1=none, 2=inches, 3=centimeters,      Attribute(FileLongName,
'TransferFunction') as TransferFunction,     Attribute(FileLongName, 'Software') as Software,
   Attribute(FileLongName, 'DateTime') as DateTime,     Attribute(FileLongName, 'Artist') as
Artist,     Attribute(FileLongName, 'HostComputer') as HostComputer,     Attribute
(FileLongName, 'WhitePoint') as WhitePoint,     Attribute(FileLongName,
'PrimaryChromaticities') as PrimaryChromaticities,     Attribute(FileLongName,
'YCbCrCoefficients') as YCbCrCoefficients,     Attribute(FileLongName, 'YCbCrSubSampling') as
YCbCrSubSampling,     Attribute(FileLongName, 'YCbCrPositioning') as YCbCrPositioning,

    //  examples: 1=centered, 2=co-sited,      Attribute(FileLongName, 'ReferenceBlackWhite')
as ReferenceBlackWhite,     Attribute(FileLongName, 'Rating') as Rating,     Attribute
(FileLongName, 'RatingPercent') as RatingPercent,     Attribute(FileLongName,
'ThumbnailFormat') as ThumbnailFormat,

    //  examples: 0=Raw Rgb, 1=Jpeg,      Attribute(FileLongName, 'Copyright') as Copyright,
  Attribute(FileLongName, 'ExposureTime') as ExposureTime,     Attribute(FileLongName,
'FNumber') as FNumber,     Attribute(FileLongName, 'ExposureProgram') as ExposureProgram,
```

```
    //  examples: 0=Not defined, 1=Manual, 2=Normal program, 3=Aperture priority, 4=Shutter
priority,

    // 5=Creative program, 6=Action program, 7=Portrait mode, 8=Landscape mode, 9=Bulb,
Attribute(FileLongName, 'ISOSpeedRatings') as ISOSpeedRatings,     Attribute(FileLongName,
'TimeZoneOffset') as TimeZoneOffset,     Attribute(FileLongName, 'SensitivityType') as
SensitivityType,

    //  examples: 0=Unknown, 1=Standard output sensitivity (SOS), 2=Recommended exposure index
(REI),

    // 3=ISO speed, 4=Standard output sensitivity (SOS) and Recommended exposure index (REI),

    //5=Standard output sensitivity (SOS) and ISO Speed, 6=Recommended exposure index (REI)
and ISO Speed,

    // 7=Standard output sensitivity (SOS) and Recommended exposure index (REI) and ISO speed,
     Attribute(FileLongName, 'ExifVersion') as ExifVersion,     Attribute(FileLongName,
'DateTimeOriginal') as DateTimeOriginal,     Attribute(FileLongName, 'DateTimeDigitized') as
DateTimeDigitized,     Attribute(FileLongName, 'ComponentsConfiguration') as
ComponentsConfiguration,

    //  examples: 1=Y, 2=Cb, 3=Cr, 4=R, 5=G, 6=B,      Attribute(FileLongName,
'CompressedBitsPerPixel') as CompressedBitsPerPixel,     Attribute(FileLongName,
'ShutterSpeedValue') as ShutterSpeedValue,     Attribute(FileLongName, 'ApertureValue') as
ApertureValue,     Attribute(FileLongName, 'BrightnessValue') as BrightnessValue, //
examples: -1=Unknown,      Attribute(FileLongName, 'ExposureBiasValue') as ExposureBiasValue,
   Attribute(FileLongName, 'MaxApertureValue') as MaxApertureValue,     Attribute
(FileLongName, 'SubjectDistance') as SubjectDistance,

    //  examples: 0=Unknown, -1=Infinity,      Attribute(FileLongName, 'MeteringMode') as
MeteringMode,

    //  examples: 0=Unknown, 1=Average, 2=CenterWeightedAverage, 3=Spot,

    // 4=MultiSpot, 5=Pattern, 6=Partial, 255=Other,      Attribute(FileLongName,
'LightSource') as LightSource,

    //  examples: 0=Unknown, 1=Daylight, 2=Fluorescent, 3=Tungsten, 4=Flash, 9=Fine weather,

    // 10=Cloudy weather, 11=Shade, 12=Daylight fluorescent,

    // 13=Day white fluorescent, 14=Cool white fluorescent,

    // 15=White fluorescent, 17=Standard light A, 18=Standard light B, 19=Standard light C,

    // 20=D55, 21=D65, 22=D75, 23=D50, 24=ISO studio tungsten, 255=other light source,
Attribute(FileLongName, 'Flash') as Flash,     Attribute(FileLongName, 'FocalLength') as
FocalLength,     Attribute(FileLongName, 'SubjectArea') as SubjectArea,     Attribute
(FileLongName, 'MakerNote') as MakerNote,     Attribute(FileLongName, 'UserComment') as
UserComment,     Attribute(FileLongName, 'SubSecTime') as SubSecTime,

     Attribute(FileLongName, 'SubsecTimeOriginal') as SubsecTimeOriginal,     Attribute
(FileLongName, 'SubsecTimeDigitized') as SubsecTimeDigitized,     Attribute(FileLongName,
'XPTitle') as XPTitle,     Attribute(FileLongName, 'XPComment') as XPComment,
```

```
    Attribute(FileLongName, 'XPAuthor') as XPAuthor,    Attribute(FileLongName,
'XPKeywords') as XPKeywords,    Attribute(FileLongName, 'XPSubject') as XPSubject,
Attribute(FileLongName, 'FlashpixVersion') as FlashpixVersion,    Attribute(FileLongName,
'ColorSpace') as ColorSpace, //  examples: 1=sRGB, 65535=Uncalibrated,    Attribute
(FileLongName, 'PixelXDimension') as PixelXDimension,    Attribute(FileLongName,
'PixelYDimension') as PixelYDimension,    Attribute(FileLongName, 'RelatedSoundFile') as
RelatedSoundFile,

    Attribute(FileLongName, 'FocalPlaneXResolution') as FocalPlaneXResolution,    Attribute
(FileLongName, 'FocalPlaneYResolution') as FocalPlaneYResolution,    Attribute(FileLongName,
'FocalPlaneResolutionUnit') as FocalPlaneResolutionUnit,

    //  examples: 1=None, 2=Inch, 3=Centimeter,    Attribute(FileLongName, 'ExposureIndex')
as ExposureIndex,    Attribute(FileLongName, 'SensingMethod') as SensingMethod,

    //  examples: 1=Not defined, 2=One-chip color area sensor, 3=Two-chip color area sensor,

    // 4=Three-chip color area sensor, 5=Color sequential area sensor,

    // 7=Trilinear sensor, 8=Color sequential linear sensor,    Attribute(FileLongName,
'FileSource') as FileSource,

    //  examples: 0=Other, 1=Scanner of transparent type,

    // 2=Scanner of reflex type, 3=Digital still camera,    Attribute(FileLongName,
'SceneType') as SceneType,

    //  examples: 1=A directly photographed image,    Attribute(FileLongName, 'CFAPattern')
as CFAPattern,    Attribute(FileLongName, 'CustomRendered') as CustomRendered,

    //  examples: 0=Normal process, 1=Custom process,    Attribute(FileLongName,
'ExposureMode') as ExposureMode,

    //  examples: 0=Auto exposure, 1=Manual exposure, 2=Auto bracket,    Attribute
(FileLongName, 'WhiteBalance') as WhiteBalance,

    //  examples: 0=Auto white balance, 1=Manual white balance,    Attribute(FileLongName,
'DigitalZoomRatio') as DigitalZoomRatio,    Attribute(FileLongName, 'FocalLengthIn35mmFilm')
as FocalLengthIn35mmFilm,    Attribute(FileLongName, 'SceneCaptureType') as SceneCaptureType,

    //  examples: 0=Standard, 1=Landscape, 2=Portrait, 3=Night scene,    Attribute
(FileLongName, 'GainControl') as GainControl,

    //  examples: 0=None, 1=Low gain up, 2=High gain up, 3=Low gain down, 4=High gain down,
  Attribute(FileLongName, 'Contrast') as Contrast,

    //  examples: 0=Normal, 1=Soft, 2=Hard,    Attribute(FileLongName, 'Saturation') as
Saturation,

    //  examples: 0=Normal, 1=Low saturation, 2=High saturation,    Attribute(FileLongName,
'Sharpness') as Sharpness,

    //  examples: 0=Normal, 1=Soft, 2=Hard,    Attribute(FileLongName,
'SubjectDistanceRange') as SubjectDistanceRange,
```

```
    //  examples: 0=Unknown, 1=Macro, 2=Close view, 3=Distant view,      Attribute
(FileLongName, 'ImageUniqueID') as ImageUniqueID,     Attribute(FileLongName,
'BodySerialNumber') as BodySerialNumber,     Attribute(FileLongName, 'CMNT_GAMMA') as CMNT_
GAMMA,     Attribute(FileLongName, 'PrintImageMatching') as PrintImageMatching,     Attribute
(FileLongName, 'OffsetSchema') as OffsetSchema,

     // ************   Interoperability Attributes   ************     Attribute(FileLongName,
'InteroperabilityIndex') as InteroperabilityIndex,     Attribute(FileLongName,
'InteroperabilityVersion') as InteroperabilityVersion,     Attribute(FileLongName,
'InteroperabilityRelatedImageFileFormat') as InteroperabilityRelatedImageFileFormat,
Attribute(FileLongName, 'InteroperabilityRelatedImageWidth') as
InteroperabilityRelatedImageWidth,     Attribute(FileLongName,
'InteroperabilityRelatedImageLength') as InteroperabilityRelatedImageLength,     Attribute
(FileLongName, 'InteroperabilityColorSpace') as InteroperabilityColorSpace,

    //  examples: 1=sRGB, 65535=Uncalibrated,      Attribute(FileLongName,
'InteroperabilityPrintImageMatching') as InteroperabilityPrintImageMatching,     //
************   GPS Attributes   ************     Attribute(FileLongName, 'GPSVersionID') as
GPSVersionID,     Attribute(FileLongName, 'GPSLatitudeRef') as GPSLatitudeRef,     Attribute
(FileLongName, 'GPSLatitude') as GPSLatitude,     Attribute(FileLongName, 'GPSLongitudeRef')
as GPSLongitudeRef,     Attribute(FileLongName, 'GPSLongitude') as GPSLongitude,     Attribute
(FileLongName, 'GPSAltitudeRef') as GPSAltitudeRef,

    //  examples: 0=Above sea level, 1=Below sea level,      Attribute(FileLongName,
'GPSAltitude') as GPSAltitude,     Attribute(FileLongName, 'GPSTimeStamp') as GPSTimeStamp,
 Attribute(FileLongName, 'GPSSatellites') as GPSSatellites,     Attribute(FileLongName,
'GPSStatus') as GPSStatus,     Attribute(FileLongName, 'GPSMeasureMode') as GPSMeasureMode,
 Attribute(FileLongName, 'GPSDOP') as GPSDOP,     Attribute(FileLongName, 'GPSSpeedRef') as
GPSSpeedRef,

     Attribute(FileLongName, 'GPSSpeed') as GPSSpeed,     Attribute(FileLongName,
'GPSTrackRef') as GPSTrackRef,     Attribute(FileLongName, 'GPSTrack') as GPSTrack,
Attribute(FileLongName, 'GPSImgDirectionRef') as GPSImgDirectionRef,     Attribute
(FileLongName, 'GPSImgDirection') as GPSImgDirection,     Attribute(FileLongName,
'GPSMapDatum') as GPSMapDatum,     Attribute(FileLongName, 'GPSDestLatitudeRef') as
GPSDestLatitudeRef,

     Attribute(FileLongName, 'GPSDestLatitude') as GPSDestLatitude,     Attribute
(FileLongName, 'GPSDestLongitudeRef') as GPSDestLongitudeRef,     Attribute(FileLongName,
'GPSDestLongitude') as GPSDestLongitude,     Attribute(FileLongName, 'GPSDestBearingRef') as
GPSDestBearingRef,     Attribute(FileLongName, 'GPSDestBearing') as GPSDestBearing,
Attribute(FileLongName, 'GPSDestDistanceRef') as GPSDestDistanceRef,

     Attribute(FileLongName, 'GPSDestDistance') as GPSDestDistance,     Attribute
(FileLongName, 'GPSProcessingMethod') as GPSProcessingMethod,     Attribute(FileLongName,
'GPSAreaInformation') as GPSAreaInformation,     Attribute(FileLongName, 'GPSDateStamp') as
GPSDateStamp,     Attribute(FileLongName, 'GPSDifferential') as GPSDifferential;

    //  examples: 0=No correction, 1=Differential correction,  LOAD @1:n as FileLongName
Inline "$(vFoundFile)" (fix, no labels); Next vFoundFile Next vExt
```

Example 3: Windows 媒体文件

此脚本读取文件夹 *MyMusic* 中所有可能的 WMA/WMV ASF 元标签。

```
/ Script to read WMA/WMV ASF meta tags for each vExt in 'asf', 'wma', 'wmv' for each
vFoundFile in filelist( GetFolderPath('MyMusic') & '\*.'& vExt )

 FileList: LOAD FileLongName,     subfield(FileLongName,'\',-1) as FileShortName,     num
(FileSize(FileLongName),'# ### ### ###',',',' ') as FileSize,     FileTime(FileLongName) as
FileTime,      Attribute(FileLongName, 'Title') as Title,     Attribute(FileLongName,
'Author') as Author,     Attribute(FileLongName, 'Copyright') as Copyright,     Attribute
(FileLongName, 'Description') as Description,

     Attribute(FileLongName, 'Rating') as Rating,     Attribute(FileLongName, 'PlayDuration')
as PlayDuration,     Attribute(FileLongName, 'MaximumBitrate') as MaximumBitrate,
Attribute(FileLongName, 'WMFSDKVersion') as WMFSDKVersion,     Attribute(FileLongName,
'WMFSDKNeeded') as WMFSDKNeeded,     Attribute(FileLongName, 'IsVBR') as IsVBR,     Attribute
(FileLongName, 'ASFLeakyBucketPairs') as ASFLeakyBucketPairs,

     Attribute(FileLongName, 'PeakValue') as PeakValue,     Attribute(FileLongName,
'AverageLevel') as AverageLevel; LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no
labels); Next vFoundFile Next vExt
```

Example 4: PNG

此脚本读取文件夹 *MyPictures* 中所有可能的 PNG 元标签。

```
// Script to read PNG meta tags for each vExt in 'png' for each vFoundFile in filelist(
GetFolderPath('MyPictures') & '\*.'& vExt )

 FileList: LOAD FileLongName,     subfield(FileLongName,'\',-1) as FileShortName,     num
(FileSize(FileLongName),'# ### ### ###',',',' ') as FileSize,     FileTime(FileLongName) as
FileTime,      Attribute(FileLongName, 'Comment') as Comment,

     Attribute(FileLongName, 'Creation Time') as Creation_Time,     Attribute(FileLongName,
'Source') as Source,     Attribute(FileLongName, 'Title') as Title,     Attribute
(FileLongName, 'Software') as Software,     Attribute(FileLongName, 'Author') as Author,
Attribute(FileLongName, 'Description') as Description,

     Attribute(FileLongName, 'Copyright') as Copyright; LOAD @1:n as FileLongName Inline
"$(vFoundFile)" (fix, no labels); Next vFoundFile Next vExt
```

ConnectString

ConnectString() 函数用于返回 ODBC 或 OLE DB 连接的活动数据连接的名称。此函数用于返回在没有执行 **connect** 语句时或在执行 **disconnect** 语句后返回空字符串。

语法：

```
ConnectString()
```

示例和结果：

脚本示例

示例	结果
LIB CONNECT TO 'Tutorial ODBC'; ConnectString: Load ConnectString() as ConnectString AutoGenerate 1;	在字段 ConnectString 中返回“Tutorial ODBC”。 此示例假设您拥有名为 Tutorial ODBC 的可用数据连接。

FileBaseName

FileBaseName 函数返回一个字符串，其中包含当前正在读取的表格文件的名称，没有路径或扩展名。

语法：

```
FileBaseName()
```

示例和结果：

脚本示例

示例	结果
LOAD *, filebasename( ) as X from C:\UserFiles\abc.txt	将在每个阅读记录中的 X 字段中返回“abc”。

FileDir

FileDir 函数用于返回一个包含至当前阅读表格文件目录的路径。

语法：

```
FileDir()
```

此函数仅在标准模式下支持文件夹数据连接。

示例和结果：

脚本示例

示例	结果
Load *, filedir( ) as X from C:\UserFiles\abc.txt	将在每个阅读记录中的 X 字段中返回“C:\UserFiles”。

FileExtension

FileExtension 函数用于返回一个包含至当前阅读表格文件扩展名的字符串。

语法：

```
FileExtension()
```

示例和结果：

脚本示例

示例	结果
LOAD *, FileExtension( ) as X from C:\UserFiles\abc.txt	将在每个阅读记录的 X 字段中返回 txt。

FileName

FileName 函数返回一个字符串，其中包含当前正在读取的表格文件的名称，没有路径或扩展名。

语法：

FileName()

示例和结果：

脚本示例

示例	结果
LOAD *, FileName( ) as X from C:\UserFiles\abc.txt	将在每个阅读记录中的 X 字段中返回 'abc.txt'。

FilePath

FilePath 函数用于返回一个包含至当前阅读表格文件的完整路径的字符串。

语法：

FilePath()

此函数仅在标准模式下支持文件夹数据连接。

示例和结果：

脚本示例

示例	结果
Load *, FilePath( ) as X from C:\UserFiles\abc.txt	将在每个阅读记录中的 X 字段中返回 'C:\UserFiles\abc.txt'。

FileSize

FileSize 函数用于返回一个包含文件 filename 字节大小的整数，或如果未指定 filename，则返回一个包含当前阅读的表格文件字节大小的整数。

语法：

FileSize([filename]**)**

参数：

参数

参数	说明
filename	文件名(如有必要)包括路径，作为文件夹或 Web 文件的数据连接。如果您没有指定文件名，将使用当前正在读取的表格文件。 **示例：** *'lib://Table Files/'* 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** *c:\data* • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** *data* • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：** *http://www.qlik.com*

示例和结果：

脚本示例

示例	结果
`LOAD *, FileSize( ) as X from abc.txt;`	将以整数形式在每个阅读记录的 X 字段中返回指定文件 (abc.txt) 的大小。
`FileSize( 'lib://DataFiles/xyz.xls' )`	将返回文件 xyz.xls 的大小。

FileTime

FileTime 函数用于返回文件 filename 的上一次修改日期和时间的时间戳。如果未指定 filename，则此函数将参考当前阅读的表格文件。

语法：

```
FileTime([ filename ])
```

参数：

参数

参数	说明
filename	文件名(如有必要)包括路径，作为文件夹或 Web 文件的数据连接。 **示例：** *'lib://Table Files/'* 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** *c:\data* • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** *data* • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：** *http://www.qlik.com*

示例和结果：

脚本示例

示例	结果
`LOAD *, FileTime( ) as X from abc.txt;`	将以整数形式在每个阅读记录的 X 字段中以时间戳形式返回文件 (abc.txt) 上一次修改的日期和时间。
FileTime('xyz.xls')	将返回文件 xyz.xls 上一次修改的时间戳。

GetFolderPath

GetFolderPath 函数用于返回 Microsoft Windows *SHGetFolderPath* 函数的值。此函数用于输入 Microsoft Windows 文件夹的名称，并返回该文件夹的完整路径。

在标准模式下不支持此函数。

语法：

```
GetFolderPath(foldername)
```

参数：

参数

参数	说明
foldername	Microsoft Windows 文件夹的名称。 文件夹名称不得包含任何空格。应从文件夹名称移除在 Windows Explorer 中看到的文件夹名称所包含的任何空格。 示例： *MyMusic* *MyDocuments*

示例和结果：

本示例的目的是获取以下 Microsoft Windows 文件夹的路径：*MyMusic*、*MyPictures* 和 *Windows*。将示例脚本添加到应用程序并重新加载。

```
LOAD   GetFolderPath('MyMusic') as MyMusic,  GetFolderPath('MyPictures') as MyPictures,
GetFolderPath('Windows') as Windows AutoGenerate 1;
```

重新加载应用程序后，已将字段 *MyMusic*、*MyPictures* 和 *Windows* 添加到数据模型。每个字段均包含在输入中定义的文件夹路径。例如：

- *C:\Users\smu\Music* for the folder *MyMusic*
- *C:\Users\smu\Pictures* for the folder *MyPictures*
- *C:\Windows* for the folder *Windows*

QvdCreateTime

此脚本函数用于返回 QVD 文件的 XML-标题时间戳(如果有)，否则返回 NULL 值。

语法：

```
QvdCreateTime(filename)
```

参数：

参数

参数	说明
filename	QVD 文件名(如有必要)包括路径，作为文件夹或 Web 数据连接。 **示例：** ***'lib://Table Files/'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** ***c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** ***data*** • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：** ***http://www.qlik.com***

示例：

```
QvdCreateTime('MyFile.qvd')

QvdCreateTime('C:\MyDir\MyFile.qvd')

QvdCreateTime('lib://DataFiles/MyFile.qvd')
```

QvdFieldName

此脚本函数返回 QVD 文件中的字段编号 **fieldno**。如果字段不存在，则返回 NULL。

语法：

```
QvdFieldName(filename , fieldno)
```

参数：

参数

参数	说明
filename	QVD 文件名(如有必要)包括路径，作为文件夹或 Web 数据连接。 **示例：** ***'lib://Table Files/'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** ***c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** ***data*** • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：** ***http://www.qlik.com***
fieldno	QVD 文件中所含的表格内的字段编号。

示例：

```
QvdFieldName ('MyFile.qvd', 5)

QvdFieldName ('C:\MyDir\MyFile.qvd', 5)

QvdFieldName ('lib://DataFiles/MyFile.qvd', 5)
```

所有三个示例返回包含在 QVD 文件中的表格的第五个字段的名称。

QvdNoOfFields

此脚本函数用于返回 QVD 文件中的字段数。

语法：

```
QvdNoOfFields(filename)
```

参数：

参数

参数	说明
filename	QVD 文件名(如有必要)包括路径，作为文件夹或 Web 数据连接。 **示例：*'lib://Table Files/'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：*c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：*data*** • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：*http://www.qlik.com***

示例：

```
QvdNoOfFields ('MyFile.qvd')

QvdNoOfFields ('C:\MyDir\MyFile.qvd')

QvdNoOfFields ('lib://DataFiles/MyFile.qvd')
```

QvdNoOfRecords

示例：此脚本函数用于返回 QVD 文件中的当前记录数。

语法：

```
QvdNoOfRecords(filename)
```

参数：

参数

参数	说明
filename	QVD 文件名(如有必要)包括路径，作为文件夹或 Web 数据连接。 **示例：*'lib://Table Files/'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：*c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：*data*** • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：*http://www.qlik.com***

示例：

```
QvdNoOfRecords ('MyFile.qvd')

QvdNoOfRecords ('C:\MyDir\MyFile.qvd')

QvdNoOfRecords ('lib://DataFiles/MyFile.qvd')
```

QvdTableName

此脚本函数用于返回存储在 QVD 文件中的表格名称。

语法：

```
QvdTableName(filename)
```

参数：

参数

参数	说明
filename	QVD 文件名(如有必要)包括路径，作为文件夹或 Web 数据连接。 **示例：** ***'lib://Table Files/'*** 在传统脚本模式下，同时支持以下路径格式： • 绝对 **示例：** ***c:\data*** • 相对 Qlik Sense 应用程序工作目录的相对路径。 **示例：** ***data*** • URL 地址(HTTP 或 FTP)，指向一个互联网或内联网的位置。 **示例：** ***http://www.qlik.com***

示例：

```
QvdTableName ('MyFile.qvd')
QvdTableName ('C:\MyDir\MyFile.qvd')
QvdTableName ('lib://data\MyFile.qvd')
```

5.11 财务函数

财务函数可用于数据加载脚本和图表表达式中计算付款和利率。
所有自变量，现金支出用负数表示。现金收款由正数表示。
此处列出用于财务函数的自变量(除了以 **range-** 开头的自变量)。

> *对于所有财务函数来说，在指定* ***rate*** *和* ***nper*** *的单位时保持一致是至关重要的。如果在一个年利率为 6% 的五年期贷款的基础上进行每月付款，则应针对* ***rate*** *使用 0.005 (6%/12)，针对* ***nper*** *使用 60 (5*12)。如果年付款在相同的贷款基础上作出，应使用 6% 作为* ***rate****，5 作为* ***nper****。*

财务函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

FV

此函数用于返回基于周期性，不变付款额以及简单年利率的一项投资的未来值。

```
FV (rate, nper, pmt [ ,pv [ , type ] ])
```

nPer

此函数用于返回基于周期性，不变付款额以及不变利率的一项投资的周期数。

```
nPer (rate, pmt, pv [ ,fv [ , type ] ])
```

Pmt

此函数用于返回基于周期性，不变付款额以及不变利率的一项贷款的付款额。它无法改变年金的周期。付款以负数表示，例如 -20。

```
Pmt (rate, nper, pv [ ,fv [ , type ] ] )
```

PV

此函数用于返回一项投资的现在价值。

```
PV (rate, nper, pmt [ ,fv [ , type ] ])
```

Rate

此函数用于按年返回每周期利率。结果拥有一个默认数字形式 **Fix** 两位小数位及 %。

```
Rate (nper, pmt , pv [ ,fv [ , type ] ])
```

BlackAndSchole

Black and Scholes 模型金融市场衍生产品的数学模型。该公式用于计算期权的理论值。在 Qlik Sense 中，**BlackAndSchole** 函数根据 Black and Scholes(欧式期权公式) 返回值。

```
BlackAndSchole(strike , time_left , underlying_price , vol , risk_free_rate , type)
```

返回数据类型： 数字

参数：

参数

参数	说明
strike	股价的未来购买价。
time_left	时间周期剩余数。
underlying_price	股价现值。
vol	(股价) 波动表示为每个时间周期的小数格式的百分比。
risk_free_rate	无风险利率表示为每个时间周期的小数格式的百分比。
call_or_put	期权的类型： 'c'，指认购期权的'call'或任何非零数值 'p'，指看跌期权的'put' 或 0。

限制：

strike、time_left 和 underlying_price 的值必须 >0。

vol 和 risk_free_rate 的值必须 <0 或 >0。

示例和结果：

脚本示例

示例	结果
`BlackAndSchole(130, 4, 68.5, 0.4, 0.04, 'call')` 这用于计算期权的理论价格，如果以每股 130 的价格购买 4 年，则现在每股增值 68.5。该公式使用的每年股价波动为 0.4 (40%)，无风险利率为 0.04 (4%)。	返回 11.245

FV

此函数用于返回基于周期性，不变付款额以及简单年利率的一项投资的未来值。

语法：

```
FV(rate, nper, pmt [ ,pv [ , type ] ])
```

返回数据类型：数字。结果默认采用货币数字格式。.

参数：

参数

参数	说明
rate	每周期的利率。
nper	年金中付款周期的总数。
pmt	每个周期的付款。它无法改变年金的周期。付款以负数表示，例如 -20。
pv	一系列未来支付的现在价值的当前值或一次付款总额。如果省略了 **pv**，假设为 0(零) 。
type	如果付款应在期末支付，则应为 0，如果付款应在期初支付，则应为 1。如果省略了 **type**，假设为 0。

示例和结果：

脚本示例

示例	结果
您将为一个新的家电分期付款 36 个月，每月 20 美元。利率为每年 6%。账单每月末出据。投资款的总价值是多少，什么时候支付最后一次账单？ `FV(0.005,36,-20)`	返回 $786.72

nPer

此函数用于返回基于周期性，不变付款额以及不变利率的一项投资的周期数。

语法：

```
nPer(rate, pmt, pv [ ,fv [ , type ] ])
```

返回数据类型：数字

参数：

参数

参数	说明
rate	每周期的利率。
nper	年金中付款周期的总数。
pmt	每个周期的付款。它无法改变年金的周期。付款以负数表示，例如 -20。
pv	一系列未来支付的现在价值的当前值或一次付款总额。如果省略了 **pv**，假设为 0(零)。
fv	您希望在完成上一次付款后获取的未来值或现金结余。如果省略了 **fv**，假设为 0。
type	如果付款应在期末支付，则应为 0，如果付款应在期初支付，则应为 1。如果省略了 **type**，假设为 0。

示例和结果：

脚本示例

示例	结果
您想销售一个家电，每月分期付款 20 美元。利率为每年 6%。账单每月末出据。如果在最后一次款项已付清后收到的款项值应该等于 800 美元需要多少个付款周期? `nPer(0.005,-20,0,800)`	返回 36.56

Pmt

此函数用于返回基于周期性，不变付款额以及不变利率的一项贷款的付款额。它无法改变年金的周期。付款以负数表示，例如 -20。

```
Pmt(rate, nper, pv [ ,fv [ , type ] ] )
```

返回数据类型：数字。结果默认采用货币数字格式。.

要想算出贷款期间的付款总额，将返回的 **pmt** 值乘以 **nper**。

参数：

参数

参数	说明
rate	每周期的利率。
nper	年金中付款周期的总数。
pv	一系列未来支付的现在价值的当前值或一次付款总额。如果省略了 **pv**，假设为 0(零)。
fv	您希望在完成上一次付款后获取的未来值或现金结余。如果省略了 **fv**，假设为 0。
type	如果付款应在期末支付，则应为 0，如果付款应在期初支付，则应为 1。如果省略了 **type**，假设为 0。

示例和结果：

脚本示例

示例	结果
以下公式返回 8 个月内必须付清的年税率 10% 的一项 20000 美元贷款的每月付款额： `Pmt(0.1/12,8,20000)`	返回 -$2,594.66
对于相同的贷款，如何付款在周期的开始到期，则支付款为： `Pmt(0.1/12,8,20000,0,1)`	返回 -$2,573.21

PV

此函数用于返回一项投资的现在价值。

```
PV(rate, nper, pmt [ ,fv [ , type ] ])
```

返回数据类型： 数字。结果默认采用货币数字格式。.

现在价值是一系列未来付款现在价值的总额。例如，当借钱时，贷款额对于贷款人来说就是现值。

参数：

参数

参数	说明
rate	每周期的利率。
nper	年金中付款周期的总数。
pmt	每个周期的付款。它无法改变年金的周期。付款以负数表示，例如 -20。
fv	您希望在完成上一次付款后获取的未来值或现金结余。如果省略了 **fv**，假设为 0。
type	如果付款应在期末支付，则应为 0，如果付款应在期初支付，则应为 1。如果省略了 **type**，假设为 0。

示例和结果：

脚本示例

示例	结果
如果利率为 7%，在五年时间期限内每月结束时向您支付 100 美元的债务的现值是多少？ `PV(0.07/12,12*5,-100,0,0)`	返回 $5,050.20

Rate

此函数用于按年返回每周期利率。结果拥有一个默认数字形式 **Fix** 两位小数位及 %。

语法：

```
Rate(nper, pmt , pv [ ,fv [ , type ] ])
```

返回数据类型： 数字。

rate可循环计算，并且可以拥有零或更多解决方案。如果**rate**的连续结果不渐渐接近，将会返回一个 NULL 值。

参数：

参数

参数	说明
nper	年金中付款周期的总数。
pmt	每个周期的付款。它无法改变年金的周期。付款以负数表示，例如 -20。
pv	一系列未来支付的现在价值的当前值或一次付款总额。如果省略了 **pv**，假设为 0(零)。
fv	您希望在完成上一次付款后获取的未来值或现金结余。如果省略了 **fv**，假设为 0。
type	如果付款应在期末支付，则应为 0，如果付款应在期初支付，则应为 1。如果省略了 **type**，假设为 0。

示例和结果：

脚本示例

示例	结果
一个五年期的 10000 美金年金贷款每月支付 300 美元，利率是多少？ `Rate(60,-300,10000)`	返回 2.00%

5.12 格式函数

格式函数用于对输入数字字段或表达式强制使用显示格式，根据数据类型，您可以指定字符作为小数位分隔符、千分位分隔符等。

这些函数都返回包含字符串和数字值的对偶值，但可被视为执行一次从数字到字符串的转换。**Dual ()** 是一个特殊情况，但其他格式函数会获取输入表达式的数字值，然后生成一个表示该数字的字符串。

相比之下，解释函数则相反：它们获取字符串表达式并计算其数字值，从而指定生成数字的格式。

这些函数均可用于数据加载脚本和图表表达式。

所有数字表示形式都指定以小数点作为小数位分隔符。

格式函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

ApplyCodepage

ApplyCodepage() 应用不同的代码页字符集到表达式内所述的字段或文本。**codepage** 参数必须是数字格式。

```
ApplyCodepage (text, codepage)
```

Date

Date() 用于使用数据加载脚本的系统变量、操作系统或格式字符串(如果提供)中设置的格式，将表达式的格式设置为日期格式。

```
Date (number[, format])
```

Dual

Dual() 用于将数字和字符串组合为单个记录，以便此记录的数字呈现形式可用于排序和计算，同时字符串值可用于显示。

```
Dual (text, number)
```

Interval

Interval() 用于使用数据加载脚本的系统变量、操作系统或格式字符串(如果提供)中的格式，将数字的格式设置为时间间隔格式。

```
Interval (number[, format])
```

Money

Money() 用于使用数据加载脚本中设置的系统变量或操作系统(如果不提供格式字符串)中设置的格式，以及可选的小数位和千分位分隔符，将表达式的格式设置为数字形式的货币值格式。

```
Money (number[, format[, dec_sep [, thou_sep]]])
```

Num

Num() 格式化数字，即使用第二个参数中指定的格式将输入的数值转换为显示文本。如果省略第二个参数，它将使用数据加载脚本中设置的小数点和千位分隔符。自定义小数位和千分位分隔符的符号为可选参数。

```
Num (number[, format[, dec_sep [, thou_sep]]])
```

Time

Time() 用于使用数据加载脚本的系统变量或操作系统(如果不提供格式字符串)中设置的时间格式，将表达式的格式设置为时间值格式。

```
Time (number[, format])
```

Timestamp

TimeStamp() 用于使用数据加载脚本的系统变量或操作系统(如果不提供格式字符串)中设置的时间戳格式，将表达式的格式设置为日期和时间值格式。

```
Timestamp (number[, format])
```

另请参见：

解释函数 (page 586)

ApplyCodepage

ApplyCodepage() 应用不同的代码页字符集到表达式内所述的字段或文本。**codepage** 参数必须是数字格式。

> *虽然 ApplyCodepage 可用于图表表达式，但是它更常用作数据加载编辑器中的脚本函数。例如，当您加载可能使用超出您控制的不同字符集保存的文件时，您可以应用代表您所需字符集的代码页。*

语法：

```
ApplyCodepage(text, codepage)
```

返回数据类型： 字符串

参数：

参数

参数	说明
text	您要对其应用不同代码页的字段或文本，通过参数 **codepage** 指定。
codepage	代表要应用于 **text** 指定的字段或表达式的代码页的数字。

示例和结果：

脚本示例

示例	结果
LOAD ApplyCodepage(ROWX,1253) as GreekProduct, ApplyCodepage (ROWY, 1255) as HebrewProduct, ApplyCodepage (ROWZ, 65001) as EnglishProduct; SQL SELECT ROWX, ROWY, ROWZ From Products;	从 SQL 加载时，源可能混用不同字符集（来自 UTF-8 格式）：的西里尔语、希伯来语等。逐行加载时将需要这些，以对每行应用不同的代码页。 **codepage** 值 1253 代表 Windows 希腊语字符集、值 1255 代表希伯来语、值 65001 代表标准拉丁语 UTF-8 字符。

另请参见： *字符集 (page 100)*

Date

Date() 用于使用数据加载脚本的系统变量、操作系统或格式字符串（如果提供）中设置的格式，将表达式的格式设置为日期格式。

语法：

```
Date(number[, format])
```

返回数据类型： 双

参数：

参数

参数	说明
number	可以设置数字的格式。
format	描述结果字符串格式的字符串。如果不提供格式字符串，则使用在数据加载脚本或操作系统中的系统变量中设置的日期格式。

示例和结果：
以下示例假设采用以下默认设置：

- 日期设置 1：YY-MM-DD
- 日期设置 2：M/D/YY

示例：

```
Date( A )
```
其中 A=35648

结果表

结果	设置 1	设置 2
字符串:	97-08-06	8/6/97
数字:	35648	35648

示例:

```
Date( A, 'YY.MM.DD' )
```
其中 A=35648

结果表

结果	设置 1	设置 2
字符串:	97.08.06	97.08.06
数字:	35648	35648

示例:

```
Date( A, 'DD.MM.YYYY' )
```
其中 A=35648.375

结果表

结果	设置 1	设置 2
字符串:	06.08.1997	06.08.1997
数字:	35648.375	35648.375

示例:

```
Date( A, 'YY.MM.DD' )
```
其中 A=8/6/97

结果表

结果	设置 1	设置 2
字符串:	NULL(什么都没有)	97.08.06
数字:	NULL	35648

Dual

Dual() 用于将数字和字符串组合为单个记录,以便此记录的数字呈现形式可用于排序和计算,同时字符串值可用于显示。

语法:

```
Dual(text, number)
```

返回数据类型：双

参数：

参数

参数	说明
text	与数字参数组合使用的字符串值。
number	与字符串参数中的字符串组合使用的数字。

在 Qlik Sense 中，所有字段值都可能是双重值。这意味着字段值即可以是数值，也可以是文本值。例如，一个日期即可包含数值 40908，也可以包含文本呈现形式 '2011-12-31'。

当几个数据项读入到一个具有不同字符串呈现形式但具有同一有效的数字呈现形式的字段中时，所有数据项都将共享遇到的第一个字符串呈现形式。

*在其他数据被读入到有关字段中之前，**dual** 函数通常在脚本中使用，以便创建首字符串呈现形式，这将显示在筛选器窗格中。*

示例和结果：

脚本示例

示例	说明
在脚本中添加下例并运行。 `Load dual ( NameDay,NumDay ) as DayOfWeek inline [ NameDay,NumDay Monday,0 Tuesday,1 Wednesday,2 Thursday,3 Friday,4 Saturday,5 Sunday,6 ];`	字段 DayOfWeek 可用于可视化，例如，用作维度。在包含星期的表格中，每周的具体日期会自动按正确的数字顺序而不是字母顺序排序。
`Load Dual('Q' & Ceil(Month (Now())/3), Ceil(Month(Now ())/3)) as Quarter AutoGenerate 1;`	本例提供当前季度。如果在一年的第一个三个月中运行 **Now()** 函数，则将第一个三个月显示为 Q1，将第二个三个月显示为 Q2，以此类推。但是，在用于排序时，字段 Quarter 将按其数值顺序（1 到 4）排序。
`Dual('Q' & Ceil(Month (Date)/3), Ceil(Month (Date)/3)) as Quarter`	与之前的示例一样，使用文本值 'Q1' 到 'Q4' 创建字段 Quarter，并为其分配数值 1 到 4。为在脚本中使用此字段，必须加载 Date 的值。
`Dual(WeekYear(Date) & '-W' & Week(Date), WeekStart (Date)) as YearWeek`	本例将创建一个字段 YearWeek，该字段具有 '2012-W22' 形式的文本值，同时分配与一周第一天的日期数对应的数值，例如：41057. 为在脚本中使用此字段，必须加载 Date 的值。

Interval

Interval() 用于使用数据加载脚本的系统变量、操作系统或格式字符串(如果提供)中的格式,将数字的格式设置为时间间隔格式。

可将时间间隔格式设置为时间、天数或天数、小时数、分钟数、秒数和分秒数的组合。

语法:

```
Interval(number[, format])
```

返回数据类型: 双

参数:

参数

参数	说明
number	可以设置数字的格式。
format	说明如何设置结果间隔字符串格式的字符串。如果省略,则使用操作系统中设置的缩写日期格式、时间格式和小数位分隔符。

示例和结果:
以下示例假设采用以下默认设置:

- 日期格式设置 1: YY-MM-DD
- 日期格式设置 2: hh:mm:ss
- 数字小数位分隔符: 。

结果表

示例	字符串	数字
`Interval( A )` 其中 A=0.375	09:00:00	0.375
`Interval( A )` 其中 A=1.375	33:00:00	1.375
`Interval( A, 'D hh:mm' )` 其中 A=1.375	1 09:00	1.375
`Interval( A-B, 'D hh:mm' )` 其中 A=97-08-06 09:00:00 和 B=96-08-06 00:00:00	365 09:00	365.375

Money

Money() 用于使用数据加载脚本中设置的系统变量或操作系统(如果不提供格式字符串)中设置的格式,以及可选的小数位和千分位分隔符,将表达式的格式设置为数字形式的货币值格式。

语法：

```
Money(number[, format[, dec_sep[, thou_sep]]])
```

返回数据类型：双

参数：

参数

参数	说明
number	可以设置数字的格式。
format	说明如何设置结果货币字符串格式的字符串。
dec_sep	指定小数位数字分隔符的字符串。
thou_sep	指定千分位数字分隔符的字符串。

如果省略参数 2-4，则使用操作系统中设置的货币格式。

示例和结果：
以下示例假设采用以下默认设置：

- MoneyFormat setting 1:kr ##0,00, MoneyThousandSep' '
- MoneyFormat setting 2:$ #,##0.00, MoneyThousandSep','

示例：

```
Money( A )
```
其中 A=35648

结果表

结果	设置 1	设置 2
字符串：	kr 35 648,00	$ 35,648.00
数字：	35648.00	35648.00

示例：

```
Money( A, '#,##0 ¥', '.' , ',' )
```
其中 A=3564800

结果表

结果	设置 1	设置 2
字符串：	3,564,800 ¥	3,564,800 ¥
数字：	3564800	3564800

Num

Num() 格式化数字，即使用第二个参数中指定的格式将输入的数值转换为显示文本。如果省略第二个参数，它将使用数据加载脚本中设置的小数点和千位分隔符。自定义小数位和千分位分隔符的符号为可选参数。

语法：

```
Num(number[, format[, dec_sep [, thou_sep]]])
```

返回数据类型： 双

Num 函数返回同时包含字符串和数字值的双重值。该函数会获取输入表达式的数值，然后生成一个表示此数字的字符串。

参数：

参数

参数	描述
number	可以设置数字的格式。
format	指定如何设置结果字符串格式的字符串。如果省略，则使用数据加载脚本中设置的十进制和千位分隔符。
dec_sep	指定小数位数字分隔符的字符串。如果省略，则使用数据加载脚本中设置的变量 DecimalSep 的值。
thou_sep	指定千分位数字分隔符的字符串。如果省略，则使用数据加载脚本中设置的变量 ThousandSep 的值。

示例：图表表达式

示例：

下表显示字段 A 等于 35648.312 时的结果。

结果

行号旁边的	结果
Num(A)	35648.312(取决于脚本中的环境变量)
Num(A, '0.0', '.')	35648.3
Num(A, '0,00', ',')	35648,31
Num(A, '#,##0.0', '.', ',')	35,648.3
Num(A, '#,##0', '.', ',')	35 648

示例:加载脚本

加载脚本

Num 可用在加载脚本中以格式化数字,即使千分位和小数位分隔符已经在脚本中设置。下面的加载脚本包括特定的千分位和小数位分隔符,但之后使用 *Num* 来以不同方式格式化数据。

在**数据加载编辑器**中,创建新的部分,然后添加示例脚本并运行它。然后,至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

```
SET ThousandSep=','; SET DecimalSep='.'; Transactions: Load *, Num(transaction_amount) as [No formatting], Num(transaction_amount,'0') as [0], Num(transaction_amount,'#,##0') as [#,##0], Num(transaction_amount,'# ###,00') as [# ###,00], Num(transaction_amount,'# ###,00',',',' ') as [# ###,00 , ',' , ' '], Num(transaction_amount,'#,###.00','.',',') as [#,###.00 , '.' , ','], Num(transaction_amount,'$#,###.00') as [$#,###.00], ; Load * Inline [ transaction_id, transaction_date, transaction_amount, transaction_quantity, discount, customer_id, size, color_code 3750, 20180830, 12423.56, 23, 0,2038593, L, Red 3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange 3752, 20180916, 15.75, 1, 0.22, 5646471, S, blue 3753, 20180922, 1251, 7, 0, 3036491, l, Black 3754, 20180922, 21484.21, 1356, 75, 049681, xs, Red 3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, Blue 3757, 20180923, 3177.4, 21, .14, 203521, XL, Black ];
```

Qlik Sense 表格示出来自加载脚本中 *Num* 函数的不同使用的结果。表格的第四列包含不正确的格式使用,例如用途。

无格式	0	#,##0	# ###,00	# ###,00 , ',' , ''	#,###.00 , '.' , ''	$#,###.00
-59.18	-59	-59	-59###,00	-59,18	-59.18	$-59,18
15.75	16	16	16###,00	15,75	15.75	$15,75
1251	1251	1,251	1251###,00	1 251,00	1,251.00	$1,251.00
3177.4	3177	3,177	3177###,00	3 177,40	3,177.40	$3,177.40
5356.31	5356	5,356	5356###,00	5 356,31	5,356.31	$5,356.31
12423.56	12424	12,424	12424###,00	12 423,56	12,423.56	$12,423.56
21484.21	21484	21,484	21484###,00	21 484,21	21,484.21	$21,484.21

示例:加载脚本

加载脚本

Num 可用在加载脚本中将数字格式化为百分比。

在**数据加载编辑器**中,创建新的部分,然后添加示例脚本并运行它。然后,至少要将结果列中列出的字段添加到应用程序中的表格才能查看结果。

```
SET ThousandSep=','; SET DecimalSep='.'; Transactions: Load *, Num(discount,'#,##0%') as [Discount #,##0%] ; Load * Inline [ transaction_id, transaction_date, transaction_amount, transaction_quantity, discount, customer_id, size, color_code 3750, 20180830, 12423.56, 23, 0,2038593, L, Red 3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange 3752, 20180916, 15.75, 1, 0.22, 5646471, S, blue 3753, 20180922, 1251, 7, 0, 3036491, l, Black 3754, 20180922, 21484.21,
```

```
1356, 75, 049681, xs, Red 3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, Blue 3757,
20180923, 3177.4, 21, .14, 203521, XL, Black ];
```

Qlik Sense 表格示出在加载脚本中使用 *Num* 函数来格式化为百分比的结果。

折扣	Discount #,##0%
0.333333333333333	33%
0.22	22%
0	0%
.14	14%
0.1	10%
0	0%
75	7,500%

Time

Time() 用于使用数据加载脚本的系统变量或操作系统(如果不提供格式字符串)中设置的时间格式,将表达式的格式设置为时间值格式。

语法:

```
Time(number[, format])
```

返回数据类型: 双

参数:

参数

参数	说明
number	可以设置数字的格式。
format	说明如何设置结果时间字符串格式的字符串。如果省略,则使用操作系统中设置的缩写日期格式、时间格式和小数位分隔符。

示例和结果:
以下示例假设采用以下默认设置:

- 时间格式设置 1: hh:mm:ss
- 时间格式设置 2: hh.mm.ss

示例:

```
Time( A )
```

其中 A=0.375

结果表

结果	设置 1	设置 2
字符串:	09:00:00	09.00.00
数字:	0.375	0.375

示例:

```
Time( A )
```
其中 A=35648.375

结果表

结果	设置 1	设置 2
字符串:	09:00:00	09.00.00
数字:	35648.375	35648.375

示例:

```
Time( A, 'hh-mm' )
```
其中 A=0.99999

结果表

结果	设置 1	设置 2
字符串:	23-59	23-59
数字:	0.99999	0.99999

Timestamp

TimeStamp() 用于使用数据加载脚本的系统变量或操作系统(如果不提供格式字符串)中设置的时间戳格式,将表达式的格式设置为日期和时间值格式。

语法:

```
Timestamp(number[, format])
```

返回数据类型: 双

参数:

参数

参数	说明
number	可以设置数字的格式。
format	说明如何设置结果时间戳字符串格式的字符串。如果省略,则使用操作系统中设置的缩写日期格式、时间格式和小数位分隔符。

示例和结果：
以下示例假设采用以下默认设置：

- 时间戳格式设置 1：YY-MM-DD hh:mm:ss
- 时间戳格式设置 2：M/D/YY hh:mm:ss

示例：

```
Timestamp( A )
```
其中 A=35648.375

结果表

结果	设置 1	设置 2
字符串：	97-08-06 09:00:00	8/6/97 09:00:00
数字：	35648.375	35648.375

示例：

```
Timestamp( A,'YYYY-MM-DD hh.mm')
```
其中 A=35648

结果表

结果	设置 1	设置 2
字符串：	1997-08-06 00.00	1997-08-06 00.00
数字：	35648	35648

5.13 一般数字函数

在以下一般数字函数中，参数为表达式，其中 **x** 应解释为实值数。所有函数均可用于数据加载脚本和图表表达式。

常见数字函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

bitcount

BitCount() 用于返回将小数的等值二进制数中设置为 1 的位数。即该函数用于返回 **integer_number** 中的设置位，其中 **integer_number** 解释为标记的 32 位整数。

```
BitCount(integer_number)
```

div

Div() 用于返回第一个参数算术除以第二个参数的整数部分。两个参数都解释为真实数字，即它们不必是整数。

```
Div (integer_number1, integer_number2)
```

fabs

Fabs() 用于返回 **x** 的绝对值。结果为正数。

```
Fabs (x)
```

fact

Fact() 用于返回正整数 **x** 的阶乘。

```
Fact (x)
```

frac

Frac() 用于返回 **x** 的小数部分。

```
Frac (x)
```

sign

Sign() 用于根据 **x** 是正数、0 或负数分别返回 1, 0 或 -1。

```
Sign (x)
```

组合和排列函数

combin

Combin() 用于返回 **q** 元素组合数，它可从一组 **p** 项目中选取。公式如下：`Combin(p,q) = p! / q!(p-q)!` 选择项目的顺序不重要。

```
Combin (p, q)
```

permut

Permut() 用于返回 **q** 元素排列数，它可从一组 **p** 项目中选取。公式如下：`Permut(p,q) = (p)! / (p - q)!` 选择项目的顺序很重要。

```
Permut (p, q)
```

模函数

fmod

fmod() 是一个广义模函数，用于返回第一个参数（被除数）整除第二个参数（除数）的余数部分。结果为实数。两个参数都解释为实数，即它们不必是整数。

```
Fmod (a, b)
```

mod

Mod() 是一个数学模函数，用于返回整数除法的非负余数。第一个参数是被除数，第二个参数是除数，这两个参数均必须是整数值。

```
Mod (integer_number1, integer_number2)
```

奇偶校验函数

even

Even() 用于返回 True (-1)(如果 **integer_number** 为偶整数或零) 。用于返回 False (0)(如果 **integer_number** 为奇整数) , 返回 NULL(如果 **integer_number** 不是整数) 。

```
Even (integer_number)
```

odd

Odd() 用于返回 True (-1)(如果 **integer_number** 为奇整数或零) 。用于返回 False (0)(如果 **integer_number** 为偶整数) , 返回 NULL(如果 **integer_number** 不是整数) 。

```
Odd (integer_number)
```

舍入函数

ceil

Ceil() 将数值向上取整到通过 **offset** 数值偏移的 **step** 的最接近倍数。

```
Ceil (x[, step[, offset]])
```

floor

Floor() 将数值向下取整到通过 **offset** 数值偏移的 **step** 的最接近倍数。

```
Floor (x[, step[, offset]])
```

round

Round() 用于返回通过偏移 **offset** 数值向上或向下取整到 **step** 最接近倍数的结果。

```
Round ( x [ , step [ , offset ]])
```

BitCount

BitCount() 用于返回将小数的等值二进制数中设置为 1 的位数。即该函数用于返回 **integer_number** 中的设置位, 其中 **integer_number** 解释为标记的 32 位整数。

语法:

```
BitCount(integer_number)
```

返回数据类型: 整数

示例和结果:

示例和结果

示例	结果
BitCount (3)	3 的二进制是 11, 因此返回 2
BitCount (-1)	-1 的二进制是 64 个 1, 因此返回 64

Ceil

Ceil() 将数值向上取整到通过 **offset** 数值偏移的 **step** 的最接近倍数。

与 **floor** 函数比较，此函数对输入数值向下取整。

语法：

```
Ceil(x[, step[, offset]])
```

返回数据类型： 数字

参数：

参数

参数	说明
x	输入数字。
step	间隔增量。默认值为 1。
offset	定义步进间隔的底数。默认值为 0。

示例和结果：

示例和结果

示例	结果
`Ceil(2.4 )`	返回 3 在这个示例中，步进的大小为 1，步进间隔的底数为 0。 间隔为 ...0 < x <=1, 1 < x <= 2, **2< x <=3**, 3< x <=4...
`Ceil(4.2 )`	返回 5
`Ceil(3.88 ,0.1)`	返回 3.9 在这个示例中，步进间隔的大小为 0.1，步进间隔的底数为 0。 间隔为 ... 3.7 < x <= 3.8, **3.8 < x <= 3.9**, 3.9 < x <= 4.0...
`Ceil(3.88 ,5)`	返回 5
`Ceil(1.1 ,1)`	返回 2
`Ceil(1.1 ,1,0.5)`	返回 1.5 在这个示例中，步进的大小为 1，步进间隔的偏移为 0.5。意思就是步进间隔的底数为 0.5，不是 0。 间隔为 ...**0.5 < x <=1.5**, 1.5 < x <= 2.5, 2.5< x <=3.5, 3.5< x <=4.5...
`Ceil(1.1 ,1,-0.01)`	返回 1.99 间隔为 ...-0.01< x <= 0.99, **0.99< x <= 1.99**, 1.99 < x <=2.99...

Combin

Combin() 用于返回 **q** 元素组合数，它可从一组 **p** 项目中选取。公式如下：Combin(p,q) = p! / q!(p-q)! 选择项目的顺序不重要。

语法：

```
Combin(p, q)
```

返回数据类型： 整数

限制：

非整数项目将会被截短。

示例和结果：

示例和结果

示例	结果
从总共 35 个乐透号码中可以选择多少组 7 个号码的组合？ Combin(35,7)	返回 6,724,520

Div

Div() 用于返回第一个参数算术除以第二个参数的整数部分。两个参数都解释为真实数字，即它们不必是整数。

语法：

```
Div(integer_number1, integer_number2)
```

返回数据类型： 整数

示例和结果：

示例和结果

示例	结果
Div(7,2)	返回 3
Div(7.1,2.3)	返回 3
Div(9,3)	返回 3
Div(-4,3)	返回 -1
Div(4,-3)	返回 -1
Div(-4,-3)	返回 1

Even

Even() 用于返回 True (-1)(如果 **integer_number** 为偶整数或零) 。用于返回 False (0)(如果 **integer_number** 为奇整数) , 返回 NULL(如果 **integer_number** 不是整数) 。

语法:

```
Even(integer_number)
```

返回数据类型: 布尔值

示例和结果:

示例和结果

示例	结果
Even(3)	返回 0 False
Even(2 * 10)	返回 -1 True
Even(3.14)	返回 NULL

Fabs

Fabs() 用于返回 **x** 的绝对值。结果为正数。

语法:

```
fabs(x)
```

返回数据类型: 数字

示例和结果:

示例和结果

示例	结果
fabs(2.4)	返回 2.4
fabs(-3.8)	返回 3.8

Fact

Fact() 用于返回正整数 **x** 的阶乘。

语法:

```
Fact(x)
```

返回数据类型: 整数

限制:

如果数字 **x** 不是整数, 则会被截断。负数将返回 NULL。

示例和结果：

示例和结果

示例	结果
`Fact( 1 )`	返回 1
`Fact( 5 )`	返回 120 (1 * 2 * 3 * 4 * 5 = 120)
`Fact( -5 )`	返回 NULL

Floor

Floor() 将数值向下取整到通过 **offset** 数值偏移的 **step** 的最接近倍数。

与 **ceil** 函数比较，此函数对输入数值向上取整。

语法：

```
Floor(x[, step[, offset]])
```

返回数据类型： 数字

参数：

参数

参数	说明
x	输入数字。
step	间隔增量。默认值为 1。
offset	定义步进间隔的底数。默认值为 0。

示例和结果：

示例和结果

示例	结果
`Floor(2.4)`	返回 2 In this example, the size of the step is 1 and the base of the step interval is 0. The intervals are ...0 <= x <1, 1 <= x < 2, **2<= x <3**, 3<= x <4....
`Floor(4.2)`	返回 4
`Floor(3.88 ,0.1)`	返回 3.8 在这个示例中，步进间隔的大小为 0.1，步进间隔的底数为 0。 间隔为 ... 3.7 <= x < 3.8, **3.8 <= x < 3.9**, 3.9 <= x < 4.0...

示例	结果
`Floor(3.88 ,5)`	返回 0
`Floor(1.1 ,1)`	返回 1
`Floor(1.1 ,1,0.5)`	返回 0.5 在这个示例中，步进的大小为 1，步进间隔的偏移为 0.5。意思就是步进间隔的底数为 0.5，不是 0。 间隔为 ...**0.5 <= x <1.5**, 1.5 <= x < 2.5, 2.5<= x <3.5,...

Fmod

fmod() 是一个广义模函数，用于返回第一个参数（被除数）整除第二个参数（除数）的余数部分。结果为实数。两个参数都解释为实数，即它们不必是整数。

语法：

```
fmod(a, b)
```

返回数据类型： 数字

参数：

参数

参数	说明
a	被除数
b	除数

示例和结果：

示例和结果

示例	结果
`fmod( 7,2 )`	返回 1
`fmod( 7.5,2 )`	返回 1.5
`fmod( 9,3 )`	返回 0
`fmod( -4,3 )`	返回 -1
`fmod( 4,-3 )`	返回 1
`fmod( -4,-3 )`	返回 -1

Frac

Frac() 用于返回 **x** 的小数部分。

以 Frac(x) + Floor(x) = x 这样的方式定义小数。简而言之，这意味着正数的小数部分即为该数值 (x) 与小数前面的整数之间的差值。

例如：11.43 的小数部分 = 11.43 - 11 = 0.43

对于负数，比如 -1.4，Floor(-1.4) = -2，将生成以下结果：

-1.4 的小数部分 = 1.4 - (-2) = -1.4 + 2 = 0.6

语法：

```
Frac(x)
```

返回数据类型： 数字

参数：

参数

参数	说明
x	需要返回小数部分的数字。

示例和结果：

示例和结果

示例	结果
`Frac( 11.43 )`	返回 0.43
`Frac( -1.4 )`	返回 0.6
从时间戳的数字表示形式中提取时间分量，从而省略日期。 `Time(Frac(44518.663888889))`	返回 3:56:00 PM

Mod

Mod() 是一个数学模函数，用于返回整数除法的非负余数。第一个参数是被除数，第二个参数是除数，这两个参数均必须是整数值。

语法：

```
Mod(integer_number1, integer_number2)
```

返回数据类型： 整数

限制：

integer_number2 必须大于 0。

示例和结果：

示例和结果

示例	结果
Mod(7,2)	返回 1
Mod(7.5,2)	返回 NULL
Mod(9,3)	返回 0
Mod(-4,3)	返回 2
Mod(4,-3)	返回 NULL
Mod(-4,-3)	返回 NULL

Odd

Odd() 用于返回 True (-1)(如果 **integer_number** 为奇整数或零) 。用于返回 False (0)(如果 **integer_number** 为偶整数) , 返回 NULL(如果 **integer_number** 不是整数) 。

语法：

```
Odd(integer_number)
```

返回数据类型： 布尔值

示例和结果：

示例和结果

示例	结果
Odd(3)	返回 -1 True
Odd(2 * 10)	返回 0 False
Odd(3.14)	返回 NULL

Permut

Permut() 用于返回 **q** 元素排列数, 它可从一组 **p** 项目中选取。公式如下：`Permut(p,q) = (p)! / (p - q)!` 选择项目的顺序很重要。

语法：

```
Permut(p, q)
```

返回数据类型： 整数

限制：

非整数型参数将被截短。

示例和结果：

示例和结果

示例	结果
在有 8 人参加的 100 米决赛中，金牌，银牌和铜牌可以有多少种分发方式？ `Permut( 8,3 )`	返回 336

Round

Round() 用于返回通过偏移 **offset** 数值向上或向下取整到 **step** 最接近倍数的结果。

如果舍入值正处于一个时间间隔的中间，它向上取整。

语法：

```
Round(x[, step[, offset]])
```

返回数据类型： 数字

> *如果您对浮点数进行四舍五入，可能会导致错误的结果。这些舍入错误是因为浮点数是由有限数量的二进制数字表示的。因此，使用已经结果舍入的数字计算出结果。如果这些舍入错误会对您的工作产生影响，在四舍五入之前乘以要将其转换为整数的数字。*

参数：

参数

参数	说明
x	输入数字。
step	间隔增量。默认值为 1。
offset	定义步进间隔的底数。默认值为 0。

示例和结果：

示例和结果

示例	结果
`Round(3.8 )`	返回 4 在这个示例中，步进的大小为 1，步进间隔的底数为 0。 间隔为 ...0 <= x <1, 1 <= x < 2, 2<= x <3, **3<= x <4**...
`Round(3.8,4 )`	返回 4

示例	结果
Round(2.5)	返回 3。 在这个示例中，步进的大小为 1，步进间隔的底数为 0。 步进间隔为 ...0 <= x <1, 1 <= x <2, **2<= x <3**...
Round(2,4)	返回 4。向上取整，因为 2 正好是步进间隔 4 的一半。 在这个示例中，步进的大小为 4，步进间隔的底数为 0。 步进间隔为 ...**0 <= x <4**, 4 <= x <8, 8<= x <12...
Round(2,6)	返回 0。向下取整，因为 2 小于步进间隔 6 的一半。 在这个示例中，步进的大小为 6，步进间隔的底数为 0。 步进间隔为 ...**0 <= x <6**, 6 <= x <12, 12<= x <18...
Round(3.88 ,0.1)	返回 3.9 在这个示例中，步进的大小为 0.1，步进间隔的底数为 0。 间隔为 ... 3.7 <= x <3.8, **3.8 <= x <3.9**, 3.9 <= x < 4.0...
Round (3.88875,1/1000)	返回 3.889 在本例中，步长的大小为 0.001，这将数字向上舍入并限制为小数点后三位。
Round(3.88 ,5)	返回 5
Round(1.1 ,1,0.5)	返回 1.5 在这个示例中，步进的大小为 1，步进间隔的底数为 0.5。 间隔为 ...**0.5 <= x <1.5**, 1.5 <= x <2.5, 2.5<= x <3.5...

Sign

Sign() 用于根据 **x** 是正数、0 或负数分别返回 1，0 或 -1。

语法：

```
Sign(x)
```

返回数据类型： 数字

限制：

如果找不到任何数值，则返回 NULL 值。

示例和结果：

示例和结果

示例	结果
`Sign( 66 )`	返回 1
`Sign( 0 )`	返回 0
`Sign( - 234 )`	返回 -1

5.14 地理空间函数

这些函数用于在地图可视化中处理地理空间数据。Qlik Sense 遵照 GeoJSON 针对地理空间数据的规定并支持以下几项：

- Point
- Linestring
- Polygon
- Multipolygon

有关 GeoJSON 规定的更多信息，请参见：
GeoJSON.org

地理空间函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

可以使用两种类别的地理空间函数：聚合和非聚合。

聚合函数使用几何体聚合（点或地区）作为输入，并返回单一几何体。例如，可以将多个地区合并在一起，并在地图上绘制聚合的单个边界。

非聚合函数使用单一几何体并返回一个几何体。例如，对于函数 GeoGetPolygonCenter()，如果将一个地区的边界几何体设置为输入，则返回该地区中心的点几何体（经度和纬度）。

以下是聚合函数：

GeoAggrGeometry
GeoAggrGeometry() 可用于将多个区域聚合成一个较大的区域，如将多个子区域聚合成一个区域。

```
GeoAggrGeometry (field_name)
```

GeoBoundingBox
GeoBoundingBox() 可用于将几何体聚合到区域中，并用于计算包含所有坐标的最小边界框。

```
GeoBoundingBox (field_name)
```

GeoCountVertex

GeoCountVertex() 可用于查找多边形几何体包含的矢量的个数。

```
GeoCountVertex(field_name)
```

GeoInvProjectGeometry

GeoInvProjectGeometry() 可用于将几何体聚合到区域中，并可应用投影的反面。

```
GeoInvProjectGeometry(type, field_name)
```

GeoProjectGeometry

GeoProjectGeometry() 可用于将几何体聚合到区域中，并可应用投影。

```
GeoProjectGeometry(type, field_name)
```

GeoReduceGeometry

GeoReduceGeometry() 用于缩减几何体包含的矢量的个数，并将多个区域聚合成一个区域，以及显示个别区域的边界线。

```
GeoReduceGeometry (geometry)
```

以下是非聚合函数：

GeoGetBoundingBox

GeoGetBoundingBox() 可在脚本和图表表达式中用于计算包含几何体所有坐标的最小地理空间边界框。

```
GeoGetBoundingBox (geometry)
```

GeoGetPolygonCenter

GeoGetPolygonCenter() 可在脚本和图表表达式中用于计算和返回几何体的中心点。

```
GeoGetPolygonCenter (geometry)
```

GeoMakePoint

GeoMakePoint() 可在脚本和图表表达式中用于通过经度和纬度创建和标记某个点。

```
GeoMakePoint (lat_field_name, lon_field_name)
```

GeoProject

GeoProject() 可在脚本和图表表达式中用于将投影应用于几何体。

```
GeoProject (type, field_name)
```

GeoAggrGeometry

GeoAggrGeometry() 可用于将多个区域聚合成一个较大的区域，如将多个子区域聚合成一个区域。

语法：

```
GeoAggrGeometry(field_name)
```

返回数据类型：字符串

参数：

参数

参数	说明
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

通常，**GeoAggrGeometry()** 可用于组合地理空间边界数据。例如，您可能拥有某个城市郊区的邮政编码和各区域的销售收入。如果销售员的区域涉及多个邮政编码地区，则该参数可用于显示其销售区域的销售总额(而不是个别区域)，以及显示颜色填充地图的结果。

GeoAggrGeometry() 可以计算个别郊区几何体的聚合，并在数据模型中生成合并的区域几何体。然后，如果调整销售区域边界，在重新加载数据后，则在地图中会反映合并后的新边界和收入。

由于 **GeoAggrGeometry()** 是聚合函数，因此如果在脚本中使用该函数，则需要包含 **Group by** 子句的 **LOAD** 语句。

*使用 **GeoAggrGeometry()** 创建的地图边界线是合并后地区的边界线。如果要显示聚合前地区的单个边界线，可以使用 **GeoReduceGeometry()**。*

示例：
此示例加载具有区域数据的 KML 文件，然后加载具有聚合区域数据的表格。

```
[MapSource]: LOAD [world.Name], [world.Point], [world.Area] FROM [lib://Downloads/world.kml]
(kml, Table is [World.shp/Features]); Map: LOAD world.Name, GeoAggrGeometry(world.Area) as
[AggrArea] resident MapSource Group By world.Name;

Drop Table MapSource;
```

GeoBoundingBox

GeoBoundingBox() 可用于将几何体聚合到区域中，并用于计算包含所有坐标的最小边界框。

GeoBoundingBox 表示为四个值 left、right、top 和 bottom 的列表。

语法：

```
GeoBoundingBox(field_name)
```

返回数据类型：字符串

参数：

参数

参数	说明
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

GeoBoundingBox() 用于聚合一组几何体并返回最小矩形的四个坐标，其中包含聚合几何体的所有坐标。

要可视化地图上的结果，需要将生成的四个坐标的字符串转换成多边形格式、使用地理多边形格式标记转换后的字段并将该字段拖放到地图对象。然后，将会在地图可视化中显示矩形方框。

GeoCountVertex

GeoCountVertex() 可用于查找多边形几何体包含的矢量的个数。

语法：

```
GeoCountVertex(field_name)
```

返回数据类型：整数

参数：

参数

参数	说明
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

GeoGetBoundingBox

GeoGetBoundingBox() 可在脚本和图表表达式中用于计算包含几何体所有坐标的最小地理空间边界框。

GeoBoundingBox() 函数创建的地理空间边界框表示为四个值 left、right、top 和 bottom 的列表。

语法：

```
GeoGetBoundingBox(field_name)
```

返回数据类型：字符串

参数：

参数

参数	说明
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

在数据加载编辑器中不要使用包含此参数和其他非聚合地理空间函数的 ***Group by*** *子句，因为这可能会导致加载错误。*

GeoGetPolygonCenter

GeoGetPolygonCenter() 可在脚本和图表表达式中用于计算和返回几何体的中心点。

在某些情况下，需要绘制点，而不是在地图上填充颜色。如果仅以地区几何体(如边界)的形式提供现有的地理空间数据，可以使用 **GeoGetPolygonCenter()** 检索地区中心的一对经度和纬度。

语法：

```
GeoGetPolygonCenter(field_name)
```

返回数据类型：字符串

参数：

参数

参数	说明
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

在数据加载编辑器中不要使用包含此参数和其他非聚合地理空间函数的 ***Group by*** *子句，因为这可能会导致加载错误。*

GeoInvProjectGeometry

GeoInvProjectGeometry() 可用于将几何体聚合到区域中，并可应用投影的反面。

语法：

```
GeoInvProjectGeometry(type, field_name)
```

返回数据类型：字符串

参数：

参数

参数	说明
type	在转换地图几何体时使用的投影类型。这可以采用两个值之一：“unit”(默认值)，将生成 1:1 的投影，或者“mercator”，将使用标准 Mercator 投影。
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

示例：

脚本示例

示例	结果
在 Load 语句中： `GeoInvProjectGeometry ('mercator',AreaPolygon) as InvProjectGeometry`	使用 Mercator 投影的反向转换来转换加载作为 **AreaPolygon** 的几何体，并存储作为 **InvProjectGeometry** 以便在可视化中使用。

GeoMakePoint

GeoMakePoint() 可在脚本和图表表达式中用于通过经度和纬度创建和标记某个点。GeoMakePoint 以经度和纬度的顺序返回点。

语法：

```
GeoMakePoint(lat_field_name, lon_field_name)
```

返回数据类型：字符串，格式化 [经度，纬度]

参数：

参数

参数	说明
lat_field_name	字段或表达式指向表示该点的纬度的字段。
lon_field_name	字段或表达式指向表示该点的经度的字段。

*在数据加载编辑器中不要使用包含此参数和其他非聚合地理空间函数的 **Group by** 子句，因为这可能会导致加载错误。*

GeoProject

GeoProject() 可在脚本和图表表达式中用于将投影应用于几何体。

语法：

```
GeoProject(type, field_name)
```

返回数据类型：字符串

参数：

参数

参数	说明
type	在转换地图几何体时使用的投影类型。这可以采用两个值之一：“unit”(默认值)，将生成 1:1 的投影，或者“mercator”，将使用 Web Mercator 投影。
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

在数据加载编辑器中不要使用包含此参数和其他非聚合地理空间函数的 ***Group by*** *子句，因为这可能会导致加载错误。*

示例：

脚本示例

示例	结果
在 Load 语句中： `GeoProject('mercator',Area) as GetProject`	Mercator 投影应用于加载作为 **Area** 的几何体，并将结果作为 **GetProject** 存储。

GeoProjectGeometry

GeoProjectGeometry() 可用于将几何体聚合到区域中，并可应用投影。

语法：

```
GeoProjectGeometry(type, field_name)
```

返回数据类型：字符串

参数：

参数

参数	说明
type	在转换地图几何体时使用的投影类型。这可以采用两个值之一：“unit”(默认值)，将生成 1:1 的投影，或者“mercator”，将使用 Web Mercator 投影。
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点(或点集)，或者一个区域。

示例：

示例	结果
在 Load 语句中： GeoProjectGeometry ('mercator',AreaPolygon) as ProjectGeometry	使用 Mercator 投影来转换作为 **AreaPolygon** 加载的几何体，并作为 **ProjectGeometry** 存储以便在可视化中使用。

GeoReduceGeometry

GeoReduceGeometry() 用于缩减几何体包含的矢量的个数，并将多个区域聚合成一个区域，以及显示个别区域的边界线。

语法：

```
GeoReduceGeometry(field_name[, value])
```

返回数据类型： 字符串

参数：

参数

参数	说明
field_name	字段或表达式指向包含要表示的几何体的字段。这可以是提供经度和纬度的一个点（或点集），或者一个区域。
value	缩减数量以应用于几何体。缩减范围从 0 到 1,0 表不缩减，1 表示矢量的最大缩减。 *使用 value 0.9 或含复杂的大型数据集的更大值可以将矢量数缩减到视觉显示错误的级别。*

GeoReduceGeometry() 也执行与 **GeoAggrGeometry()** 类似的函数，在此函数中，将多个区域聚合成一个较大区域。如果使用 **GeoReduceGeometry()**，单个边界线与聚合前数据的差别会显示在地图上。

由于 **GeoReduceGeometry()** 是聚合函数，因此如果在脚本中使用该函数，则需要包含 **Group by** 子句的 **LOAD** 语句。

示例：
此示例加载具有区域数据的 KML 文件，然后加载具有缩小和聚合区域数据的表格。

```
[MapSource]: LOAD [world.Name], [world.Point], [world.Area] FROM [lib://Downloads/world.kml] (kml, Table is [World.shp/Features]); Map: LOAD world.Name, GeoReduceGeometry(world.Area,0.5) as [ReducedArea] resident MapSource Group By world.Name;

Drop Table MapSource;
```

5.15 解释函数

解释函数用于计算输入文本字段或表达式的内容值，以及对生成的数字值强制使用指定数据格式。使用这些函数，可以根据数据类型指定数字格式，包括属性，例如：小数位分隔符、千分位分隔符和日期格式。

解释函数都返回包含字符串和数字值的双重值，但可被视为执行一次从字符串到数字的转换。这些函数会获取输入表达式的文本值，然后生成一个表示此字符串的数字。

相比之下，格式函数则相反：它们获取数字表达式并计算其字符串值，从而指定生成文本的显示格式。

如果没有使用解释函数，Qlik Sense 会将数据解释为数字、日期、时间、时间戳和字符串的混合数据，同时对由脚本变量和操作系统定义的数字格式、日期格式和时间格式使用默认设置。

所有解释函数均可用于数据加载脚本和图表表达式。

所有数字表示形式都指定以小数点作为小数位分隔符。

解释函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Date#

Date# 用于使用在第二个参数(如果提供)中指定的格式计算表达式的日期值。如果忽视此格式代码，则使用设置于操作系统中的默认日期格式。

```
Date# (page 587)(text[, format])
```

Interval#

Interval#() 用于使用操作系统中设置的格式(默认情况下)或第二个参数(如果提供)中指定的格式，计算文本表达式的时间间隔值。

```
Interval# (page 588)(text[, format])
```

Money#

Money#() 用于使用在加载脚本或操作系统(如果不提供格式字符串)中设置的格式，将文本字符串转换为货币值。自定义小数位和千分位分隔符的符号为可选参数。

```
Money# (page 589)(text[, format[, dec_sep[, thou_sep ] ] ])
```

Num#

Num() 将文本字符串解释为数值，即使用第二个参数中指定的格式将输入字符串转换为数字。如果省略第二个参数，它将使用数据加载脚本中设置的小数点和千位分隔符。自定义小数位和千分位分隔符的符号为可选参数。

```
Num# (page 590)(text[ , format[, dec_sep[ , thou_sep]]])
```

Text

Text() 用于强制将表达式作文本进行处理，即使可能解释为数字。

```
Text(expr)
```

Time#

Time#() 用于使用数据加载脚本或操作系统(如果不提供格式字符串)中设置的时间格式，计算表达式的时间值。.

```
Time# (page 591)(text[, format])
```

Timestamp#

Timestamp#() 用于使用数据加载脚本或操作系统(如果不提供格式字符串)中设置的时间戳格式，计算表达式的日期和时间值。

```
Timestamp# (page 592)(text[, format])
```

另请参见：

- *格式函数 (page 554)*

Date#

Date# 用于使用在第二个参数(如果提供)中指定的格式计算表达式的日期值。

语法：

```
Date#(text[, format])
```

返回数据类型： 双

参数：

参数

参数	说明
text	可以计算文本字符串值。
format	描述待评估的文本字符串格式的字符串。如果遗漏了，则使用在数据加载脚本或操作系统中的系统变量中设置的日期格式。

示例和结果：

以下示例使用日期格式 **M/D/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。

将此示例脚本添加到应用程序并运行。

```
Load *,
Num(Date#(StringDate)) as Date;
LOAD * INLINE [
```

```
StringDate
8/7/97
8/6/1997
```

如果创建包括 **StringDate** 和 **Date** 的表格作为维度，结果如下：

结果

StringDate	日期
8/7/97	35649
8/6/1997	35648

Interval#

Interval#() 用于使用操作系统中设置的格式(默认情况下)或第二个参数(如果提供)中指定的格式，计算文本表达式的时间间隔值。

语法：

```
Interval#(text[, format])
```

返回数据类型： 双

参数：

参数

参数	说明
text	可以计算文本字符串值。
format	说明在将字符串转换为数字间隔时要使用的预期输入格式的字符串。 如果省略，则使用操作系统中设置的缩写日期格式、时间格式和小数位分隔符。

interval# 函数将文本时间间隔转换为数字时间间隔。

示例和结果：
以下示例假设按照操作系统设置：

- 缩写日期格式：YY-MM-DD
- 时间格式：M/D/YY
- 数字小数位分隔符：。

结果

示例	结果
`Interval#( A, 'D hh:mm' )` 其中 A='1 09:00'	1.375

Money#

Money#() 用于使用在加载脚本或操作系统(如果不提供格式字符串)中设置的格式，将文本字符串转换为货币值。自定义小数位和千分位分隔符的符号为可选参数。

语法：

```
Money#(text[, format[, dec_sep [, thou_sep ] ] ])
```

返回数据类型： 双

参数：

参数

参数	说明
text	可以计算文本字符串值。
format	说明在将字符串转换为数字间隔时要使用的预期输入格式的字符串。 如果省略，则使用在操作系统中设置的货币格式。
dec_sep	指定小数位数字分隔符的字符串。如果省略，则使用数据加载脚本中设置的 MoneyDecimalSep 值。
thou_sep	指定千分位数字分隔符的字符串。如果省略，则使用数据加载脚本中设置的 MoneyThousandSep 值。

money# 函数的作用和 **num#** 函数类似，但其以货币格式脚本变量或系统货币设置作为小数位和千分位分隔符的默认值。

示例和结果：
以下示例假定了以下两个操作系统设置：

- 货币格式默认设置 1: kr # ##0,00
- 货币格式默认设置 2: $ #,##0.00

```
Money#(A , '# ##0,00 kr' )
```
其中 A=35 648,37 kr

结果

结果	设置 1	设置 2
字符串	35 648.37 kr	35 648.37 kr
数字	35648.37	3564837

```
Money#( A, ' $#', '.', ',' )
```
其中 A= $35,648.37

结果

结果	设置 1	设置 2
字符串	$35,648.37	$35,648.37
数字	35648.37	35648.37

Num#

Num() 将文本字符串解释为数值，即使用第二个参数中指定的格式将输入字符串转换为数字。如果省略第二个参数，它将使用数据加载脚本中设置的小数点和千位分隔符。自定义小数位和千分位分隔符的符号为可选参数。

语法：

```
Num#(text[, format[, dec_sep [, thou_sep ] ] ])
```

返回数据类型： 双

Num#() 函数返回同时包含字符串和数字值的双重值。函数接受输入表达式的文本表示并生成一个数字。它不会改变数字的格式：输出的格式与输入的格式相同。

参数：

参数

参数	描述
text	可以计算文本字符串值。
format	指定第一个参数中使用的数字格式的字符串。如果省略，则使用数据加载脚本中设置的十进制和千位分隔符。
dec_sep	指定小数位数字分隔符的字符串。如果省略，则使用数据加载脚本中设置的变量 DecimalSep 的值。
thou_sep	指定千分位数字分隔符的字符串。如果省略，则使用数据加载脚本中设置的变量 ThousandSep 的值。

示例和结果：
下表显示不同 A 值的 *Num#(A, '#', '.', ',')* 的结果。

结果

行号旁边的	字符串形式	数值(此处以小数点显示)
35,648.31	35,648.31	35648.31
35 648.312	35 648.312	35648.312
35.648,3123	35.648,3123	-
35 648,31234	35 648,31234	-

Text

Text() 用于强制将表达式作文本进行处理，即使可能解释为数字。

语法：

```
Text (expr)
```

返回数据类型： 双

示例：

```
Text( A )
```
其中 A=1234

结果

字符串	数字
1234	-

示例：

```
Text( pi( ) )
```

结果

字符串	数字
3.1415926535898	-

Time#

Time#() 用于使用数据加载脚本或操作系统(如果不提供格式字符串)中设置的时间格式，计算表达式的时间值。.

语法：

```
time#(text[, format])
```

返回数据类型： 双

参数：

参数

参数	说明
text	可以计算文本字符串值。
format	描述待评估的文本字符串格式的字符串。如果省略，则使用操作系统中设置的缩写日期格式、时间格式和小数位分隔符。

示例：

- 时间格式默认设置 1: hh:mm:ss
- 时间格式默认设置 2: hh.mm.ss

```
time#( A )
```
其中 A=09:00:00

结果

结果	设置 1	设置 2
字符串：	09:00:00	09:00:00
数字：	0.375	-

示例：

- 时间格式默认设置 1: hh:mm:ss
- 时间格式默认设置 2: hh.mm.ss

```
time#( A, 'hh.mm' )
```
其中 A=09.00

结果

结果	设置 1	设置 2
字符串：	09.00	09.00
数字：	0.375	0.375

Timestamp#

Timestamp#() 用于使用数据加载脚本或操作系统(如果不提供格式字符串)中设置的时间戳格式，计算表达式的日期和时间值。

语法：

```
timestamp#(text[, format])
```

返回数据类型： 双

参数：

参数

参数	说明
text	可以计算文本字符串值。
format	描述待评估的文本字符串格式的字符串。如果省略，则使用操作系统中设置的缩写日期格式、时间格式和小数位分隔符。ISO 8601 支持时间戳。

示例：

以下示例使用日期格式 **M/D/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。

将此示例脚本添加到应用程序并运行。

```
Load *,
Timestamp(Timestamp#(String)) as TS;
LOAD * INLINE [
字符串
2015-09-15T12:13:14
1952-10-16T13:14:00+0200
1109-03-01T14:15
];
```

如果创建包括 **String** 和 **TS** 的表格作为维度，结果如下：

结果

字符串	TS
2015-09-15T12:13:14	9/15/2015 12:13:14 PM
1952-10-16T13:14:00+0200	10/16/1952 11:14:00 AM
1109-03-01T14:15	3/1/1109 2:15:00 PM

5.16 内部记录函数

内部记录函数可用于：

- 数据加载脚本(当对当前记录的评估需要一个来自以前加载的数据记录的值时)。
- 图表表达式(当需要来自可视化数据集的其他值时)。

当图表内部记录函数用于任何图表表达式时，按图表 Y 值排序或者按垂直表表达式列排序不可用。因此，这些排序替代项会自动禁用。

只有在行数少于 100 的表中才能可靠地进行自参考表达式定义，但是情况可能会发生变化，具体取决于 Qlik 引擎在什么硬件上运行。

行函数

这些函数只可用于图表表达式中。

Above

Above() 用于评估表格中列段数据内当前行上方的行的表达式。要计算的行取决于 **offset** 值，如果存在，则默认计算直接上面的行。对于除表格以外的图表，**Above()** 用于计算图表的等效垂直表中当前行上面的行的值。

```
Above - 图表函数([TOTAL [<fld{,fld}>]] expr [ , offset [,count]])
```

Below

Below() 用于评估表格中列段数据内当前行下面的行的表达式。要计算的行取决于 **offset** 值，如果存在，则默认计算直接下面的行。对于除表格以外的图表，**Below()** 用于计算图表的等效垂直表中当前列下面的行的值。

```
Below - 图表函数([TOTAL[<fld{,fld}>]] expression [ , offset [,count ]])
```

Bottom

Bottom() 用于评估表格中列段数据最后一(底部)行的表达式。要计算的行取决于 **offset** 值，如果存在，则默认计算底行。对于除表格以外的图表，用于计算图表的等效垂直表中当前列的最后一行的值。

```
Bottom - 图表函数([TOTAL[<fld{,fld}>]] expr [ , offset [,count ]])
```

Top

Top() 用于评估表格中列段数据第一(顶部)行的表达式。要计算的行取决于 **offset** 值，如果存在，则默认计算顶行。对于除表格以外的图表，**Top()** 用于计算图表的等效垂直表中当前列的第一行的值。

```
Top - 图表函数([TOTAL [<fld{,fld}>]] expr [ , offset [,count ]])
```

NoOfRows

NoOfRows() 用于返回表格中当前列段数据的行数。对于位图图表，**NoOfRows()** 用于返回图表的等效垂直表中的行数。

```
NoOfRows - 图表函数([TOTAL])
```

列函数

这些函数只可用于图表表达式中。

Column

Column() 用于返回在列中找到与 **ColumnNo** 在垂直表中找到的值对应的值，将会忽略维度。例如，**Column(2)** 用于返回第二个度量列的值。

```
Column - 图表函数(ColumnNo)
```

Dimensionality

Dimensionality() 用于返回当前行的维度数量。在透视表中，此函数返回包含非聚合内容的总维度列数，即不包含部分总和或折叠聚合。

```
Dimensionality - 图表函数 ( )
```

Secondarydimensionality

SecondaryDimensionality() 返回包含非聚合函数内容的维度透视表行数，即不包含部分总和或折叠聚合。此函数等同于水平透视表维度的 **dimensionality()** 函数。

```
SecondaryDimensionality- 图表函数 ( )
```

字段函数

FieldIndex

FieldIndex() 用于返回字段 **field_name**(按加载顺序) 中的字段值 **value** 的位置。

```
FieldIndex (field_name , value)
```

FieldValue

FieldValue() 用于返回在字段 **field_name**(按加载顺序) 的位置 **elem_no** 找到的值。

```
FieldValue (field_name , elem_no)
```

FieldValueCount

FieldValueCount() 是一个**整数**函数，用于返回字段中相异值的数量。

```
FieldValueCount (field_name)
```

透视表函数

这些函数只可用于图表表达式中。

After

After() 用于返回使用透视表的维度值评估的表达式值，因为维度值显示在透视表的行段内当前列之后的列。

```
After - 图表函数([TOTAL] expression [ , offset [,n]])
```

Before

Before() 返回使用透视表的维度值评估的表达式，因为维度值显示在透视表的行段内当前列之前的列。

```
Before - 图表函数([TOTAL] expression [ , offset [,n]])
```

First

First() 用于返回使用透视表的维度值评估的表达式值，因为维度值显示在透视表的当前行段的第一列。在除透视表之外的所有图表类型中，此函数会返回 NULL。

```
First - 图表函数([TOTAL] expression [ , offset [,n]])
```

Last

Last() 返回使用透视表的维度值评估的表达式值，因为维度值显示在透视表的当前行段的最后一列。在除透视表之外的所有图表类型中，此函数会返回 NULL。

```
Last - 图表函数([TOTAL] expression [ , offset [,n]])
```

ColumnNo

ColumnNo() 用于返回透视表的当前行段中的当前列数。第一列是数字 1。

```
ColumnNo - 图表函数([TOTAL])
```

NoOfColumns

NoOfColumns() 用于返回透视表的当前行段中的列数。

```
NoOfColumns - 图表函数([TOTAL])
```

数据加载脚本中的内部记录函数

Exists

Exists() 用于确定是否已经将特定字段值加载到数据加载脚本中的字段。此函数用于返回 TRUE 或 FALSE，这样它可以用于 **LOAD** 语句或 **IF** 语句中的 **where** 子句。

```
Exists (field_name [, expr])
```

LookUp

Lookup() 用于查找已经加载的表格，并返回与在字段 **match_field_name** 中第一次出现的值 **match_field_value** 对应的 **field_name** 值。表格可以是当前表格或之前加载的其他表格。

```
LookUp (field_name, match_field_name, match_field_value [, table_name])
```

Peek

Peek() 用于在表格中返回已经加载行的字段值。可以将行号指定为表格。如果未指定行号，将使用上次加载的记录。

```
Peek (field_name[, row_no[, table_name ] ])
```

Previous

Previous() 用于查找使用因 **where** 子句而未丢弃的以前输入记录的数据的 **expr** 表达式的值。在内部表格的首个记录中，此函数将返回 NULL 值。

```
Previous (page 625)(expr)
```

另请参见：

- *范围函数 (page 643)*

Above - 图表函数

Above() 用于评估表格中列段数据内当前行上方的行的表达式。要计算的行取决于 **offset** 值，如果存在，则默认计算直接上面的行。对于除表格以外的图表，**Above()** 用于计算图表的等效垂直表中当前行上面的行的值。

语法：

```
Above([TOTAL] expr [ , offset [,count]])
```

返回数据类型： 双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。

参数	说明
offset	指定 offsetn(大于 0) 后，将表达式评估从当前行开始向上移动 n 行。 指定 0 偏移量可以计算当前行上的表达式的值。 指定负偏移量数值后，使 Above 函数效果类似于具有相应正偏移量数值的 Below 函数。
count	通过指定第三个参数 **count** 大于 1，函数将返回一连串 **count** 值，每个值对应一个从原始单元格开始向上计数的 **count** 表格行。 此时，可以将该函数用作任何特殊范围函数的参数。 *范围函数 (page 643)*
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

在列段数据的第一行中返回 NULL 值，因为其上没有行。

列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算，不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度，或者如果已指定 TOTAL 限定符，则计算整个表格中的表达式。

如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

限制：

递归调用将返回 NULL 值。

示例和结果：

Example 1:

示例 1 的表格可视化

Customer	Sum([Sales])	Above(Sum(Sales))	Sum(Sales)+Above(Sum(Sales))	Above offset 3	Higher?
	2566	-	-	-	-
Astrida	587	-	-	-	-
Betacab	539	587	1126	-	-
Canutility	683	539	1222	-	Higher
Divadip	757	683	1440	1344	Higher

在此示例中显示的表格的屏幕截图中，表格可视化内容通过维度 **Customer** 和以下度量进行创建：`Sum(Sales)` 和 `Above(Sum(Sales))`。

对于包含 **Astrida** 的行 **Customer**，列 Above(Sum(Sales)) 返回 NULL，因为其上没有行。**Betacab** 行的结果显示 **Astrida** 的 Sum(Sales) 值，**Canutility** 的结果显示 **Betacab** 的 **Sum(Sales)** 值，以此类推。

对于标有 Sum(Sales)+Above(Sum(Sales)) 的列，**Betacab** 的行将显示行 **Betacab** + **Astrida** (539+587) 的 **Sum(Sales)** 值的相加结果。**Betacab** 行的结果将显示 **Canutility** + **Canutility** (683+539) 的 **Sum (Sales)** 值的相加结果。

使用表达式 `Sum(Sales)+Above(Sum(Sales), 3)` 创建的标有 Above offset 3 的度量具有参数 **offset**(已设置为 3)，并且能够获取当前行上面三行中的值。它将当前 **Customer** 的 **Sum(Sales)** 值添加到上面三行 **Customer** 的值中。对前三个 **Customer** 行返回的值是 NULL 值。

此表格还显示了更复杂的度量：根据 `Sum(Sales)+Above(Sum(Sales))` 创建的一个值以及根据 **Higher?** 创建的一个标有 `IF(Sum(Sales)>Above(Sum(Sales)), 'Higher')` 的值。

此函数也可以用于图表(如条形图)，但不能用于表格。

对于其他图表类型，应将图表转换成等效垂直表，这样您就可以轻松解释该函数涉及到的行。

Example 2:

在此示例中显示的表格的屏幕截图中，已将更多维度添加到可视化内容中：**Month** 和 **Product**。对于包含多个维度的图表，表达式(包含 **Above**、**Below**、**Top** 和 **Bottom** 函数)的结果取决于 Qlik Sense 对列维度进行排序的顺序。Qlik Sense 根据最后排序的维度得出的列段数据计数函数的值。可以在 **排序** 下的属性面板中控制列排序顺序，并且列不一定按顺序显示在表格中。

在以下示例 2 的表格可视化屏幕截图中，最后排序的维度是 **Month**，因此 **Above** 函数基于月评估。每个月(**Jan** 到 **Aug**)，即一个列段数据的每个 **Product** 值都有一系列结果。随后是下一个列段数据的一系列结果：下一个 **Product** 每个 **Month** 的结果。每个 **Product** 的每个 **Customer** 值都将有一个列段数据。

示例 2 的表格可视化

Customer	Product	Month	Sum([Sales])	Above(Sum(Sales))
			2566	-
Astrida	AA	Jan	46	-
Astrida	AA	Feb	60	46
Astrida	AA	Mar	70	60
Astrida	AA	Apr	13	70
Astrida	AA	May	78	13
Astrida	AA	Jun	20	78
Astrida	AA	Jul	45	20
Astrida	AA	Aug	65	45

Example 3:

在示例 3 的表格可视化屏幕截图中，最后排序的维度是 **Product**。为此，可在属性面板的“排序”标签中将维度 Product 移到第 3 个位置。每个 **Product** 都会评估 **Above** 函数，因为只有两个产品 **AA** 和 **BB**，并且每个系列只有一个非空结果。在月 **Jan** 的 **BB** 行中，**Above(Sum(Sales))** 的值为 46。对于 **AA** 行，值为 NULL。对于任何一个月，每个 **AA** 行的值将始终为 NULL，因为在 AA 行的上方没有任何 **Product** 值。在月 **Feb** 的 **AA** 和 **BB** 行中评估第二个系列，对于 **Customer** 值 **Astrida** 以此类推。当为 **Astrida** 评估完所有月份后，为第二个 **Customer**Betacab 值重复此顺序，以此类推。

示例 3 的表格可视化

Customer	Product	Month	Sum([Sales])	Above(Sum(Sales))
			2566	-
Astrida	AA	Jan	46	-
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	-
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	-
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	-
Astrida	BB	Apr	13	13

示例 4:

Example 4:	**结果**
Above 函数可用作范围函数的输入。例如：`RangeAvg (Above(Sum(Sales),1,3))`。	在 Above() 函数的参数中，将 offset 设置为 1，并将 count 设置为 3。函数在列段数据（其中有一行）当前行正上方的三行中查找表达式 Sum(Sales) 的结果。这三个值用作 RangeAvg() 函数的输入，用于查找所提供的数字范围中的平均值。 以 Customer 为维度的表格为 RangeAvg() 表达式提供了以下结果。 Astrida - Betacab 587 Canutility 563 Divadip: 603

示例中所使用的数据：

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
```

```
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

另请参见：

- *Below - 图表函数 (page 600)*
- *Bottom - 图表函数 (page 603)*
- *Top - 图表函数 (page 626)*
- *RangeAvg (page 645)*

Below - 图表函数

Below() 用于评估表格中列段数据内当前行下面的行的表达式。要计算的行取决于 **offset** 值，如果存在，则默认计算直接下面的行。对于除表格以外的图表，**Below()** 用于计算图表的等效垂直表中当前列下面的行的值。

语法：

```
Below([TOTAL] expr [ , offset [,count ]])
```

返回数据类型： 双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。

参数	说明
offset	指定 **offset**n 大于 1 后，将表达式评估从当前行开始向下移动 n 行。 指定 0 偏移量可以计算当前行上的表达式的值。 指定负偏移量数值后，使 **Below** 函数效果类似于具有相应正偏移量数值的 **Above** 函数。
count	通过指定第三个参数 **count** 大于 1，函数将返回一连串 **count** 值，每个值对应一个从原始单元格开始向下计数的 **count** 表格行。此时，可以将该函数用作任何特殊范围函数的参数。 *范围函数 (page 643)*
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

在列段数据的最后一行中返回 NULL 值，因为该行下面没有其他行。

列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算，不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度，或者如果已指定 TOTAL 限定符，则计算整个表格中的表达式。

如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

限制：

递归调用将返回 NULL 值。

示例和结果：

Example 1:

示例 1 的表格可视化

Customer	Sum([Sales])	Below(Sum(Sales))	Sum(Sales)+Below(Sum(Sales))	Below + Offset 3	Higher
	2566	-	-	-	-
Astrida	587	539	1126	1344	Higher
Betacab	539	683	1222	-	-
Canutility	683	757	1440	-	-
Divadip	757	-	-	-	-

在示例 1 的屏幕截图中显示的表格中，表格可视化内容通过维度 **Customer** 和以下度量进行创建：`Sum(Sales)` 和 `Below(Sum(Sales))`。

对于包含 **Divadip** 的 **Customer** 行，列 **Below(Sum(Sales))** 列返回 NULL，因为其下没有行。**Canutility** 行的结果显示 **Divadip** 的 Sum(Sales) 值，**Betacab** 的结果显示 **Canutility** 的 **Sum(Sales)** 值，以此类推。

此表格还显示了更复杂的度量，您可在标记以下内容的列中看到：`Sum(Sales)+Below(Sum(Sales))`、**Below +Offset 3** 和 **Higher?**。这些表达式的工作方式如以下段落所述。

对于标有 **Sum(Sales)+Below(Sum(Sales))** 的列，**Astrida** 的行将显示行 **Betacab** + **Astrida** (539+587) 的 **Sum(Sales)** 值的相加结果。**Betacab** 行的结果将显示 **Canutility** + **Betacab** (539+683) 的 **Sum (Sales)** 值的相加结果。

使用表达式 `Sum(Sales)+Below(Sum(Sales), 3)` 创建的标有 **Below +Offset 3** 的度量具有参数 **offset** (已设置为 3)，并且能够获取当前行下面三行中的值。它将当前 **Customer** 的 **Sum(Sales)** 值添加到下面三行 **Customer** 的值中。后三个 **Customer** 行的值是 NULL 值。

标记 **Higher?** 的度量通过以下表达式创建：`IF(Sum(Sales)>Below(Sum(Sales)), 'Higher')`。此表达式将度量 **Sum(Sales)** 当前行的值与其下一行进行比较。如果当前行的值较大，则输出文本“Higher”。

此函数也可以用于图表(如条形图)，但不能用于表格。

对于其他图表类型，应将图表转换成等效垂直表，这样您就可以轻松解释该函数涉及到的行。

对于包含多个维度的图表，表达式(包含 **Above**、**Below**、**Top** 和 **Bottom** 函数) 的结果取决于 Qlik Sense 对列维度进行排序的顺序。Qlik Sense 根据最后排序的维度得出的列段数据计数函数的值。可以在**排序**下的属性面板中控制列排序顺序，并且列不一定按顺序显示在表格中。有关更多信息，请参阅 **Above** 函数中的示例：2。

示例 2

Example 2:	**结果**
Below 函数可用作范围函数的输入。例如：`RangeAvg (Below(Sum(Sales),1,3))`。	在 **Below()** 函数的参数中，将 offset 设置为 1，并将 count 设置为 3。函数在列段数据(其中有一行) 当前行正下方的三行中查找表达式 **Sum (Sales)** 的结果。这三个值用作 RangeAvg() 函数的输入，用于查找所提供的数字范围中的平均值。 以 **Customer** 为维度的表格为 RangeAvg() 表达式提供了以下结果。
	Astrida 659.67 Betacab 720 Canutility 757 Divadip: -

示例中所使用的数据:

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

另请参见:

- *Above - 图表函数 (page 596)*
- *Bottom - 图表函数 (page 603)*
- *Top - 图表函数 (page 626)*
- *RangeAvg (page 645)*

Bottom - 图表函数

Bottom() 用于评估表格中列段数据最后一(底部)行的表达式。要计算的行取决于 **offset** 值,如果存在,则默认计算底行。对于除表格以外的图表,用于计算图表的等效垂直表中当前列的最后一行的值。

语法:

```
Bottom([TOTAL] expr [ , offset [,count ]])
```

返回数据类型：双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
offset	指定 **offset**n 大于 1 后，将表达式评估向上移到底行上面的 n 行。 指定负偏移量数值后，使 **Bottom** 函数效果类似于具有相应正偏移量数值的 **Top** 函数。
count	通过指定第三个参数 **count** 大于 1，函数返回的不是一个值，而是一连串 **count** 值，每个值对应当前列段数据的最后一个 **count** 行中的一行。此时，可以将该函数用作任何特殊范围函数的参数。 *范围函数 (page 643)*
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算，不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度，或者如果已指定 TOTAL 限定符，则计算整个表格中的表达式。

如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

限制：

递归调用将返回 NULL 值。

示例和结果：

示例 1 的表格可视化

Customer	Sum([Sales])	Bottom(Sum(Sales))	Sum(Sales)+Bottom(Sum(Sales))	Bottom offset 3
	2566	**757**	**3323**	**3105**
Astrida	587	757	1344	1126
Betacab	539	757	1296	1078
Canutility	683	757	1440	1222
Divadip	757	757	1514	1296

在此示例中显示的表格的屏幕截图中，表格可视化内容通过维度 **Customer** 和以下度量进行创建：`Sum(Sales)` 和 `Bottom(Sum(Sales))`。

全部行的 **Bottom(Sum(Sales))** 列均返回 757，因为此值是底行值：**Divadip**。

此表格还显示了更复杂的度量：根据 `Sum(Sales)+Bottom(Sum(Sales))` 创建的一个值以及标有 **Bottom offset 3** 的一个值，后者使用表达式 `Sum(Sales)+Bottom(Sum(Sales), 3)` 创建，且具有设置为 **offset** 的参数 3。它将当前行的 **Sum(Sales)** 值添加到从底行开始第三行中的值中，即，当前行加上 **Betacab** 的值。

示例：2

在此示例中显示的表格的屏幕截图中，已将更多维度添加到可视化内容中：**Month** 和 **Product**。对于包含多个维度的图表，表达式（包含 **Above**、**Below**、**Top** 和 **Bottom** 函数）的结果取决于 Qlik Sense 对列维度进行排序的顺序。Qlik Sense 根据最后排序的维度得出的列段数据计数函数的值。可以在 **排序** 下的属性面板中控制列排序顺序，并且列不一定按顺序显示在表格中。

在第一个表格中，基于 **Month** 评估表达式，在第二个表格中，基于 **Product** 评估表达式。度量 **End value** 包含表达式 `Bottom(Sum(Sales))`。**Month** 的底行是 Dec，屏幕截图中所示 **Product** 的 Dec 的值是 22。（某些行在屏幕截图外编辑，以便节省空间。）

示例 2 的第二个表格。End value 度量的 Bottom 的值基于 Month (Dec)。

Customer	Product	Month	Sum(Sales)	End value
			2566	-
Astrida	AA	Jan	46	22
Astrida	AA	Feb	60	22
Astrida	AA	Mar	70	22
Astrida	AA	Sep	78	22
Astrida	AA	Oct	12	22
Astrida	AA	Nov	78	22
Astrida	AA	Dec	22	22
Astrida	BB	Jan	46	22

示例 2 的第二个表格。End value 度量的 Bottom 的值基于 Product(Astrida 的 BB)。

Customer	Product	Month	Sum(Sales)	End value
			2566	-
Astrida	AA	Jan	46	46
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	60
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	70
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	13
Astrida	BB	Apr	13	13

有关更多信息，请参阅 **Above** 函数中的示例：2。

示例 3

示例：3	**结果**
Bottom 函数可用作范围函数的输入。例如：`RangeAvg (Bottom(Sum(Sales),1,3))`。	在 **Bottom()** 函数的参数中，将 offset 设置为 1，并将 count 设置为 3。函数在列段数据中底行上方的行开始的三行上查找表达式 **Sum(Sales)** 的结果（因为 offset=1），并且列段数据（其中有一行）上方有两行。这三个值用作 RangeAvg() 函数的输入，用于查找所提供的数字范围中的平均值。 以 **Customer** 为维度的表格为 RangeAvg() 表达式提供了以下结果。
	Astrida 659.67 Betacab 659.67 Canutility 659.67 Divadip： 659.67

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
```

```
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

另请参见：

Top - 图表函数 (page 626)

Column - 图表函数

Column() 用于返回在列中找到与 **ColumnNo** 在垂直表中找到的值对应的值，将会忽略维度。例如，**Column(2)** 用于返回第二个度量列的值。

语法：

Column(ColumnNo)

返回数据类型： 双

参数：

参数

参数	说明
ColumnNo	表格中包含度量的列的列数。 *Column() 函数会忽略维度列。*

限制：

如果 **ColumnNo** 引用没有度量的列，则返回 NULL 值。

递归调用将返回 NULL 值。

示例和结果：

示例：总销售额百分比

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	505	29.70
A	AA	16	4	64	505	12.67
A	BB	9	9	81	505	16.04
B	BB	10	5	50	505	9.90
B	CC	20	2	40	505	7.92
B	DD	25	-	0	505	0.00
C	AA	15	8	120	505	23.76
C	CC	19	-	0	505	0.00

示例：所选客户的销售额百分比

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	295	50.85
A	AA	16	4	64	295	21.69
A	BB	9	9	81	295	27.46

示例和结果

示例	结果
使用以下表达式将 Order Value 作为度量添加到表格中：`Sum (UnitPrice*UnitSales)`。 使用以下表达式将 Total Sales Value 添加为度量：`Sum(TOTAL UnitPrice*UnitSales)` 使用以下表达式将 % Sales 添加为度量：`100*Column(1)/Column(2)`	根据 Order Value 列获取 Column(1) 的结果，因为此列是第一个度量列。 根据 Total Sales Value 获取 Column(2) 的结果，因为此列是第二个度量列。 请参阅示例 *总销售额百分比 (page 608)* 中 % Sales 列的结果。
选择 Customer A。	此选择项会更改 Total Sales Value，因此会更改 %Sales。请参阅示例 *所选客户的销售额百分比 (page 608)*。

示例中所使用的数据：

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

Dimensionality - 图表函数

Dimensionality() 用于返回当前行的维度数量。在透视表中，此函数返回包含非聚合内容的总维度列数，即不包含部分总和或折叠聚合。

语法：

```
Dimensionality ( )
```

返回数据类型： 整数

限制：

此函数仅可用于图表。对于除透视表之外的所有图表类型，它会返回除总计之外的所有行的维度数，即 0。

示例：使用维数的图表表示

示例：图表表达式

Dimensionality() 函数可与透视表一起用作图表表达式，其中您希望根据具有非聚合数据的行中的维度数应用不同的单元格格式。本例使用 Dimensionality() 函数将背景色应用于与给定条件匹配的表格单元格。

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
ProductSales:  Load * inline [ Country,Product,Sales,Budget Sweden,AA,100000,50000
Germany,AA,125000,175000 Canada,AA,105000,98000 Norway,AA,74850,68500 Ireland,AA,49000,48000
Sweden,BB,98000,99000 Germany,BB,115000,175000 Norway,BB,71850,68500 Ireland,BB,31000,48000 ]
(delimiter is ',');
```

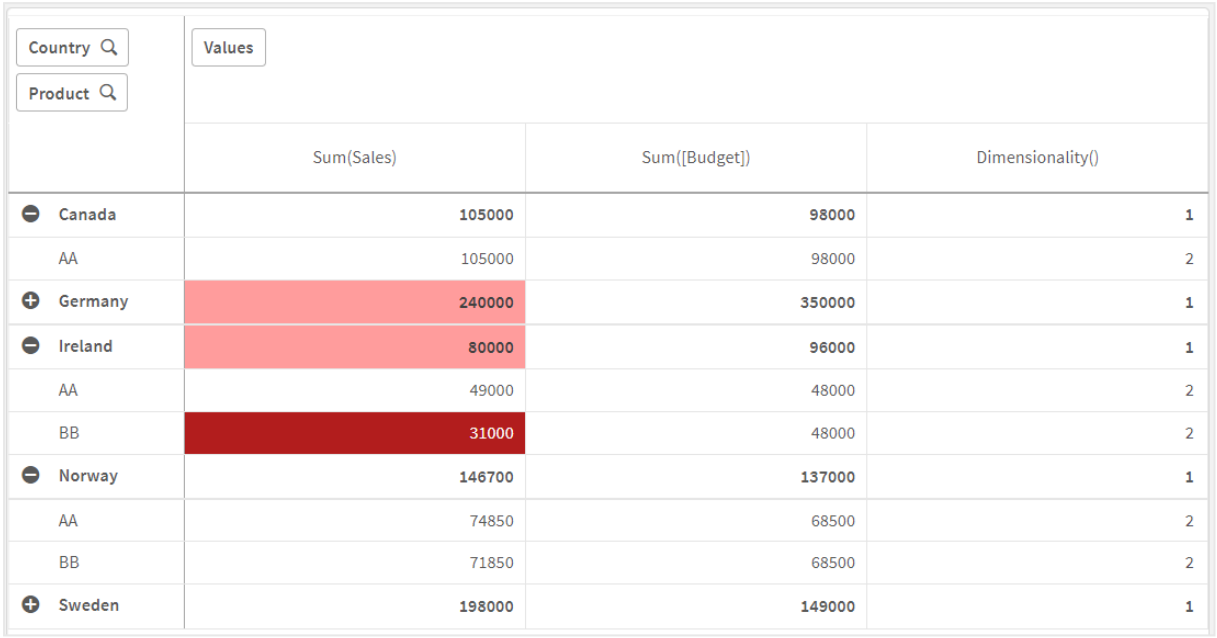
图表表达式

在 Qlik Sense 工作表中创建头饰表可视化，以 **Country** 和 **Product** 为维度。添加 **Sum(Sales)**、**Sum (Budget)** 和 **Dimensionality()** 作为度量。

在**属性**面板中，输入以下表达式作为 **Sum(Sales)** 度量值的**背景色表达式**：

```
If(Dimensionality()=1 and Sum(Sales)<Sum(Budget),RGB(255,156,156), If(Dimensionality()=2 and
Sum(Sales)<Sum(Budget),RGB(178,29,29) ))
```

结果：



Country / Product	Values		
	Sum(Sales)	Sum([Budget])	Dimensionality()
Canada	105000	98000	1
AA	105000	98000	2
Germany	240000	350000	1
Ireland	80000	96000	1
AA	49000	48000	2
BB	31000	48000	2
Norway	146700	137000	1
AA	74850	68500	2
BB	71850	68500	2
Sweden	198000	149000	1

解释

表达式 `If(Dimensionality()=1 and Sum(Sales)<Sum(Budget),RGB(255,156,156), If(Dimensionality()=2 and Sum(Sales)<Sum(Budget),RGB(178,29,29)))` 包含检查维度值以及每个产品的 Sum(Sales) 和 Sum(Budget) 的条件语句。如果满足这些条件，则对 Sum(Sales) 值应用背景色。

Exists

Exists() 用于确定是否已经将特定字段值加载到数据加载脚本中的字段。此函数用于返回 TRUE 或 FALSE，这样它可以用于 **LOAD** 语句或 **IF** 语句中的 **where** 子句。

> *您也可使用 **Not Exists()** 来确定是否尚未加载字段值，但是如果要在 **where** 子句中使用 **Not Exists()**，建议您小心。**Exists()** 函数在当前表格中测试之前加载的表格和之前加载的值。因此，仅加载第一次出现的值。如果遇到第二次出现的值，值已经被加载。有关更多信息，请查看示例。*

语法：

```
Exists(field_name [, expr])
```

返回数据类型：布尔值

参数：

参数

参数	说明
field_name	您希望在其中搜索值的字段的名称。您可使用没有引号的确切字段名称。 字段必须已经由脚本加载。这意味着您无法在脚本中进一步往下引用在子句中加载的字段。
expr	您希望检查值是否存在。您可使用引用当前加载语句中一个或数个字段的确切值或表达式。 *您无法引用未包含在当前加载语句中的字段。* 该参数为可选。如果您忽略它，函数将检查当前记录中的 **field_name** 的值是否已存在。

示例和结果：

示例 1

```
Exists (Employee)
```

如果当前记录中的字段值 **Employee** 已存在于任何以前已读入的包含该字段的记录，则返回 -1 (True)。

语句 `Exists (Employee, Employee)` 和 `Exists (Employee)` 功能相同。

示例 2

```
Exists(Employee, 'Bill')
```

如果在字段 **Employee**(员工) 的当前内容中发现字段值 **'Bill'**，则返回 -1 (True)。

示例 3

```
Employees: LOAD * inline [ Employee|ID|Salary Bill|001|20000 John|002|30000 Steve|003|35000 ] (delimiter is '|');  Citizens: Load * inline [ Employee|Address Bill|New York Mary|London Steve|Chicago Lucy|Madrid Lucy|Paris John|Miami ] (delimiter is '|') where Exists (Employee); Drop Tables Employees;
```

由此得到表格，您可在借助维度 Employee(员工) 和 Address(地址) 在表格可视化中使用该表格。

where 子句：`where Exists (Employee)`，是指只能从表格 Citizens(市民) 将同时位于 Employees(员工) 中的姓名加载到新表格。Drop 语句删除表格 Employees(员工) 以避免混淆。

结果

Employee	Address
Bill	New York
John	Miami
Steve	Chicago

示例 4:

```
Employees: Load * inline [  Employee|ID|Salary Bill|001|20000 John|002|30000 Steve|003|35000 ] (delimiter is '|');   Citizens: Load * inline [ Employee|Address Bill|New York Mary|London Steve|Chicago Lucy|Madrid Lucy|Paris John|Miami ] (delimiter is '|') where not Exists (Employee);   Drop Tables Employees;
```

where 子句包括 not: `where not Exists (Employee)`。

这意味着只能从表格 Citizens(市民) 将不在 Employees(员工) 中的姓名加载到新表格。

请注意在 Citizens(市民) 中对于 Lucy 有两个值，但是在结果表格中仅包含了一个。当您加载带值 Lucy 的第一行时，其包含在 Employee 字段中。因此，当检查第二行时，已存在值。

结果

Employee	Address
Mary	London
Lucy	Madrid

示例 5

该示例示出如何加载所有值。

```
Employees: Load Employee As Name; LOAD * inline [ Employee|ID|Salary Bill|001|20000 John|002|30000 Steve|003|35000 ] (delimiter is '|');   Citizens: Load * inline [ Employee|Address Bill|New York Mary|London Steve|Chicago Lucy|Madrid Lucy|Paris John|Miami ] (delimiter is '|') where not Exists (Name, Employee);   Drop Tables Employees;
```

要得到 Lucy 的所有值，您需要进行两项更改：

- 在 Employee 重命名为 Name 的位置插入了 Employees表的前一个加载。
  `Load Employee As Name;`
- 在 Citizens 中将 Where 条件更改为：
  `not Exists (Name, Employee).`

这将为 Name 和 Employee 创建字段。如果检查带 Lucy 的第二行，它仍未存在于 Name(名称) 中。

结果

Employee	Address
Mary	London
Lucy	Madrid
Lucy	Paris

FieldIndex

FieldIndex() 用于返回字段 **field_name**(按加载顺序) 中的字段值 **value** 的位置。

语法:

```
FieldIndex(field_name , value)
```

返回数据类型: 整数

参数:

参数

参数	说明
field_name	需要索引的字段的名称。例如, 表格中的列。必须指定作为字符串值。这意味着必须用单引号将字段名称括起来。
value	**field_name** 字段的值。

限制:

如果无法在字段 **field_name** 的字段值中找到 **value**, 则返回 0。

示例和结果:

下例使用字段:表格 **Names** 中的 **First name**。

示例和结果

示例	结果
将示例数据添加到应用程序并运行。	加载表格 **Names** 作为样本数据。
图表函数:在包含维度 First name 的表格中, 添加作为度量:	
`FieldIndex ('First name','John')`	1, 因为“John”在 **First name** 字段的加载顺序中第一个显示。请注意, 在筛选器窗格中, **John** 将作为从顶部开始的第 2 个值显示, 因为是按字母顺序排序, 不像在加载顺序中一样。
`FieldIndex ('First name','Peter')`	4, 因为 **FieldIndex()** 仅返回一个值, 该值是在加载顺序中第一个出现的值。

示例	结果
脚本函数:在加载表格 **Names** 作为示例数据的情况下:	
John1: Load FieldIndex('First name','John') as MyJohnPos Resident Names;	MyJohnPos=1, 因为“John”在 **First name** 字段的加载顺序中第一个显示。请注意, 在筛选器窗格中, **John** 将作为从顶部开始的第 2 个值显示, 因为是按字母顺序排序, 不像在加载顺序中一样。
Peter1: Load FieldIndex('First name','Peter') as MyPeterPos Resident Names;	MyPeterPos=4, 因为 **FieldIndex()** 仅返回一个值, 该值是在加载顺序中第一个出现的值。

示例中所使用的数据:

```
Names: LOAD * inline [ First name|Last name|Initials|Has cellphone John|Anderson|JA|Yes
Sue|Brown|SB|Yes Mark|Carr|MC|No Peter|Devonshire|PD|No Jane|Elliot|JE|Yes Peter|Franc|PF|Yes
] (delimiter is '|');                    John1: Load FieldIndex('First name','John') as MyJohnPos Resident
Names;   Peter1: Load FieldIndex('First name','Peter') as MyPeterPos Resident Names;
```

FieldValue

FieldValue() 用于返回在字段 **field_name**(按加载顺序) 的位置 **elem_no** 找到的值。

语法:

```
FieldValue(field_name , elem_no)
```

返回数据类型: 双

参数:

参数

参数	说明
field_name	需要值的字段的名称。例如, 表格中的列。必须指定作为字符串值。这意味着必须用单引号将字段名称括起来。
elem_no	按加载顺序返回值的字段的位置(元素) 数量。这相当于表格中的行, 但它取决于加载元素(行) 的顺序。

限制:

如果 **elem_no** 大于字段值数量, 则返回 NULL。

示例

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下示例。

```
Names:                                  LOAD * inline [ First name|Last name|Initials|Has cellphone John|And
Sue|Brown|SB|Yes Mark|Carr|MC |No Peter|Devonshire|PD|No Jane|Elliot|JE|Yes Peter|Franc|PF|Yes
] (delimiter is '|');                                   John1:                                                  Load
Names;  Peter1:  Load FieldValue('First name',5) as MyPos2 Resident Names;
```

创建可视化

在 Qlik Sense 工作表中创建表格可视化。将字段 **First name**、**MyPos1** 和 **MyPos2** 添加至表格。

结果

First name	MyPos1	MyPos2
Jane	John	Jane
John	John	Jane
Mark	John	Jane
Peter	John	Jane
Sue	John	Jane

解释

FieldValue('First name','1') 将 John 作为所有名字的 **MyPos1** 的值，因为 John 在**名字**字段的加载顺序中出现在第一位。请注意，在筛选器窗格中，John 将作为从顶部开始的第 2 个值显示在 Jane 后面，因为是按字母顺序排序，不像在加载顺序中一样。

FieldValue('First name','5') 将 Jane 作为所有名字的 **MyPos2** 的值，因为 Jane 在 **First name** 字段的加载顺序中出现在第五位。

FieldValueCount

FieldValueCount() 是一个**整数**函数，用于返回字段中相异值的数量。

部分重新加载可以从数据中删除值，而这些值不会反映在返回的数字中。返回的数字将对应于初始重新加载或任何后续部分重新加载中加载的所有不同值。

语法：

```
FieldValueCount(field_name)
```

返回数据类型：整数

参数：

参数

参数	描述
field_name	需要值的字段的名称。例如，表格中的列。必须指定作为字符串值。这意味着必须用单引号将字段名称括起来。

示例和结果：

下例使用字段：表格 **Names** 中的 **First name**。

示例和结果

示例	结果
将示例数据添加到应用程序并运行。	加载表格 **Names** 作为样本数据。
图表函数：在包含维度 First name 的表格中，添加作为度量：	
`FieldValueCount('First name')`	5，因为 **Peter** 显示两次。
`FieldValueCount('Initials')`	6，因为 **Initials** 只有特殊值。
脚本函数：在加载表格 **Names** 作为示例数据的情况下：	

`FieldCount1:`
 	
`Load FieldValueCount('First name') as MyFieldCount1`
 	
`Resident Names;`
	`MyFieldCount1=5`，因为“Peter”显示两次。

`FieldCount2:`
 	
`Load FieldValueCount('Initials') as MyInitialsCount1`
 	
`Resident Names;`
	`MyFieldCount1=6`，因为“Initials”只有特殊值。

示例中所使用的数据：

```
Names: LOAD * inline [ First name|Last name|Initials|Has cellphone John|Anderson|JA|Yes
Sue|Brown|SB|Yes Mark|Carr|MC|No Peter|Devonshire|PD|No Jane|Elliot|JE|Yes Peter|Franc|PF|Yes
] (delimiter is '|');                    FieldCount1: Load FieldValueCount('First name') as MyFieldCount1
Resident Names;   FieldCount2: Load FieldValueCount('Initials') as MyInitialsCount1 Resident
Names;
```

LookUp

Lookup() 用于查找已经加载的表格，并返回与在字段 **match_field_name** 中第一次出现的值 **match_field_value** 对应的 **field_name** 值。表格可以是当前表格或之前加载的其他表格。

语法：

```
lookup(field_name, match_field_name, match_field_value [, table_name])
```

返回数据类型：双

参数：

参数

参数	说明
field_name	需要返回值的字段的名称。输入值必须为字符串(例如引用的文字)。
match_field_name	要在其中查找 **match_field_value** 的字段的名称。输入值必须为字符串(例如引用的文字)。
match_field_value	要在 **match_field_name** 字段中查找的值。
table_name	要在其中查找值的表格的名称。输入值必须为字符串(例如引用的文字)。 如果省略了 **table_name**，假定为当前表格。

引用当前表格的参数，不带引号。要引用其他表格，须使用单引号将参数括起来。

限制：

搜索顺序即为加载顺序，除非表格为复杂操作的结果(如联接)，在这种情况下顺序并未很好地定义。**field_name** 及 **match_field_name** 必须为相同表格中的字段，由 **table_name** 指定。

如果未找到匹配值，则返回 NULL。

示例

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下示例。

```
ProductList: Load * Inline [ ProductID|Product|Category|Price 1|AA|1|1 2|BB|1|3 3|CC|2|8
4|DD|3|2 ] (delimiter is '|');   OrderData: Load *, Lookup('Category', 'ProductID', ProductID,
'ProductList') as CategoryID Inline [ InvoiceID|CustomerID|ProductID|Units 1|Astrida|1|8
1|Astrida|2|6 2|Betacab|3|10 3|Divadip|3|5 4|Divadip|4|10 ] (delimiter is '|');   Drop Table
ProductList;
```

创建可视化

在 Qlik Sense 工作表中创建表格可视化。将字段 **ProductID**、**InvoiceID**、**CustomerID**、**Units** 和 **CategoryID** 添加至表格。

结果

结果表

ProductID	InvoiceID	CustomerID	单位:	CategoryID
1	1	Astrida	8	1
2	1	Astrida	6	1
3	2	Betacab	10	2
3	3	Divadip	5	2
4	4	Divadip	10	3

解释

样本数据使用以下格式的 **Lookup()** 函数:

```
Lookup('Category', 'ProductID', ProductID, 'ProductList')
```

首先加载 **ProductList** 表格。

Lookup() 函数用于构建 **OrderData** 表格。它将第三个参数指定为 **ProductID**。这是用于在 **ProductList** 的第二个参数**'ProductID'**中查找值的字段，用单引号括起来表示。

此函数返回“**Category**”的值(**ProductList** 表格中)，然后加载作为 **CategoryID**。

drop 语句用于从数据模型删除 **ProductList** 表格(因为不再需要)，从而保留所得的 **OrderData** 表格:

Lookup() 函数的用法非常灵活，可以用于访问先前加载的所有表格。但是，与 Applymap() 函数相比，它的速度相对较慢。

另请参见:

- *ApplyMap (page 636)*

NoOfRows - 图表函数

NoOfRows() 用于返回表格中当前列段数据的行数。对于位图图表，**NoOfRows()** 用于返回图表的等效垂直表中的行数。

如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

语法:

```
NoOfRows([TOTAL])
```

返回数据类型： 整数

参数：

参数

参数	说明
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

示例：使用 NoOfRows 的图表表达式

示例 - 图表表达式

加载脚本

将以下数据作为数据加载编辑中的内联加载载入，以创建以下图表表达式示例。

```
Temp: LOAD * inline [ Region|SubRegion|RowNo()|NoOfRows() Africa|Eastern Africa|Western Americas|Central Americas|Northern Asia|Eastern Europe|Eastern Europe|Northern Europe|Western Oceania|Australia ] (delimiter is '|');
```

图表表达式

在 Qlik Sense 工作表中创建表可视化，以 **Region** 和 **SubRegion** 为维度。添加 `RowNo( )`、 `NoOfRows()` 和 `NoOfRows(Total)` 为度量

结果

Region	SubRegion	RowNo()	NoOfRows()	NoOfRows (Total)
Africa	Eastern	1	2	9
Africa	Western	2	2	9
Americas	Central	1	2	9
Americas	Northern	2	2	9
Asia	Eastern	1	1	9
Europe	Eastern	1	3	9
Europe	Northern	2	3	9
Eurrope	Western	3	3	9
Oceania	Australia	1	1	9

解释

在本例中，排序顺序是按第一维度 Region 排序的。因此，每个列段由一组具有相同值的区域组成，例如 Africa。

RowNo() 列显示每个列段数据的行号，例如，Africa 地区有两行。然后，再次从 1 开始为下一个列段数据的行进行编号，即 Americas。

NoOfRows() 列统计每个列段数据中的行数，例如，欧洲在列段数据中有三行。

NoOfRows(Total) 列由于 `NoOfRows()` 的参数 `TOTAL` 而忽略维度，并对表中的行进行计数。

如果表格是按第二维度 SubRegion 排序的，则列段将基于该维度，因此每个 SubRegion 的行编号都将更改。

另请参见：

- *RowNo - 图表函数 (page 408)*

Peek

Peek() 用于在表格中返回已经加载行的字段值。可以将行号指定为表格。如果未指定行号，将使用上次加载的记录。

peek() 函数最常用于查找以前加载的表中的相关边界，即特定字段的第一个值或最后一个值。在大多数情况下，该值存储在一个变量中供以后使用，例如，作为 do-while 循环中的一个条件。

语法：

```
Peek(
field_name
[, row_no[, table_name ] ])
```

返回数据类型： 双

参数：

参数

参数	说明
field_name	需要返回值的字段的名称。输入值必须为字符串(例如引用的文字)。
row_no	表格中的行用于指定所需的字段。可以是表达式，但解算结果必须为整数。0 表示第一个记录，1 表示第二个记录，以此类推。负数表示从表格末端开始计算的顺序。-1 表示最后读取的记录。 如果未指定**row_no**，则假定为 -1。
table_name	表格标签不能以冒号结束。如果未指定**table_name**，则假定为当前表格。如果用于 **LOAD** 语句之外或指向另外一个表格，则必须包括 **table_name**。

限制：

该函数只能从已加载的记录中返回值。这意味着在表的第一条记录中，使用 -1 作为 row_no 的调用将返回 NULL。

示例和结果：

示例 1

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
EmployeeDates: Load * Inline [ EmployeeCode|StartDate|EndDate 101|02/11/2010|23/06/2012
102|01/11/2011|30/11/2013 103|02/01/2012| 104|02/01/2012|31/03/2012 105|01/04/2012|31/01/2013
106|02/11/2013| ] (delimiter is '|');       First_last_Employee: Load  EmployeeCode,  Peek
('EmployeeCode',0,'EmployeeDates') As FirstCode,  Peek('EmployeeCode',-1,'EmployeeDates') As
LastCode Resident EmployeeDates;
```

结果表

员工代码	StartDate	EndDate	FirstCode	LastCode
101	02/11/2010	23/06/2012	101	106
102	01/11/2011	30/11/2013	101	106
103	02/01/2012		101	106
104	02/01/2012	31/03/2012	101	106
105	01/04/2012	31/01/2013	101	106
106	02/11/2013		101	106

FirstCode = 101，因为 `Peek('EmployeeCode',0, 'EmployeeDates')` 返回表格 EmployeeDates 的 EmployeeCode 中的第一个值。

LastCode = 106，因为 `Peek('EmployeeCode',-1, 'EmployeeDates')` 返回表格 EmployeeDates 的 EmployeeCode 中的最后一个值。

替代参数 **row_no** 返回表格中其他行的值，如下所示：

`Peek('EmployeeCode',2, 'EmployeeDates')` 用于返回表格中的第三个值 103(作为 FirstCode)。

但是，请注意，如果在这些示例中没有将表格指定为第三个参数 **table_name**，此函数引用当前表格(在此例中，为内部表格)。

示例 2

如果要访问表中更深层的数据，需要分两步进行：首先，将整个表加载到临时表中，然后在使用 **Peek()** 时对其重新排序。

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
T1: LOAD * inline [ ID|Value 1|3 1|4 1|6 3|7 3|8 2|1 2|11 5|2 5|78 5|13 ] (delimiter is '|');
T2:  LOAD *, IF(ID=Peek('ID'), Peek('List')&','&Value,Value) AS List RESIDENT T1 ORDER BY ID
ASC; DROP TABLE T1;
```

Create a table in a sheet in your app with **ID**, **List**, and **Value** as the dimensions.

结果表

ID	列表	值
1	3,4	4
1	3,4,6	6
1	3	3
2	1,11	11
2	1	1
3	7,8	8
3	7	7
5	2,78	78
5	2,78,13	13
5	2	2

IF() 语句是根据临时表格 T1 构建。
`Peek('ID')` 引用当前表格 T2 的上一行中的字段 ID。
`Peek('List')` 引用当前表格 T2 的上一行中的字段 List，目前正在构建要解算的表达式。

如下运算语句：
如果 ID 的当前值与 ID 的上一个值相同，则写入 Peek('List') 的值串联 Value 的当前值。否则，只写入 Value 的当前值。

如果 Peek('List') 已经包含串联结果，则会将 Peek('List') 的新结果串联至其当前值。

*注意，**Order by** 子句。该子句用于指定表格的排序方式(按 ID 进行升序排序)。如果没有使用此子句，Peek() 函数将使用内部表格拥有的任意排序方式，这可能会导致产生不可预测的结果。*

示例 3

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
Amounts: Load  Date#(Month,'YYYY-MM') as Month, Amount, Peek(Amount) as AmountMonthBefore
Inline [Month,Amount 2022-01,2 2022-02,3 2022-03,7 2022-04,9 2022-05,4 2022-06,1];
```

结果表

金额	AmountMonthBefore	月
1	4	2022-06
2	-	2022-01
3	2	2022-02
4	9	2022-05
7	3	2022-03
9	7	2022-04

字段 AmountMonthBefore 将保存上个月的金额。

这里省略了 row_no 和 table_name 参数，因此使用默认值。在本例中，以下三个函数调用是等效的：

- Peek(Amount)
- Peek(Amount,-1)
- Peek(Amount,-1,'Amounts')

将 -1 用作 row_no 并不意味着将使用前一行中的值。通过替换该值，可以获取表中其他行的值：

Peek(Amount,2) 用于返回表格中的第三个值：7。

示例 4：

数据需要正确排序才能得到正确的结果，但遗憾的是，情况并非总是如此。此外，Peek() 函数不能用于引用尚未加载的数据。通过使用临时表并对数据进行多次传递，可以避免此类问题。

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
tmp1Amounts: Load * Inline [Month,Product,Amount 2022-01,B,3 2022-01,A,8 2022-02,B,4 2022-02,A,6 2022-03,B,1 2022-03,A,6 2022-04,A,5 2022-04,B,5 2022-05,B,6 2022-05,A,7 2022-06,A,4 2022-06,B,8];  tmp2Amounts: Load *, If(Product=Peek(Product),Peek(Amount)) as AmountMonthBefore Resident tmp1Amounts Order By Product, Month Asc;  Drop Table tmp1Amounts; Amounts: Load *, If(Product=Peek(Product),Peek(Amount)) as AmountMonthAfter Resident tmp2Amounts Order By Product, Month Desc;  Drop Table tmp2Amounts;
```

解释

初始表是按月份排序的，这意味着 peek() 函数在很多情况下会返回错误产品的金额。因此，该表需要重新排序。这是通过运行第二次数据传递并创建一个新表来完成的。注意，Order by 子句。它先按产品将记录排序，然后按月份升序排序。

需要 If() 函数，因为如果前一行包含同一产品但属于上一个月的数据，则只应计算 AmountMonthBefore。通过将当前行的产品与前一行的产品进行比较，可以验证此条件。

创建第二个表时，使用 Drop 创建第二个表时，使用Drop Table 语句删除第一个表。

最后，对数据进行第三次遍历，但现在月份的排序是相反的。这样，也可以计算 AmountMonthAfter。

Order by子句指定表格的排序方式;如果没有使用这些子句, Peek() 函数将使用内部表格拥有的任意排序方式, 这可能会导致产生不可预测的结果。

结果

结果表

月	产品	金额	AmountMonthBefore	AmountMonthAfter
2022-01	A	8	-	6
2022-02	B	3	-	4
2022-03	A	6	8	6
2022-04	B	4	3	1
2022-05	A	6	6	5
2022-06	B	1	4	5
2022-01	A	5	6	7
2022-02	B	5	1	6
2022-03	A	7	5	4
2022-04	B	6	5	8
2022-05	A	4	7	-
2022-06	B	8	6	-

示例 5

将示例脚本添加到应用程序并运行。要查看结果, 将结果列中列出的字段添加到应用程序中的工作表。

```
T1: Load * inline [ Quarter, Value 2003q1, 10000 2003q1, 25000 2003q1, 30000 2003q2, 1250 2003q2, 55000 2003q2, 76200 2003q3, 9240 2003q3, 33150 2003q3, 89450 2003q4, 1000 2003q4, 3000 2003q4, 5000 2004q1, 1000 2004q1, 1250 2004q1, 3000 2004q2, 5000 2004q2, 9240 2004q2, 10000 2004q3, 25000 2004q3, 30000 2004q3, 33150 2004q4, 55000 2004q4, 76200 2004q4, 89450 ];  T2: Load *, rangesum(SumVal,peek('AccSumVal')) as AccSumVal; Load Quarter, sum(Value) as SumVal resident T1 group by Quarter;
```

结果

结果表

季度	SumVal	AccSumVal
2003q1	65000	65000
2003q2	132450	197450
2003q3	131840	329290

季度	SumVal	AccSumVal
2003q4	9000	338290
2004q1	5250	343540
2004q2	24240	367780
2004q3	88150	455930
2004q4	220650	676580

解释

LOAD 语句 **Load *, rangesum(SumVal,peek('AccSumVal')) as AccSumVal** 包括一个递归调用，其中以前的值被添加到当前值。此操作用于计算脚本中值的累积。

另请参见：

Previous

Previous() 用于查找使用因 **where** 子句而未丢弃的以前输入记录的数据的 **expr** 表达式的值。在内部表格的首个记录中，此函数将返回 NULL 值。

语法：

```
Previous(expr)
```

返回数据类型： 双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。 表达式可以包含嵌套的 **previous()** 函数以访问能够进一步回滚的记录。数据直接从输入源获取，使其也可引用尚未载入 Qlik Sense 的字段，也就是说即使它们存储在相关的数据库中也可以引用。

限制：

在内部表格的首个记录中，此函数返回 NULL 值。

示例：

在加载脚本中输入以下内容

```
Sales2013:
Load *, (Sales - Previous(Sales) )as Increase Inline [
Month|Sales
1|12
```

```
2|13
3|15
4|17
5|21
6|21
7|22
8|23
9|32
10|35
11|40
12|41
] (delimiter is '|');
```

通过在 **Load** 语句中使用 **Previous()** 函数，我们可以将 Sales 的当前值与上一个值进行比较，并在第三个字段 Increase 中使用该值。

结果表

月	销售额值	增大
1	12	-
2	13	1
3	15	2
4	17	2
5	21	4
6	21	0
7	22	1
8	23	1
9	32	9
10	35	3
11	40	5
12	41	1

Top - 图表函数

Top() 用于评估表格中列段数据第一(顶部)行的表达式。要计算的行取决于 **offset** 值，如果存在，则默认计算顶行。对于除表格以外的图表，**Top()** 用于计算图表的等效垂直表中当前列的第一行的值。

语法：

```
Top([TOTAL] expr [ , offset [,count ]])
```

返回数据类型：双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
offset	指定 **offset** n 大于 1 后，将表达式评估向下移到顶行下面的 n 行。 指定负偏移量数值后，使 **Top** 函数效果类似于具有相应正偏移量数值的 **Bottom** 函数。
count	通过指定第三个参数 **count** 大于 1，函数将返回一连串 **count** 值，每个值对应当前列段数据的最后一个 **count** 行中的一行。此时，可以将该函数用作任何特殊范围函数的参数。 *范围函数 (page 643)*
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算，不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度，或者如果已指定 TOTAL 限定符，则计算整个表格中的表达式。

如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

限制：

递归调用将返回 NULL 值。

示例和结果：

示例：1

在此示例中显示的表格的屏幕截图中，表格可视化内容通过维度 **Customer** 和以下度量进行创建：`Sum(Sales)` 和 `Top(Sum(Sales))`。

全部行的 **Top(Sum(Sales))** 列均返回 587，因为此值是顶行值：**Astrida**。

此表格还显示了更复杂的度量：根据 `Sum(Sales)+Top(Sum(Sales))` 创建的一个值以及标有 **Top offset 3** 的一个值，后者使用表达式 `Sum(Sales)+Top(Sum(Sales), 3)` 创建，且具有设置为 **offset** 的参数 3。它将当前行的 **Sum(Sales)** 值添加到从顶行开始第三行中的值中，即，当前行加上 **Canutility** 的值。

示例 1

Top and Bottom

Customer	Sum(Sales)	Top(Sum(Sales))	Sum(Sales)+Top(Sum(Sales))	Top offset 3
Totals	2566	587	3153	3249
Astrida	587	587	1174	1270
Betacab	539	587	1126	1222
Canutility	683	587	1270	1366
Divadip	757	587	1344	1440

示例：2

在此示例中显示的表格的屏幕截图中，已将更多维度添加到可视化内容中：**Month** 和 **Product**。对于包含多个维度的图表，表达式（包含 **Above**、**Below**、**Top** 和 **Bottom** 函数）的结果取决于 Qlik Sense 对列维度进行排序的顺序。Qlik Sense 根据最后排序的维度得出的列段数据计数函数的值。可以在**排序**下的属性面板中控制列排序顺序，并且列不一定按顺序显示在表格中。

示例 2 的第二个表格。First value 度量的 Top 的值基于 Month (Jan)。

Customer	Product	Month	Sum(Sales)	First value
			2566	-
Astrida	AA	Jan	46	46
Astrida	AA	Feb	60	46
Astrida	AA	Mar	70	46
Astrida	AA	Apr	13	46
Astrida	AA	May	78	46
Astrida	AA	Jun	20	46
Astrida	AA	Jul	45	46
Astrida	AA	Aug	65	46
Astrida	AA	Sep	78	46
Astrida	AA	Oct	12	46
Astrida	AA	Nov	78	46
Astrida	AA	Dec	22	46

示例 2 的第二个表格。First value 度量的 Top 的值基于 Product(Astrida 的 AA)。

Customer	Product	Month	Sum(Sales)	First value
			2566	-
Astrida	AA	Jan	46	46
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	60
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	70
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	13
Astrida	BB	Apr	13	13

有关更多信息，请参阅 **Above** 函数中的示例：2。

示例 3

示例：3	**结果**
Top 函数可用作范围函数的输入。例如：`RangeAvg (Top(Sum(Sales),1,3))`。	在 **Top()** 函数的参数中，将 offset 设置为 1，并将 count 设置为 3。函数在列段数据中底行下方的行开始的三行上查找表达式 **Sum(Sales)** 的结果（因为 offset=1），并且列段数据（其中有一行）下方有两行。这三个值用作 RangeAvg() 函数的输入，用于查找所提供的数字范围中的平均值。 以 **Customer** 为维度的表格为 RangeAvg() 表达式提供了以下结果。
	Astrida 603 Betacab 603 Canutility 603 Divadip： 603

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
```

```
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

另请参见：

- *Bottom - 图表函数 (page 603)*
- *Above - 图表函数 (page 596)*
- *Sum - 图表函数 (page 212)*
- *RangeAvg (page 645)*
- *范围函数 (page 643)*

SecondaryDimensionality- 图表函数

SecondaryDimensionality() 返回包含非聚合函数内容的维度透视表行数，即不包含部分总和或折叠聚合。此函数等同于水平透视表维度的 **dimensionality()** 函数。

语法：

```
SecondaryDimensionality( )
```

返回数据类型： 整数

限制：

除非用于透视表，否则 **SecondaryDimensionality** 函数始终返回 0。

After - 图表函数

After() 用于返回使用透视表的维度值评估的表达式值，因为维度值显示在透视表的行段内当前列之后的列。

语法：

```
after([TOTAL] expr [, offset [, count ]])
```

在除透视表之外的所有图表类型中，此函数会返回 NULL。

参数：

参数	说明
expr	表达式或字段包含要度量的数据。
offset	指定 **offset** n(大于 1) 会将表达式评估从当前行开始向右移动 n 行。 指定 0 偏移量可以计算当前行上的表达式的值。 指定负偏移量数值后，使 **After** 函数效果类似于具有相应正偏移量数值的 **Before** 函数。
count	通过指定第三个参数 **count**(大于 1)，函数将返回一连串值，每个值对应一个从原始单元格开始向右计数的 **count** 表格行。
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

行段的最后一列会返回 NULL 值，因为其后没有列。

如果透视表有多个水平维度，则当前行片断将只包括值与所有维度行中当前列相同的列，除显示字段排序间上一次水平维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义.

示例：

```
after( sum( Sales ))
after( sum( Sales ), 2 )
after( total sum( Sales ))
```

`rangeavg (after(sum(x),1,3))` 用于返回当前列右边三列内评估的 **sum(x)** 函数的三个结果的平均值。

Before - 图表函数

Before() 返回使用透视表的维度值评估的表达式，因为维度值显示在透视表的行段内当前列之前的列。

语法：

```
before([TOTAL] expr [, offset [, count]])
```

在除透视表之外的所有图表类型中，此函数会返回 NULL。

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。

参数	说明
offset	指定 **offset** n(大于 1) 会将表达式评估从当前行开始向左移动 n 行。 指定 0 偏移量可以计算当前行上的表达式的值。 指定负偏移量数值后，使 **Before** 函数效果类似于具有相应正偏移量数值的 **After** 函数。
count	通过指定第三个参数 **count**(大于 1)，函数将返回一连串值，每个值对应一个从原始单元格开始向左计数的 **count** 表格行。
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

行段的第一列会返回 NULL 值，因为其前没有列。

如果透视表有多个水平维度，则当前行片断将只包括值与所有维度行中当前列相同的列，除显示字段排序间上一次水平维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义.

示例：

```
before( sum( Sales ))
before( sum( Sales ), 2 )
before( total sum( Sales ))
```

`rangeavg (before(sum(x),1,3))` 用于返回当前列左边三列内评估的 **sum(x)** 函数的三个结果的平均值。

First - 图表函数

First() 用于返回使用透视表的维度值评估的表达式值，因为维度值显示在透视表的当前行段的第一列。在除透视表之外的所有图表类型中，此函数会返回 NULL。

语法：

```
first([TOTAL] expr [, offset [, count]])
```

参数：

参数

参数	说明
expression	表达式或字段包含要度量的数据。
offset	指定 **offset** n(大于 1) 会将表达式评估从当前行开始向右移动 n 行。 指定 0 偏移量可以计算当前行上的表达式的值。 指定负偏移量数值后，使 **First** 函数效果类似于具有相应正偏移量数值的 **Last** 函数。

参数	说明
count	通过指定第三个参数 **count**(大于 1) , 函数将返回一连串值 , 每个值对应一个从原始单元格开始向右计数的 **count** 表格行。
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数 , 则当前列段数据总是与整列相等。

如果透视表有多个水平维度 , 则当前行片断将只包括值与所有维度行中当前列相同的列 , 除显示字段排序间上一次水平维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义.

示例 :

```
first( sum( Sales ))
first( sum( Sales ), 2 )
first( total sum( Sales )
```

`rangeavg (first(sum(x),1,5))`返回当前行段最左边五列内评估的 **sum(x)** 函数的结果的平均值。

Last - 图表函数

Last() 返回使用透视表的维度值评估的表达式值 , 因为维度值显示在透视表的当前行段的最后一列。在除透视表之外的所有图表类型中 , 此函数会返回 NULL。

语法 :

```
last([TOTAL] expr [, offset [, count]])
```

参数 :

参数

参数	说明
expr	表达式或字段包含要度量的数据。
offset	指定 **offset** n(大于 1) 会将表达式评估从当前行开始向左移动 n 行。 指定 0 偏移量可以计算当前行上的表达式的值。 指定负偏移量数值后 , 使 **First** 函数效果类似于具有相应正偏移量数值的 **Last** 函数。
count	通过指定第三个参数 **count**(大于 1) , 函数将返回一连串值 , 每个值对应一个从原始单元格开始向左计数的 **count** 表格行。
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数 , 则当前列段数据总是与整列相等。

如果透视表有多个水平维度 , 则当前行片断将只包括值与所有维度行中当前列相同的列 , 除显示字段排序间上一次水平维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义.

示例：

```
last( sum( Sales ))
last( sum( Sales ), 2 )
last( total sum( Sales )
```

`rangeavg (last(sum(x),1,5))` 用于返回当前行段最右边五列内评估的 **sum(x)** 函数的结果的平均值。

ColumnNo - 图表函数

ColumnNo() 用于返回透视表的当前行段中的当前列数。第一列是数字 1。

语法：

```
ColumnNo([total])
```

参数：

参数

参数	说明
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

如果透视表有多个水平维度，则当前行片断将只包括值与所有维度行中当前列相同的列，除显示字段排序间上一次水平维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义.

示例：

```
if( ColumnNo( )=1, 0, sum( Sales ) / before( sum( Sales )))
```

NoOfColumns - 图表函数

NoOfColumns() 用于返回透视表的当前行段中的列数。

语法：

```
NoOfColumns([total])
```

参数：

参数

参数	说明
TOTAL	如果表格是单维度或如果将 **TOTAL** 限定符用作参数，则当前列段数据总是与整列相等。

如果透视表有多个水平维度，则当前行片断将只包括值与所有维度行中当前列相同的列，除显示字段排序间上一次维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义.

示例：

```
if( ColumnNo( )=NoOfColumns( ), 0, after( sum( Sales )))
```

5.17 逻辑函数

本部分介绍处理逻辑运算的函数。所有函数均可用于数据加载脚本和图表表达式。

IsNum

返回 -1 (True)(如果将表达式解释为数字) , 否则返回 0 (False)。

```
IsNum( expr )
```

IsText

返回 -1 (True)(如果表达式显示为文本) , 否则返回 0 (False)。

```
IsText( expr )
```

*如果表达式为 NULL，**IsNum** 和 **IsText** 均返回 0。*

示例：

以下示例加载含有混合文本和数值的内联表，并添加两个字段用于检查相应文本值的值是否为数值。

```
Load *, IsNum(Value), IsText(Value)
Inline [
Value
23
Green
Blue
12
33Red];
```

最终生成的表格如下所示：

Resulting table

Value	IsNum(Value)	IsText(Value)
23	-1	0
Green	0	-1
Blue	0	-1
12	-1	0
33Red	0	-1

5.18 映射函数

本节介绍用于处理映射表格的函数。映射表格可用于在脚本执行期间替换字段值或字段名称。

映射函数只能用于数据加载脚本。

映射函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

ApplyMap

ApplyMap 脚本函数用于将表达式输出映射至先前加载的映射表。

```
ApplyMap ('mapname', expr [ , defaultexpr ] )
```

MapSubstring

MapSubstring 脚本函数用于将任何表达式的一部分映射至先前加载的映射表。映射区分大小写且不重复，子字符串会从左至右映射。

```
MapSubstring ('mapname', expr)
```

ApplyMap

ApplyMap 脚本函数用于将表达式输出映射至先前加载的映射表。

语法：

```
ApplyMap('map_name', expression [ , default_mapping ] )
```

返回数据类型： 双

参数：

参数

参数	说明
map_name	以前通过 **mapping load** 或 **mapping select** 语句创建的映射表的名称。该名称必须用单直引号引起来。 *如果您在宏扩展的变量中使用该函数并引用不存在的映射表格，函数调用会失败并且不会创建字段。*
expression	表达式，即将被映射的结果。
default_mapping	如果已指定，即如果映射表不包含与 expression 相匹配的值，则可将此值用作默认值。如果未指定，则 expression 的值将原样返回。

ApplyMap 的输出字段不应有和其输入字段中的一个相同的名称。这可能导致意外结果。不使用的示例:`ApplyMap('Map', A) as A`。

示例:

在此例中,我们加载了销售人员和国家代码(表示销售人员所居住的国家)的列表。我们使用表格将国家代码映射到国家,以便将国家代码替换为国家名称。在映射表中仅定义了三个国家,其他国家代码已映射到'Rest of the world'。

```
// Load mapping table of country codes:
map1:
mapping LOAD *
Inline [
CCode, Country
Sw, Sweden
Dk, Denmark
No, Norway
] ;

// Load list of salesmen, mapping country code to country
// If the country code is not in the mapping table, put Rest of the world
Salespersons:
LOAD *,
ApplyMap('map1', CCode,'Rest of the world') As Country
Inline [
CCode, Salesperson
Sw, John
Sw, Mary
Sw, Per
Dk, Preben
Dk, Olle
No, Ole
Sf, Risttu
] ;

// We don't need the CCode anymore
Drop Field 'CCode';
```

最终生成的表格(销售人员)如下所示:

Resulting table

Salesperson	Country
John	Sweden
Mary	Sweden
Per	Sweden
Preben	Denmark
Olle	Denmark

Salesperson	Country
Ole	Norway
Risttu	Rest of the world

MapSubstring

MapSubstring 脚本函数用于将任何表达式的一部分映射至先前加载的映射表。映射区分大小写且不重复，子字符串会从左至右映射。

语法：

```
MapSubstring('map_name', expression)
```

返回数据类型： 字符串

参数：

参数

参数	说明
map_name	先前在 **mapping load** 或 **mapping select** 语句中读取的映射表的名称。名称必须必须用直的单引号括起来。 *如果您在宏扩展的变量中使用该函数并引用不存在的映射表格，函数调用会失败并且不会创建字段。*
expression	结果被子字符串映射的表达式。

示例：

在此例中，我们加载产品型号的列表。每一种产品型号都拥有一组使用复合代码描述的属性。使用带有 MapSubstring 的映射表，我们可以展开属性代码的描述。

```
map2:
mapping LOAD *
Inline [
AttCode, Attribute
R, Red
Y, Yellow
B, Blue
C, Cotton
P, Polyester
S, Small
M, Medium
L, Large
] ;

Productmodels:
```

```
LOAD *,
MapSubString('map2', AttCode) as Description
Inline [
Model, AttCode
Twixie, R C S
Boomer, B P L
Raven, Y P M
Seedling, R C L
SeedlingPlus, R C L with hood
Younger, B C with patch
MultiStripe, R Y B C S/M/L
] ;
// We don't need the AttCode anymore
Drop Field 'AttCode';
```

最终生成的表格如下所示：

Resulting table

Model	Description
Twixie	Red Cotton Small
Boomer	Blue Polyester Large
Raven	Yellow Polyester Medium
Seedling	Red Cotton Large
SeedlingPlus	Red Cotton Large with hood
Younger	Blue Cotton with patch
MultiStripe	Red Yellow Blue Cotton Small/Medium/Large

5.19 数学函数

本节介绍执行数学常数和布尔值计算的函数。这些函数没有任何参数，但是它后面的括号不能省略。

所有函数均可用于数据加载脚本和图表表达式。

e

该函数返回自然对数 **e** (2.71828...) 的基数。

```
e( )
```

false

返回一个对偶值，文本值'False'和数值“0”，可以用作表达式的逻辑假。

```
false( )
```

pi

该函数返回 π (3.14159...) 的值。

```
pi( )
```

rand

该函数返回 0 与 1 之间的随机数字。并且，该函数也可用于创建样本数据。

```
rand( )
```

示例：

以下示例脚本用于创建拥有 1000 条记录且包含随机选择的大写字符的表格，即范围为 65 至 91 (65+26) 的字符。

```
Load
     Chr( Floor(rand() * 26) + 65) as UCaseChar,
     RecNo() as ID
     Autogenerate 1000;
```

true

返回一个对偶值，文本值'True'和数值“-1”，可以用作表达式的逻辑真。

```
true( )
```

5.20 NULL 函数

本节介绍用于返回或检测 NULL 值的函数。

所有函数均可用于数据加载脚本和图表表达式。

NULL 函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

EmptyIsNull

EmptyIsNull 函数将空字符串转换成 NULL。因此，如果参数是空字符串，则返回 NULL，否则返回参数。

```
EmptyIsNull (expr )
```

IsNull

IsNull 函数用于检验表达式的值是否为 NULL，如果是，则返回 -1 (True)，否则返回 0 (False)。

```
IsNull (expr )
```

Null

Null 函数用于返回 NULL 值。

```
NULL( )
```

EmptyIsNull

EmptyIsNull 函数将空字符串转换成 NULL。因此，如果参数是空字符串，则返回 NULL，否则返回参数。

语法：

```
EmptyIsNull(exp )
```

示例和结果：

脚本示例

示例	结果
EmptyIsNull(AdditionalComments)	此表达式将以 null 形式返回 *AdditionalComments* 字段的任何空字符串值，而不是空字符串。返回非空字符串和数字。
EmptyIsNull(PurgeChar (PhoneNumber, ' -()'))	此表达式将从 *PhoneNumber* 字段中删除任何破折号、空格和括号。如果没有剩余字符，则 EmptyIsNull 函数将空字符串返回为 null；空电话号码与没有电话号码相同。

IsNull

IsNull 函数用于检验表达式的值是否为 NULL，如果是，则返回 -1 (True)，否则返回 0 (False)。

语法：

```
IsNull(expr )
```

*不能将长度为零的字符串看作是 NULL，否则将会导致 **IsNull** 函数返回 False。*

示例：数据加载脚本

在此例中，已加载包含四行的内联表，其中前面三行不包含任何内容，即 Value 列中的 - 或 'NULL'。我们使用 **Null** 函数将这些值转换为带有中间前置 **LOAD** 的真正的 NULL 值呈现形式。

第一个前置 **LOAD** 用于添加一个字段，通过使用 **IsNull** 函数来检查值是否为 NULL。

```
NullsDetectedAndConverted:

LOAD *,
If(IsNull(ValueNullConv), 'T', 'F') as IsItNull;

LOAD *,
If(len(trim(Value))= 0 or Value='NULL' or Value='-', Null(), Value ) as ValueNullConv;

LOAD * Inline
[ID, Value
0,
1,NULL
```

```
2,-
3,Value];
```

以下是最终生成的表格。在 ValueNullConv 列中，NULL 值用 - 表示。

Resulting table

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T
2	-	-	T
3	Value	Value	F

NULL

Null 函数用于返回 NULL 值。

语法：

```
Null( )
```

示例：数据加载脚本

在此例中，已加载包含四行的内联表，其中前面三行不包含任何内容，即 Value 列中的 - 或 'NULL'。我们想要将这些值转移为真正的 NULL 值呈现形式。

中间前置 **LOAD** 使用 **Null** 函数执行转换。

第一个前置 **LOAD** 用于添加一个字段来检查该值是否为 NULL，在此例中仅供说明。

```
NullsDetectedAndConverted:

LOAD *,
If(IsNull(ValueNullConv), 'T', 'F') as IsItNull;

LOAD *,
If(len(trim(Value))= 0 or Value='NULL' or Value='-', Null(), Value ) as ValueNullConv;

LOAD * Inline
[ID, Value
0,
1,NULL
2,-
3,Value];
```

以下是最终生成的表格。在 ValueNullConv 列中，NULL 值用 - 表示。

Resulting table

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T

ID	Value	ValueNullConv	IsItNull
2	-	-	T
3	Value	Value	F

5.21 范围函数

范围函数是采用值数组并产生单个值作为结果的函数。所有范围函数都可用于数据加载脚本和图表表达式。

例如，在可视化中，范围函数可用于计算内部记录函数的单个值。在数据加载脚本中，范围函数可用于计算内部表中值阵列的单个值。

*范围函数可替代以下一般数字函数：**numsum**、**numavg**、**numcount**、**nummin** 和 **nummax** (现在应被视为已过时)。*

基本范围函数

RangeMax

RangeMax() 用于返回在表达式或字段内找到的最高数值。

```
RangeMax (first_expr[, Expression])
```

RangeMaxString

RangeMaxString() 用于以文本排序顺序返回在表达式或字段中找到的最后一个值。

```
RangeMaxString (first_expr[, Expression])
```

RangeMin

RangeMin() 用于返回在表达式或字段内找到的最低数值。

```
RangeMin (first_expr[, Expression])
```

RangeMinString

RangeMinString() 用于以文本排序顺序返回在表达式或字段中找到的第一个值。

```
RangeMinString (first_expr[, Expression])
```

RangeMode

RangeMode() 用于查找表达式或字段中最常出现的值(即模式值)。

```
RangeMode (first_expr[, Expression])
```

RangeOnly

RangeOnly() 是一个对偶函数，用于返回一个值(如果表达式计算为一个独特的值)。如果不是一个独特的值，则返回 **NULL** 值。

```
RangeOnly (first_expr[, Expression])
```

RangeSum
RangeSum() 返回一系列值的总和。所有非数字值均被视为0。

```
RangeSum (first_expr[, Expression])
```

计数器范围函数

RangeCount
RangeCount() 用于返回表达式或字段中文本值和数字值的数量。

```
RangeCount (first_expr[, Expression])
```

RangeMissingCount
RangeMissingCount() 用于返回表达式或字段中非数字值(包括 NULL)的数量。

```
RangeMissingCount (first_expr[, Expression])
```

RangeNullCount
RangeNullCount() 用于查找表达式或字段中 NULL 值的数量。

```
RangeNullCount (first_expr[, Expression])
```

RangeNumericCount
RangeNumericCount() 用于查找表达式或字段中数字值的数量。

```
RangeNumericCount (first_expr[, Expression])
```

RangeTextCount
RangeTextCount() 用于返回表达式或字段中文本值的数量。

```
RangeTextCount (first_expr[, Expression])
```

统计范围函数

RangeAvg
RangeAvg() 用于返回范围的平均值。可以将一系列值或表达式导入该函数。

```
RangeAvg (first_expr[, Expression])
```

RangeCorrel
RangeCorrel() 用于返回两组数据的相关系数。相关系数是数据集之间关系的度量。

```
RangeCorrel (x_values , y_values[, Expression])
```

RangeFractile
RangeFractile() 用于返回与数值系列的第 n 个**fractile**(位数)对应的值。

```
RangeFractile (fractile, first_expr[ ,Expression])
```

RangeKurtosis
RangeKurtosis() 用于返回与数值范围的峰度对应的值。

```
RangeKurtosis (first_expr[, Expression])
```

RangeSkew

RangeSkew() 用于返回与数值范围的偏度对应的值。

```
RangeSkew (first_expr[, Expression])
```

RangeStdev

RangeStdev() 用于查找数字系列的标准偏差。

```
RangeStdev (expr1[, Expression])
```

财务范围函数

RangeIRR

RangeIRR() 用于返回按输入值表示的一系列现金流的内部回报率。

```
RangeIRR (value[, value][, Expression])
```

RangeNPV

RangeNPV() 用于返回基于折扣率和一系列未来定期付款(负值)和收入(正值)的投资的净现值。结果拥有一个 **money** 的默认数字格式。

```
RangeNPV (discount_rate, value[, value][, Expression])
```

RangeXIRR

RangeXIRR() 用于返回现金流计划表的内部回报率(不必是周期性的)。要计算一系列周期性现金流的内部回报率,请使用 **RangeIRR** 函数。

```
RangeXIRR (values, dates[, Expression])
```

RangeXNPV

RangeXNPV() 用于返回现金流计划表的净现值(不必是周期性的)。结果默认采用货币数字格式。要计算一系列周期性现金流的净现值,请使用 **RangeNPV** 函数。

```
RangeXNPV (discount_rate, values, dates[, Expression])
```

另请参见:

内部记录函数 (page 593)

RangeAvg

RangeAvg() 用于返回范围的平均值。可以将一系列值或表达式导入该函数。

语法:

```
RangeAvg(first_expr[, Expression])
```

返回数据类型: 数字

参数:

该函数的参数可能包含内部记录函数,并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

如果找不到任何数值，则返回 NULL 值。

示例和结果：

脚本示例

示例	结果
`RangeAvg (1,2,4)`	返回 2.33333333
`RangeAvg (1,'xyz')`	返回 1
`RangeAvg (null( ), 'abc')`	返回 NULL

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
RangeTab3:
LOAD recno() as RangeID, RangeAvg(Field1,Field2,Field3) as MyRangeAvg INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

结果列表显示了为表格中的每条记录返回的 MyRangeAvg 值。

结果表

RangeID	MyRangeAvg
1	7
2	4
3	6
4	12.666
5	6.333
6	5

带有表达式的示例：

```
RangeAvg (Above(MyField),0,3))
```

返回当前行与当前行上两行中计算的三个 **MyField** 值范围结果的滑动平均值。通过指定第三个参数作为 3，**Above()** 函数会返回三个值，如果上方有足够的行，会将其作为 **RangeAvg()** 函数的输入。

示例中所使用的数据：

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeAvg (Above(MyField,0,3))	Comments
10	10	因为这是顶行，该范围仅包含一个值。
2	6	此行上方只有一行，因此范围为：10,2.
8	6.6666666667	等于 RangeAvg(10,2,8)
18	9.333333333	-
5	10. 333333333	-
9	10.6666666667	-

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

另请参见：

- *Avg - 图表函数 (page 246)*
- *Count - 图表函数 (page 216)*

RangeCorrel

RangeCorrel() 用于返回两组数据的相关系数。相关系数是数据集之间关系的度量。

语法：

```
RangeCorrel(x_value , y_value[, Expression])
```

返回数据类型： 数字

数据系列应作为 (x,y) 对输入。例如，评估两个数据系列（阵列 1 和阵列 2，其中阵列 1 = 2,6,9；阵列 2 = 3,8,4）时，将写入 `RangeCorrel (2,3,6,8,9,4)`，返回 0.269。

参数：

参数

参数	说明
x-value, y-value	每个值均表示由内部记录函数和第三可选参数返回的单个值或一系列值。每个值或每一系列值都必须对应单个 **x-value** 或一系列 **y-values**。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

计算此函数至少需要两对坐标。

文本值，NULL 值和缺失值都返回 NULL。

示例和结果：

函数示例

示例	结果
`RangeCorrel (2,3,6,8,9,4,8,5)`	返回 0.2492。此函数可加载到脚本中，或添加到表达式编辑器的可视化中。

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
RangeList:
Load * Inline [
ID1|x1|y1|x2|y2|x3|y3|x4|y4|x5|y5|x6|y6
01|46|60|70|13|78|20|45|65|78|12|78|22
02|65|56|22|79|12|56|45|24|32|78|55|15
03|77|68|34|91|24|68|57|36|44|90|67|27
04|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');

XY:
LOAD recno() as RangeID, * Inline [
X|Y
2|3
6|8
9|4
8|5
](delimiter is '|');
```

在 ID1 作为维度和度量的表格中 RangeCorrel(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6))，对于 ID1 值中的每个，**RangeCorrel()** 函数在六个 x,y 对的范围内找到 **Correl** 的值。

结果表

ID1	MyRangeCorrel
01	-0.9517
02	-0.5209
03	-0.5209
04	-0.1599

示例：

```
XY:
LOAD recno() as RangeID, * Inline [
X|Y
2|3
6|8
9|4
8|5
](delimiter is '|');
```

在 RangeID 作为维度和度量的表格中 RangeCorrel(Below(X,0,4,BelowY,0,4)), **RangeCorrel()** 函数使用 **Below()** 函数的结果，这是因为第三参数 (count) 设置为 4，从加载的表 XY 生成一系列四个 x-y 值。

结果表

RangeID	MyRangeCorrel2
01	0.2492
02	-0.9959
03	-1.0000
04	-

RangeID 01 的值与手动输入 RangeCorrel(2,3,6,8,9,4,8,5) 的值相同。对于 RangeID 的其他值，Below() 函数生成的系列为：(6,8,9,4,8,5)、(9,4,8,5) 和 (8,5)，其中最后一个产生空结果。

另请参见：

Correl - 图表函数 (page 249)

RangeCount

RangeCount() 用于返回表达式或字段中文本值和数字值的数量。

语法：

```
RangeCount(first_expr[, Expression])
```

返回数据类型：整数

参数：

该函数的参数可能包含内部记录函数，并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要计数的数据。
Expression	可选表达式或字段包含要计数的数据范围。

限制：

不对 NULL 值计数。

示例和结果：

函数示例

示例	结果
`RangeCount (1,2,4)`	返回 3
`RangeCount (2,'xyz')`	返回 2
`RangeCount (null( ))`	返回 0
`RangeCount (2,'xyz', null())`	返回 2

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
RangeTab3:
LOAD recno() as RangeID, RangeCount(Field1,Field2,Field3) as MyRangeCount INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

结果列表显示了为表格中的每条记录返回的 MyRangeCount 值。

结果表

RangeID	MyRangeCount
1	3
2	3
3	3
4	3
5	3
6	3

带有表达式的示例：

```
RangeCount (Above(MyField,1,3))
```

返回 **MyField** 的三个结果中包含的值的数量。通过指定 **Above()** 函数的第一个参数作为 1，并指定第二个参数作为 3，它会返回当前行上方前三个字段中的值，如果拥有足够的行，会将其作为 **RangeCount()** 函数的输入。

示例中所使用的数据：

样本数据

MyField	RangeCount(Above(MyField,1,3))
10	0
2	1
8	2
18	3
5	3
9	3

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

另请参见：

- *Count - 图表函数 (page 216)*

RangeFractile

RangeFractile() 用于返回与数值系列的第 n 个**fractile**(位数) 对应的值。

RangeFractile() 在计算分位数时会使用最按近排行之间的线性插值。

语法:

```
RangeFractile(fractile, first_expr[, Expression])
```

返回数据类型: 数字

参数:

该函数的参数可能包含内部记录函数, 并在其内部返回一系列值。

参数

参数	说明
fractile	可以计算与分位数(表示为分数的位数) 对应的介于 0 和 1 之间的数字。
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

示例和结果:

函数示例

示例	结果
RangeFractile (0.24,1,2,4,6)	返回 1.72
RangeFractile(0.5,1,2,3,4,6)	返回 3
RangeFractile (0.5,1,2,5,6)	返回 3.5

示例:

将示例脚本添加到应用程序并运行。要查看结果, 将结果列中列出的字段添加到应用程序中的工作表。

```
RangeTab:
LOAD recno() as RangeID, RangeFractile(0.5,Field1,Field2,Field3) as MyRangeFrac INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

结果列表显示了为表格中的每条记录返回的 MyRangeFrac 值。

结果表

RangeID	MyRangeFrac
1	6
2	3
3	8
4	11
5	5
6	4

带有表达式的示例：

```
RangeFractile (0.5, Above(Sum(MyField),0,3))
```

在此例中，内部记录函数 **Above()** 包含可选的 offset 和 count 参数。这将产生一系列可用作任何范围函数的输入的结果。在这种情况下，`Above(Sum(MyField),0,3)` 会返回当前行和上方两行的 `MyField` 结果。这些值可以作为 **RangeFractile()** 函数的输入。因此，对于下面表格中的底部行，这等同于 `RangeFractile(0.5, 3,4,6)`，即对序列计算 3、4 和 6 的 0.5 分位数。下面表格中的前两行，范围中的值数目相应减少，其中没有行在当前行上方。将会为其他内部记录函数产生类似的结果。

样本数据

MyField	RangeFractile(0.5, Above(Sum(MyField),0,3))
1	1
2	1.5
3	2
4	3
5	4
6	5

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
1
2
3
4
5
6
] ;
```

另请参见：

- *Above - 图表函数 (page 596)*
- *Fractile - 图表函数 (page 252)*

RangeIRR

RangeIRR() 用于返回按输入值表示的一系列现金流的内部回报率。

内部收益率由定期发生的付款(负值)和收入(正值)构成的投资回报率决定。

语法：

```
RangeIRR(value[, value][, Expression])
```

返回数据类型： 数字

参数：

参数

参数	说明
value	由内部记录函数和第三个可选参数返回的单个值或一系列值。计算此函数至少需要一个正值和一个负值。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

文本值，NULL 值和缺失值都忽略不计。

示例表格

示例	结果
`RangeIRR(-70000,12000,15000,18000,21000,26000)`	返回 0.0866

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。
`RangeTab3:`
`LOAD *,`
`recno() as RangeID,`
`RangeIRR(Field1,Field2,Field3) as RangeIRR;`
`LOAD * INLINE [`
`Field1|Field2|Field3`
`-10000|5000|6000`
`-2000|NULL|7000`
`-8000|'abc'|8000`
`-1800|11000|9000`
`-5000|5000|9000`
`-9000|4000|2000`
`] (delimiter is '|');`
	结果列表显示了为表格中的每条记录返回的 RangeIRR 值。
	RangeID RangeIRR
	1 0.0639
	2 0.8708
	3 -
	4 5.8419
	5 0.9318
	6 -0.2566

另请参见：

- *内部记录函数 (page 593)*

RangeKurtosis

RangeKurtosis() 用于返回与数值范围的峰度对应的值。

语法：

```
RangeKurtosis(first_expr[, Expression])
```

返回数据类型： 数字

参数：

该函数的参数可能包含内部记录函数，并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

如果找不到任何数值，则返回 NULL 值。

示例和结果：

函数示例

示例	结果
RangeKurtosis (1,2,4,7)	返回 -0.28571428571429

另请参见：

- *Kurtosis - 图表函数 (page 259)*

RangeMax

RangeMax() 用于返回在表达式或字段内找到的最高数值。

语法：

```
RangeMax(first_expr[, Expression])
```

返回数据类型：数字

参数：

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

如果找不到任何数值，则返回 NULL 值。

示例和结果：

函数示例

示例	结果
`RangeMax (1,2,4)`	返回 4
`RangeMax (1,'xyz')`	返回 1
`RangeMax (null( ), 'abc')`	返回 NULL

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
RangeTab3:
LOAD recno() as RangeID, RangeMax(Field1,Field2,Field3) as MyRangeMax INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

结果列表显示了为表格中的每条记录返回的 MyRangeMax 值。

结果表

RangeID	MyRangeMax
1	10
2	7

RangeID	MyRangeMax
3	8
4	18
5	9
6	9

带有表达式的示例：

```
RangeMax (Above(MyField,0,3))
```

返回在当前行和当前行上方两行中计算的三个 **MyField** 字段值范围的最大值。通过指定第三个参数作为 3，**Above()** 函数会返回三个值，如果上方有足够的行，会将其作为 **RangeMax()** 函数的输入。

示例中所使用的数据：

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeMax (Above(Sum(MyField),1,3))
10	10
2	10
8	10
18	18
5	18
9	18

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

RangeMaxString

RangeMaxString() 用于以文本排序顺序返回在表达式或字段中找到的最后一个值。

语法：

```
RangeMaxString(first_expr[, Expression])
```

返回数据类型：字符串

参数：

该函数的参数可能包含内部记录函数，并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

示例和结果：

函数示例

示例	结果
`RangeMaxString (1,2,4)`	返回 4
`RangeMaxString ('xyz','abc')`	返回“xyz”
`RangeMaxString (5,'abc')`	返回“abc”
`RangeMaxString (null( ))`	返回 NULL

带有表达式的示例：

```
RangeMaxString (Above(MaxString(MyField),0,3))
```

返回当前行和当前行上两行中评估的 **MaxString(MyField)** 函数三个结果中最后的值（以文本排序方式）。

示例中所使用的数据：

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeMaxString(Above(MaxString(MyField),0,3))
10	10
abc	abc
8	abc
def	def
xyz	xyz
9	xyz

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
'def'
'xyz'
9
] ;
```

另请参见：

- *MaxString - 图表函数 (page 365)*

RangeMin

RangeMin() 用于返回在表达式或字段内找到的最低数值。

语法：

```
RangeMin(first_expr[, Expression])
```

返回数据类型： 数字

参数：

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

如果找不到任何数值，则返回 NULL 值。

示例和结果：

函数示例

示例	结果
`RangeMin (1,2,4)`	返回 1
`RangeMin (1,'xyz')`	返回 1
`RangeMin (null( ), 'abc')`	返回 NULL

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
RangeTab3:
LOAD recno() as RangeID, RangeMin(Field1,Field2,Field3) as MyRangeMin INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

结果列表显示了为表格中的每条记录返回的 MyRangeMin 值。

结果表

RangeID	MyRangeMin
1	5
2	2
3	2
4	9
5	5
6	2

带有表达式的示例：

```
RangeMin (Above(MyField,0,3)
```

返回在当前行和当前行上方两行中计算的三个 **MyField** 字段值范围的最小值。通过指定第三个参数作为 3，**Above()** 函数会返回三个值，如果上方有足够的行，会将其作为 **RangeMin()** 函数的输入。

示例中所使用的数据：

样本数据

MyField	RangeMin(Above(MyField,0,3))
10	10
2	2
8	2
18	2
5	5
9	5

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
2
```

```
8
18
5
9
] ;
```

另请参见:

- *Min - 图表函数 (page 204)*

RangeMinString

RangeMinString() 用于以文本排序顺序返回在表达式或字段中找到的第一个值。

语法:

```
RangeMinString(first_expr[, Expression])
```

返回数据类型: 字符串

参数:

该函数的参数可能包含内部记录函数,并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

示例和结果:

函数示例

示例	结果
`RangeMinString (1,2,4)`	返回 1
`RangeMinString ('xyz','abc')`	返回“abc”
`RangeMinString (5,'abc')`	返回 5
`RangeMinString (null( ))`	返回 NULL

带有表达式的示例:

```
RangeMinString (Above(MinString(MyField),0,3))
```

返回当前行和当前行上两行中评估的 **MinString(MyField)** 函数三个结果中的第一个值(以文本排序方式)。

示例中所使用的数据:

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeMinString(Above(MinString(MyField),0,3))
10	10
abc	10
8	8
def	8
xyz	8
9	9

示例中所使用的数据:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
'def'
'xyz'
9
] ;
```

另请参见:

- *MinString - 图表函数 (page 368)*

RangeMissingCount

RangeMissingCount() 用于返回表达式或字段中非数字值(包括 NULL)的数量。

语法:

```
RangeMissingCount(first_expr[, Expression])
```

返回数据类型: 整数

参数:

该函数的参数可能包含内部记录函数,并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要计数的数据。
Expression	可选表达式或字段包含要计数的数据范围。

示例和结果：

函数示例

示例	结果
`RangeMissingCount (1,2,4)`	返回 0
`RangeMissingCount (5,'abc')`	返回 1
`RangeMissingCount (null( ))`	返回 1

带有表达式的示例：

```
RangeMissingCount (Above(MinString(MyField),0,3))
```

返回当前行和当前行上两行中评估的 **MinString(MyField)** 函数三个结果中的非数字值数量。

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeMissingCount (Above(MinString (MyField),0,3))	Explanation
10	2	返回 2, 因为此行上面没有行, 因此 3 个值中缺失 2 个。
abc	2	返回 2, 因为当前行上面只有 1 行, 并且当前行是非数字值 ('abc')。
8	1	返回 1, 因为 3 行中有 1 行包含非数字值 ('abc')。
def	2	返回 2, 因为 3 行中有 2 行包含非数字值('def' 和 'abc') 。
xyz	2	返回 2, 因为 3 行中有 2 行包含非数字值(' xyz' 和 'def') 。
9	2	返回 2, 因为 3 行中有 2 行包含非数字值(' xyz' 和 'def') 。

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
'def'
'xyz'
9
] ;
```

另请参见:

- *MissingCount - 图表函数 (page 219)*

RangeMode

RangeMode() 用于查找表达式或字段中最常出现的值(即模式值)。

语法:

```
RangeMode(first_expr {, Expression})
```

返回数据类型: 数字

参数:

该函数的参数可能包含内部记录函数,并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制:

如果多个值共享最高频率,则返回 NULL。

示例和结果:

函数示例

示例	结果
`RangeMode (1,2,9,2,4)`	返回 2
`RangeMode ('a',4,'a',4)`	返回 NULL
`RangeMode (null( ))`	返回 NULL

示例:

将示例脚本添加到应用程序并运行。要查看结果,将结果列中列出的字段添加到应用程序中的工作表。

```
RangeTab3:
LOAD recno() as RangeID, RangeMode(Field1,Field2,Field3) as MyRangeMode INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
```

```
9,4,2
];
```

结果列表显示了为表格中的每条记录返回的 **MyRangeMode** 值。

结果表

RangeID	MyRangMode
1	-
2	-
3	8
4	-
5	5
6	-

带有表达式的示例：

```
RangeMode (Above(MyField,0,3))
```

返回在当前行和当前行上方两行中评估的三个 **MyField** 字段结果中最常出现的值。通过指定第三个参数作为 3，**Above()** 函数会返回三个值，如果上方有足够的行，会将其作为 **RangeMode()** 函数的输入。

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeMode(Above(MyField,0,3))
10	返回 10，因为上面没有行，因此单个值是最常出现的。
2	-
8	-
18	-
5	-
9	-

另请参见：

- *Mode - 图表函数 (page 207)*

RangeNPV

RangeNPV() 用于返回基于折扣率和一系列未来定期付款(负值)和收入(正值)的投资的净现值。结果拥有一个 **money** 的默认数字格式。

对于不必是周期性的现金流，请参阅 *RangeXNPV (page 677)*。

语法：

```
RangeNPV(discount_rate, value[,value][, Expression])
```

返回数据类型： 数字

参数：

参数

参数	说明
discount_rate	每周期的利率。
value	每个周期结束时发生的付款或收入。每个值都可能是由内部记录函数和第三个可选参数返回的单个值或一系列值。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

文本值，NULL 值和缺失值都忽略不计。

示例	结果
RangeNPV(0.1,-10000,3000,4200,6800)	返回 1188.44
将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。 RangeTab3: LOAD *, recno() as RangeID, RangeNPV(Field1,Field2,Field3) as RangeNPV; LOAD * INLINE [Field1|Field2|Field3 10|5|-6000 2|NULL|7000 8|'abc'|8000 18|11|9000 5|5|9000 9|4|2000] (delimiter is '|');	结果列表显示了为表格中的每条记录返回的 RangeNPV 值。 RangeID RangeNPV 1 $-49.13 2 $777.78 3 $98.77 4 $25.51 5 $250.83 6 $20.40

另请参见:

- *内部记录函数 (page 593)*

RangeNullCount

RangeNullCount() 用于查找表达式或字段中 NULL 值的数量。

语法:

```
RangeNullCount(first_expr [, Expression])
```

返回数据类型: 整数

参数:

该函数的参数可能包含内部记录函数,并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

示例和结果:

函数示例

示例	结果
`RangeNullCount (1,2,4)`	返回 0
`RangeNullCount (5,'abc')`	返回 0
`RangeNullCount (null( ), null( ))`	返回 2

带有表达式的示例:

```
RangeNullCount (Above(Sum(MyField),0,3))
```

返回当前行和当前行上两行中评估的 **Sum(MyField)** 函数三个结果中的 NULL 值数量。

*在以下示例中复制 **MyField** 不会导致出现 NULL 值。*

样本数据

MyField	RangeNullCount(Above(Sum(MyField),0,3))
10	返回 2,因为此行上面没有行,因此 3 个值中缺失 2 个 (=NULL)。
'abc'	返回 1,因为当前行上面只有一行,因此三个值中缺失一个 (=NULL)。
8	返回 0,因为三行中没有任何一行为 NULL 值。

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
] ;
```

另请参见：

- *NullCount - 图表函数 (page 222)*

RangeNumericCount

RangeNumericCount() 用于查找表达式或字段中数字值的数量。

语法：

```
RangeNumericCount(first_expr[, Expression])
```

返回数据类型： 整数

参数：

该函数的参数可能包含内部记录函数，并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

示例和结果：

函数示例

示例	结果
RangeNumericCount (1,2,4)	返回 3
RangeNumericCount (5,'abc')	返回 1
RangeNumericCount (null( ))	返回 0

带有表达式的示例：

```
RangeNumericCount (Above(MaxString(MyField),0,3))
```

返回当前行和当前行上两行中评估的 **MaxString(MyField)** 函数三个结果中的数字值数量。

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeNumericCount(Above(MaxString(MyField),0,3))
10	1
abc	1
8	2
def	1
xyz	1
9	1

示例中所使用的数据:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
def
xyz
9
] ;
```

另请参见:

- *NumericCount - 图表函数 (page 224)*

RangeOnly

RangeOnly() 是一个对偶函数,用于返回一个值(如果表达式计算为一个独特的值)。如果不是一个独特的值,则返回 **NULL** 值。

语法:

```
RangeOnly(first_expr[, Expression])
```

返回数据类型: 双

参数:

该函数的参数可能包含内部记录函数,并在其内部返回一系列值。

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

示例和结果：

示例	结果
RangeOnly (1,2,4)	返回 NULL
RangeOnly (5,'abc')	返回 NULL
RangeOnly (null( ), 'abc')	返回“abc”
RangeOnly(10,10,10)	返回 10

另请参见：

- *Only - 图表函数 (page 210)*

RangeSkew

RangeSkew() 用于返回与数值范围的偏度对应的值。

语法：

```
RangeSkew(first_expr[, Expression])
```

返回数据类型： 数字

参数：

该函数的参数可能包含内部记录函数，并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

如果找不到任何数值，则返回 NULL 值。

示例和结果：

函数示例

示例	结果
rangeskew (1,2,4)	返回 0.93521952958283
rangeskew (above (SalesValue,0,3))	返回从当前行与当前行上两行中计算的 above() 函数返回的三个值的范围的滑动偏度。

示例中所使用的数据：

样本数据

CustID	RangeSkew(Above(SalesValue,0,3))
1-20	-, -, 0.5676, 0.8455, 1.0127, -0.8741, 1.7243, -1.7186, 1.5518, 1.4332, 0, 1.1066, 1.3458, 1.5636, 1.5439, 0.6952, -0.3766

```
SalesTable:
LOAD recno() as CustID, * inline [
SalesValue
101
163
126
139
167
86
83
22
32
70
108
124
176
113
95
32
42
92
61
21
] ;
```

另请参见：

- *Skew - 图表函数 (page 288)*

RangeStdev

RangeStdev() 用于查找数字系列的标准偏差。

语法：

```
RangeStdev(first_expr[, Expression])
```

返回数据类型： 数字

参数：

该函数的参数可能包含内部记录函数，并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

如果找不到任何数值，则返回 NULL 值。

示例和结果：

函数示例

示例	结果
`RangeStdev (1,2,4)`	返回 1.5275252316519
`RangeStdev (null( ))`	返回 NULL
`RangeStdev (above (SalesValue),0,3))`	返回从当前行与当前行上两行中计算的 above() 函数返回的三个值的范围的滑动标准。

示例中所使用的数据：

样本数据

CustID	RangeStdev(SalesValue, 0,3))
1-20	-,43.841, 34.192, 18.771, 20.953, 41.138, 47.655, 36.116, 32.716, 25.325, 38,000, 27.737, 35.553, 33.650, 42.532, 33.858, 32.146, 25.239, 35.595

```
SalesTable:
LOAD recno() as CustID, * inline [
SalesValue
101
163
126
139
167
86
83
22
32
70
108
124
176
113
95
32
42
92
61
21
```

```
] ;
```

另请参见：

- *Stdev - 图表函数 (page 291)*

RangeSum

RangeSum() 返回一系列值的总和。所有非数字值均被视为0。

语法：

```
RangeSum(first_expr[, Expression])
```

返回数据类型： 数字

参数：

该函数的参数可能包含内部记录函数，并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

限制：

RangeSum 函数将所有非数字值视为 0。

示例和结果：

示例

示例	结果
`RangeSum (1,2,4)`	返回 7
`RangeSum (5,'abc')`	返回 5
`RangeSum (null( ))`	返回 0

示例：

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。

```
RangeTab3:

LOAD recno() as RangeID, Rangesum(Field1,Field2,Field3) as MyRangeSum INLINE [

Field1, Field2, Field3
```

```
10,5,6

2,3,7

8,2,8

18,11,9

5,5,9

9,4,2
];
```

结果列表显示了为表格中的每条记录返回的 MyRangeSum 值。

结果表

RangeID	MyRangeSum
1	21
2	12
3	18
4	38
5	19
6	15

带有表达式的示例：

```
RangeSum (Above(MyField,0,3))
```

返回当前行和当前行上方两行中三个 **MyField)** 字段值的总和。通过指定第三个参数作为 3，**Above()** 函数会返回三个值，如果上方有足够的行，会将其作为 **RangeSum()** 函数的输入。

示例中所使用的数据：

*禁用 **MyField** 排序可确保示例符合预期。*

样本数据

MyField	RangeSum(Above(MyField,0,3))
10	10
2	12
8	20
18	28
5	31
9	32

示例中所使用的数据:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

另请参见:

- *Sum - 图表函数 (page 212)*
- *Above - 图表函数 (page 596)*

RangeTextCount

RangeTextCount() 用于返回表达式或字段中文本值的数量。

语法:

```
RangeTextCount(first_expr[, Expression])
```

返回数据类型: 整数

参数:

该函数的参数可能包含内部记录函数,并在其内部返回一系列值。

参数

参数	说明
first_expr	表达式或字段包含要度量的数据。
Expression	可选表达式或字段包含要度量的数据范围。

示例和结果:

函数示例

示例	结果
RangeTextCount (1,2,4)	返回 0
RangeTextCount (5,'abc')	返回 1
RangeTextCount (null( ))	返回 0

带有表达式的示例:

```
RangeTextCount (Above(MaxString(MyField),0,3))
```

返回当前行和当前行上两行中评估的 **MaxString(MyField)** 函数三个结果中的文本值数量。

示例中所使用的数据:

*禁用 **MyField** 排序可确保示例符合预期。*

示例数据

MyField	MaxString(MyField)	RangeTextCount(Above(Sum(MyField),0,3))
10	10	0
abc	abc	1
8	8	1
def	def	2
xyz	xyz	2
9	9	2

示例中所使用的数据:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
null()
'xyz'
9
] ;
```

另请参见:

TextCount - 图表函数 (page 228)

RangeXIRR

RangeXIRR() 用于返回现金流计划表的内部回报率(不必是周期性的)。要计算一系列周期性现金流的内部回报率,请使用 **RangeIRR** 函数。

语法:

```
RangeXIRR(value, date{, value, date})
```

返回数据类型：数字

参数：

参数

参数	说明
value	对应付款日期计划表的现金流或一系列现金流。系列值必须至少包含一个正值和一个负值。
date	对应现金流支付的付款日期或付款日期计划表。

限制：

文本值，NULL 值和缺失值都忽略不计。

所有付款按 365 天一年年折扣。

示例	结果
`RangeXIRR(-2500,'2008-01-01',2750,'2008-09-01')`	返回 0.1532

另请参见：

- *RangeIRR (page 654)*

RangeXNPV

RangeXNPV() 用于返回现金流计划表的净现值(不必是周期性的)。结果默认采用货币数字格式。要计算一系列周期性现金流的净现值，请使用 **RangeNPV** 函数。

语法：

```
RangeXNPV(discount_rate, values, dates[, Expression])
```

返回数据类型：数字

参数：

参数

参数	说明
discount_rate	每周期的利率。
values	对应付款日期计划表的现金流或一系列现金流。每个值都可能是由内部记录函数和第三个可选参数返回的单个值或一系列值。系列值必须至少包含一个正值和一个负值。
dates	对应现金流支付的付款日期或付款日期计划表。

限制：

文本值，NULL 值和缺失值都忽略不计。

所有付款按 365 天一年年折扣。

示例表格

示例	结果
`RangeXNPV(0.1, -2500,'2008-01-01',2750,'2008-09-01')`	返回 80.25

将示例脚本添加到应用程序并运行。要查看结果，将结果列中列出的字段添加到应用程序中的工作表。
`RangeTab3:`
`LOAD *,`
`recno() as RangeID,`
`RangeXNPV(Field1,Field2,Field3) as RangeNPV;`
`LOAD * INLINE [`
`Field1|Field2|Field3`
`10|5|-6000`
`2|NULL|7000`
`8|'abc'|8000`
`18|11|9000`
`5|5|9000`
`9|4|2000`
`] (delimiter is '|');`
	结果列表显示了为表格中的每条记录返回的 RangeXNPV 值。
	RangeID: RangeXNPV
	1: $-49.13
	2: $777.78
	3: $98.77
	4: $25.51
	5: $250.83
	6: $20.40

5.22 排名和集群函数

这些函数只可用于图表表达式中。

图表中的排名函数

当使用这些函数时，会自动禁用零值。NULL 值将被忽略。

Rank

Rank() 用于在表达式中计算图表的行数，并且对于每一行显示在表达式中计算的维度值的相对位置。当计算表达式的值时，该函数将结果与包含当前列片段的其他行的结果比较，然后返回片段中当前行的排名。

```
Rank - 图表函数([TOTAL [<fld {, fld}>]] expr[, mode[, fmt]])
```

HRank

HRank() 用于对表达式求值，并将结果与包含透视表的当前行段的其他行的结果进行比较。然后，此函数返回段内当前行的排行。

```
HRank- 图表函数([TOTAL] expr[, mode[, fmt]])
```

图表中的集群函数

KMeans2D

属性组**站点许可证**包含与 Qlik Sense 系统许可证相关的属性。所有字段都是必填字段，不能为空。

站点许可证属性

属性名称	说明
所有者名称	Qlik Sense 产品所有者的用户名。
所有者组织	Qlik Sense 产品所有者所属组织的名称。
序列号	分配给 Qlik Sense 软件的序列号。
控制号	分配给 Qlik Sense 软件的控制编号。
LEF 访问	分配给 Qlik Sense 软件的 License Enabler File (LEF)。

KMeans2D() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的集群 id。集群算法使用的列分别由参数 coordinate_1 和 coordinate_2 确定。二者都是聚合型。创建的集群数由 num_clusters 参数确定。数据可以通过规范参数进行规范化。

```
KMeans2D - 图表函数(num_clusters, coordinate_1, coordinate_2 [, norm])
```

KMeansND

KMeansND() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的集群 id。集群算法使用的列由参数 coordinate_1 和 coordinate_2 等确定(可达 n 列)。这些都是聚合型。创建的集群数由 num_clusters 参数确定。

```
KMeansND - 图表函数(num_clusters, num_iter, coordinate_1, coordinate_2
[,coordinate_3 [, ...]])
```

KMeansCentroid2D

KMeansCentroid2D() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的所需坐标。集群算法使用的列分别由参数 coordinate_1 和 coordinate_2 确定。二者都是聚合型。创建的集群数由 num_clusters 参数确定。数据可以通过规范参数进行规范化。

```
KMeansCentroid2D - 图表函数(num_clusters, coordinate_no, coordinate_1,
coordinate_2 [, norm])
```

KMeansCentroidND

KMeansCentroidND() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的所需坐标。集群算法使用的列由参数 coordinate_1 和 coordinate_2 等确定(可达 n 列)。这些都是聚合型。创建的集群数由 num_clusters 参数确定。

```
KMeansCentroidND- 图表函数(num_clusters, num_iter, coordinate_no, coordinate_
1, coordinate_2 [,coordinate_3 [, ...]])
```

Rank - 图表函数

Rank() 用于在表达式中计算图表的行数，并且对于每一行显示在表达式中计算的维度值的相对位置。当计算表达式的值时，该函数将结果与包含当前列片段的其他行的结果比较，然后返回片段中当前行的排名。

列段数据

	Region	Country	Population	Rank(Population)
Column segment #1	Americas	Mexico	128,932,753	2
	Americas	Canada	37,742,154	3
	Americas	United States of America	331,002,651	1
Column segment #2	Europe	Sweden	10,099,265	4
	Europe	United Kingdom	67,886,011	2
	Europe	France	65,273,511	3
	Europe	Germany	83,783,942	1

对于非表格图表，将定义当前列段数据，就像显示在图表的垂直表等同物中一样。

语法：

```
Rank([TOTAL] expr[, mode[, fmt]])
```

返回数据类型： 双

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
mode	指定函数结果的数字呈现形式。
fmt	指定函数结果的文本呈现形式。
TOTAL	如果图表是一维或如果表达式前面有 **TOTAL** 限定符，则该函数用于评估整列。如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

排行以双重值形式返回，在每一行拥有一个唯一排行的情况下，排行是一个介于 1 和当前列段数据行数之间的整数。

当多行共享同一个排名时，文本和数字呈现形式可使用 **mode** 和 **fmt** 参数进行控制。

mode

第二个参数 **mode** 可获取以下值：

mode 示例

值	说明
0(默认)	如果共享组中的全部排行处在整个排行中间值的下半部分,全部行都获得共享组的最低排行。 如果共享组中的全部排行处在整个排行中间值的上半部分,全部行都获得共享组的最高排行。 如果共享组中的排行跨越整个排行的中间值,全部行都获得整个列片断中最高和最低排行的平均值。
1	全部行的最低排行。
2	全部行的平均排行。
3	全部行的最高排行。
4	第一行的最低排行,然后每一行都提高一位。

fmt

第三个参数 **fmt** 可获取以下值:

fmt示例

值	说明
0(默认)	全部行中的低值 **-** 高值(如 3 - 4)。
1	全部行的高值。
2	第一行的低值,以后各行都为空白。

mode 4 和 **fmt** 2 的行顺序由图表维度的排序决定。

示例和结果:

使用维度 Product 和 Sales 创建两个可视化,使用 Product 和 UnitSales 创建其他可视化。添加度量,如下表所示。

排名示例

示例	结果
示例 1。创建一个表格，维度为 Customer 和 Sales、度量为 Rank (Sales)	结果取决于维度的排序顺序。如果按 Customer 对表格排序，表格会列出 Astrida 的所有 Sales 值，然后列出 Betacab 的所有同类型值，以此类推。Sales 值为 12 的 Rank(Sales) 结果将显示 10，Sales 值为 13 的相同字段结果将显示 9，以此类推，并且对 Sales 值为 78 的排行值返回 1。下一个列段数据从 Betacab 开始，在此列段数据中其第一个 Sales 值是 12。此字段的排行值 Rank(Sales) 显示为 11。 如果此表格按 Sales 排序，则列段数据包含 Sales 值以及相应的 Customer。因为有两个 Sales 值是 12(对于 Astrida 和 Betacab) ，因此该列段数据每个 Customer 值的 Rank (Sales) 值都是 1-2。这是因为有两个 Customer 值的 Sales 值都是 12。如果有 4 个值，则所有行的结果都是 1-4。这显示了使用参数 fmt 默认值 (0) 时的结果。
示例 2。将维度 Customer 替换为 Product，并添加度量 Rank(Sales,1,2)	这样将在每个列段数据的第一行中返回 1，并将所有其他行留空，因为参数 **mode** 和 **fmt** 分别设置为 1 和 2。

示例 1 的结果，按 Customer 对表格排序：

结果表

Customer	Sales	Rank(Sales)
Astrida	12	10
Astrida	13	9
Astrida	20	8
Astrida	22	7
Astrida	45	6
Astrida	46	5
Astrida	60	4
Astrida	65	3
Astrida	70	2
Astrida	78	1
Betcab	12	11

示例 1 的结果，按 Sales 对表格排序：

结果表

Customer	Sales	Rank(Sales)
Astrida	12	1-2
Betacab	12	1-2
Astrida	13	1
Betacab	15	1
Astrida	20	1
Astrida	22	1-2
Betacab	22	1-2
Betacab	24	1-2
Canutility	24	1-2

示例中所使用的数据:

```
ProductData:

Load * inline [

Customer|Product|UnitSales|UnitPrice

Astrida|AA|4|16

Astrida|AA|10|15

Astrida|BB|9|9

Betacab|BB|5|10

Betacab|CC|2|20

Betacab|DD|0|25

Canutility|AA|8|15

Canutility|CC|0|19

] (delimiter is '|');


Sales2013:
crosstable (Month, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
```

```
] (delimiter is '|');
```

另请参见：

- *Sum - 图表函数 (page 212)*

HRank- 图表函数

HRank() 用于对表达式求值，并将结果与包含透视表的当前行段的其他行的结果进行比较。然后，此函数返回段内当前行的排行。

语法：

```
HRank([ TOTAL ] expr [ , mode [, fmt ] ])
```

返回数据类型： 双

> *该函数只在透视表中起作用。在全部其他类别的图表中，它返回 NULL。*

参数：

参数

参数	说明
expr	表达式或字段包含要度量的数据。
mode	指定函数结果的数字呈现形式。
fmt	指定函数结果的文本呈现形式。
TOTAL	如果图表是一维或如果表达式前面有 **TOTAL** 限定符，则该函数用于评估整列。如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

如果透视表是一维，或者如果表达式前面有一个 **total** 限定词，则当前行片断总是与整行相等。如果透视表有多个水平维度，则当前行片断将只包括值与所有维度行中当前列相同的列，除显示字段排序间上一次水平维度的行之外。

排名将会以双值的方式返回，当每一列拥有一个唯一排名的情况下将会是一个介与 1 和当前行片断列数之间的整数。

当多列共享同一个排行时，文本和数字呈现形式可使用 **mode** 和 **format** 参数进行控制。

第二个参数 **mode** 用于指定函数结果的数字呈现形式：

mode 示例

值	说明
0(默认)	如果共享组中的全部排行处在整个排行中间值的下半部分,全部列都获得共享组的最低排行。 如果共享组中的全部排行处在整个排行中间值的上半部分,全部列都获得共享组的最高排行。 如果共享组中的排行跨越整个排行的中间值,全部行都获得整个列片断中最高和最低排行的平均值。
1	该组中全部列的最低排行。
2	该组中全部列的平均排行。
3	该组中全部列的最高排行。
4	第一列的最低排行,然后该组中每一列都依次提高一位。

第三个参数 **format** 用于指定函数结果的文本呈现形式:

format 示例

值	说明
0(默认)	在组的全部列中的低值**&' - '&**高值(如 3 - 4)。
1	该组中全部列的低值。
2	第一列的低值,以后各列都为空白。

mode 4 和 **format** 2 的列顺序由图表维度的排序决定。

示例:

```
HRank( sum( Sales ))
HRank( sum( Sales ), 2 )
HRank( sum( Sales ), 0, 1 )
```

用 K 均值优化实际示例

下面的示例演示了一个实际的用例,其中 k 均值集群和形心函数应用于数据集。K 均值函数将数据点分隔成共享相似性的集群。随着 K 均值算法在可配置的迭代次数上的应用,集群变得更加紧凑和差异化。

K 均值在各种各样的用例中跨多个领域使用;集群用例的一些示例包括客户细分、欺诈检测、预测帐户损耗、针对客户的激励、网络犯罪识别和交付路线优化。K 均值聚类算法正越来越多地被用于企业试图推断模式和优化服务产品的场景。

Qlik Sense K 均值与形心函数

Qlik Sense 提供两个 K 均值函数,根据相似性将数据点分组到集群中。请参阅 *KMeans2D - 图表函数 (page 694)* 和 *KMeansND - 图表函数 (page 704)*。**KMeans2D** 函数接受二维数据,通过**散点图**很好地显示结果。**KMeansND** 函数接受两个以上的维度。由于在标准图表上概念化二维结果很容易,下面

的演示在使用两个维度的**散点图**上应用 K 均值。K 均值聚类可以通过表达式着色来可视化;或按本例中所述的维度。

Qlik Sense 形心函数确定集群中所有数据点的算术平均位置,并标识一个中心点或该集群的形心。对于每个图表行(或记录),纸芯函数显示分配了该数据点的集群的坐标。请参阅 *KMeansCentroid2D - 图表函数 (page 715)* 和 *KMeansCentroidND- 图表函数 (page 716)*。

用例和示例概述

下面的示例将通过模拟的真实世界场景逐步展开。美国纽约州的一家纺织公司必须通过降低交货成本来减少开支。一种方法是将仓库搬迁到离分销商更近的地方。该公司在纽约州拥有 118 个分销商。下面的演示模拟了运营经理如何使用 K 均值函数将分销商划分为五个集群地理位置,然后使用形心函数确定这些集群中心的五个最佳仓库位置。目标是发现可用于确定五个中心仓库位置的地图坐标。

数据集

该数据集基于纽约州随机生成的名称和地址,并具有准确的经纬度坐标。数据集包含以下十列:id、first_name、last_name、telephone、address、city、state、zip、latitude、longitude。数据集在下面作为一个文件提供,您可以在本地下载,然后上传到 Qlik Sense 或对数据加载编辑器内联。创建的应用程序可以命名为 *Distributors KMeans and Centroid*,应用程序中的第一个工作表命名为 *Distribution cluster analysis*。

选择以下链接来下载样本数据文件:DistributorData.csv

Distributor 数据集:Qlik Sense 中数据加载编辑器的内联加载 (page 692)

标题:DistributorData

记录的总数:118

应用 KMeans2D 函数

在本例中,使用 *DistributorData* 数据集演示**散点图**的配置,应用 **KMeans2D** 函数,并按维度为图表着色。

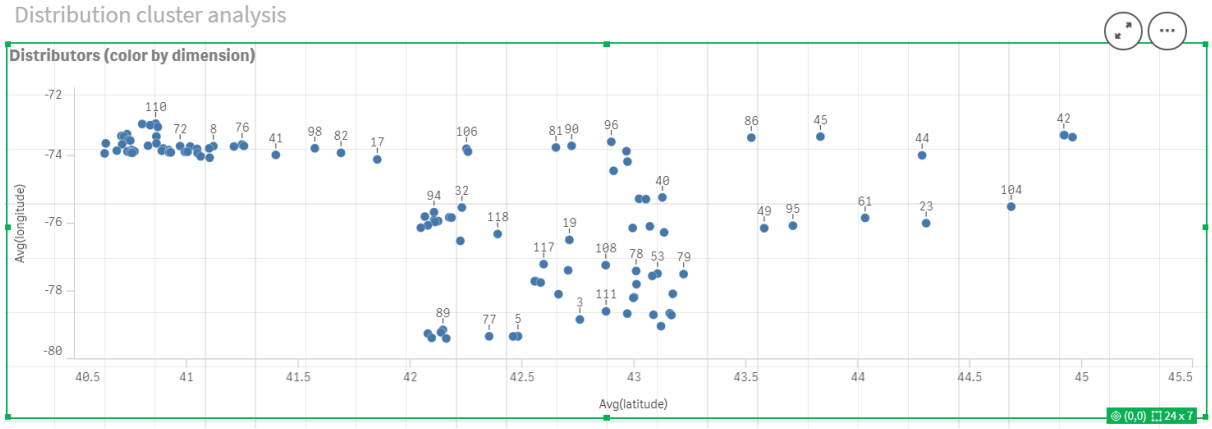
注意 Qlik Sense K 均值函数支持使用名为深度差 (DeD) 的方法支持自动聚合。如果用户为集群数设置 0,则会为该数据集确定最优集群数。但是,在本例中,为 **num_clusters** 参数创建了一个变量(有关语法,请参阅 *KMeans2D - 图表函数 (page 694)*)。因此,所需的集群数 (k=5) 由一个变量指定。

1. **散点图**被拖到工作表上并命名为 *Distributors (by dimension)*。
2. 创建了**变量**以指定集群数。**变量**被命名为 *vDistClusters*。对于变量 **Definition**,输入 *5*。
3. 图表的**数据**配置:
    a. 在**维度**下,选择**气泡**的 *id* 字段。输入**标签**的 *Cluster id*。
    b. 在**度量**下,*Avg([latitude])* 是 **X 轴**的表达式。
    c. 在**度量**下,*Avg([longitude])* 是 **Y 轴**的表达式。
4. **外观**配置:
    a. 在**颜色和图例**下,为**颜色**选择了**自定义**。
    b. 为图表着色选择了**按维度**。

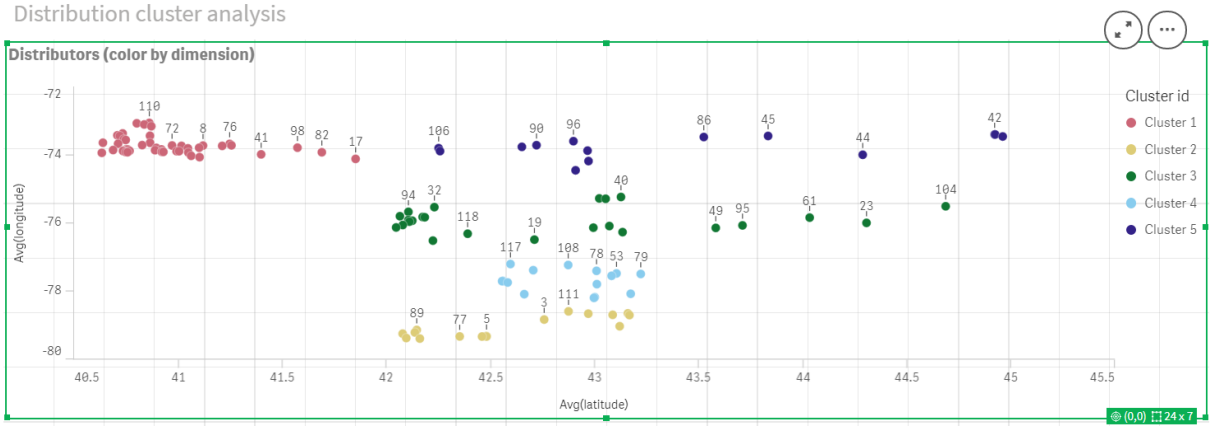
c. 输入了以下表达式：*=pick(aggr(KMeans2D(vDistClusters,only(latitude),only (longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')*

d. 选择了**固定颜色**的复选框。

应用按维度 K 均值着色之前的散点图



应用按维度 K 均值着色之后的散点图



添加**表格**：*分销商*

有一个方便的表格可以快速访问相关数据，这会很有帮助。**散点图**通过表格显示 *ID*，其中添加了相应分商名称供参考。

1. 一个名为 *Distributors* 的**表**被拖到工作表上，并添加了以下**列**（维度）：*id*、*first\u name* 和 *last\u name*。

表格：分销商名称

Distributors

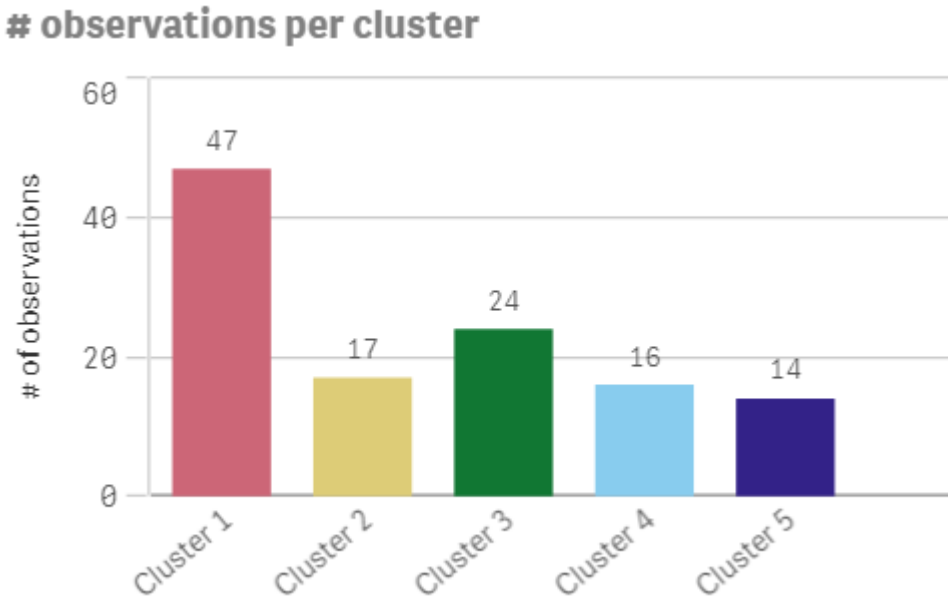
id	first_name	last_name
1	Kaiya	Snow
2	Dean	Roy
3	Eden	Paul
4	Bryanna	Higgins
5	Elisabeth	Lee
6	Skylar	Robinson
7	Cody	Bailey
8	Dario	Sims
9	Deacon	Hood

添加**条形图**:*# observations per cluster*

对于仓库配送场景，了解每个仓库将为多少分销商提供服务是很有帮助的。因此，将创建一个**条形图**，用于测量分配给每个集群的分销商数量。

1. **条形图**被拖动到工作表上。图表被命名为:*# observations per cluster*。
2. **条形图**的**数据配置**:
    a. 添加一个标注为*群集*的**维度**(可以在应用表达式后添加标签)。输入了以下表达式:*=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')*
    b. 添加了标记为 *# of observations* 的**度量**。输入了以下表达式:*=count(aggr(KMeans2D (vDistClusters,only(latitude),only(longitude)),id))*
3. **外观**配置:
    a. 在**颜色和图例**下，为**颜色**选择了**自定义**。
    b. 为图表着色选择了**按维度**。
    c. 输入了以下表达式:*=pick(aggr(KMeans2D(vDistClusters,only(latitude),only (longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')*
    d. 选择了**固定颜色**的复选框。
    e. **显示图例**已关闭。
    f. 在**演示**下方，**值标签**切换为**自动**。
    g. 在 **X 轴**下方:选择了**集群**、**只有标签**。

条形图：# observations per cluster



应用 Centroid2D 函数

为 **Centroid2D** 函数添加了第二个表，该表将标识潜在仓库位置的坐标。此表显示了五个已标识的分销商组的中心位置(形心值)。

1. 将一个**表**拖到工作表上并命名为*群集形心*，并添加以下列：
   a. 添加了标记为 *Clusters* 的**维度**。输入了以下表达式：*=pick(aggr(KMeans2D (vDistClusters,only(latitude),only(longitude)),id)+1,'Warehouse 1','Warehouse 2','Warehouse 3','Warehouse 4','Warehouse 5'*)
   b. 添加了标记为 *latitude (D1)* 的**度量**。输入了以下表达式：*=only(aggr(KMeansCentroid2D (vDistClusters,0,only(latitude),only(longitude)),id))*
   注意参数 **coordinate_no** 对应于第一个 dimension(0)。在该情况下，将根据 x 轴绘制维度 *latitude*。如果我们使用的是 **CentroidND** 函数，并且最多有六个维度，那么这些参数项可以是六个值中的任意一个：0、1、2、3、4 或 5。
   c. 添加了标记为 *longitude (D2)* 的**度量**。输入了以下表达式：*=only(aggr (KMeansCentroid2D(vDistClusters,1,only(latitude),only(longitude)),id))*
   此表达式中的参数 **coordinate_no** 对应于第二个维度(1)。根据 y 轴绘制了维度 *longitude*。

表格：群集形心计算

Cluster centroids

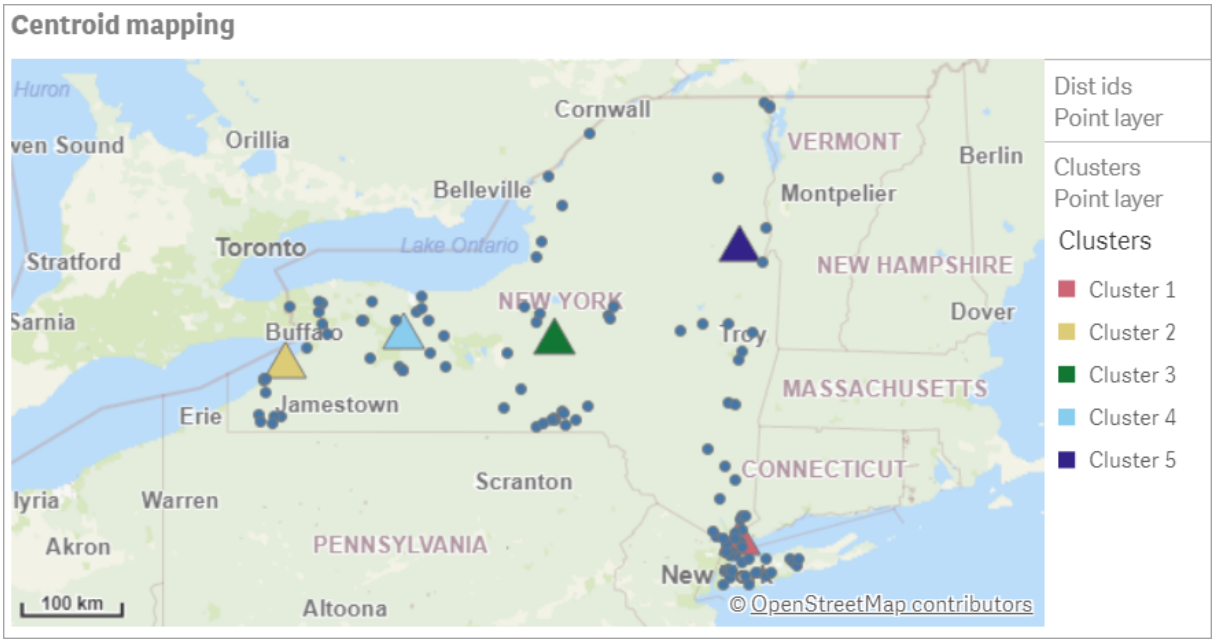
Clusters	latitude (D1)	longitude (D2)
Totals	-	-
Warehouse 1	40.945422240426	-73.719966482979
Warehouse 2	42.590538729412	-79.067889217647
Warehouse 3	42.805089516667	-75.901621883333
Warehouse 4	42.8581692625	-77.6800485875
Warehouse 5	43.436770771429	-73.734622635714

形心映射

下一步是映射形心。它由应用程序开发人员决定，如果他们喜欢把可视化放置在单独的工作表上，就可以如此。

1. 名为 *Centroid mapping* 的**映射**被拖动到工作表上。
2. 在**层**部分中。选择了**添加层**，然后选择了**点层**。
    a. 选择了**字段** *id* 并且添加了 *Dist ids* **标签**。
    b. 在**位置**部分中，选中**纬度和经度字段**的复选框。
    c. 对于**纬度**，选择了*纬度*字段。
    d. 对于**经度**，选择了*经度*字段。
    e. 在**大小和形状**区域中，对于**形状**选择**气泡**，并在滑块上将**大小**减小到“首选项”。
    f. 在**颜色**部分中，选择**单色**，对于**颜色**选择蓝色，对于**轮廓**选择灰色(这些选择也是首选项)。
3. 在**层**部分中，通过选择**添加层**，然后选择**点层**，添加第二个**点层**。
    a. 输入了以下表达式：*=aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)*
    b. 添加了**标签***群集*。
    c. 在**位置**部分中，选中**纬度和经度字段**的复选框。
    d. 对于在本例中沿 x 轴绘制的**纬度**，添加以下表达式：*=aggr(KMeansCentroid2D (vDistClusters,0,only(latitude),only(longitude)),id)*
    e. 对于在本例中沿 y 轴绘制的**经度**，添加以下表达式：*=aggr(KMeansCentroid2D (vDistClusters,1,only(latitude),only(longitude)),id)*
    f. 在**大小和形状**区域中，对于**形状**选择**三角形**，并在滑块上将**大小**减小到“首选项”。
    g. 在**颜色和图例**下，为**颜色**选择了**自定义**。
    h. 为图表着色选择了**按维度**。输入了以下表达式：*=pick(aggr(KMeans2D (vDistClusters,only(latitude),only(longitude)),id)+1,'Cluster 1','Cluster 2','Cluster 3','Cluster 4','Cluster 5')*
    i. 添加了标记为 *Clusters* 的维度。
4. 在**地图设置**中，对于**投影**选择了**自适应**。对于**度量单位**选择了**公制**。

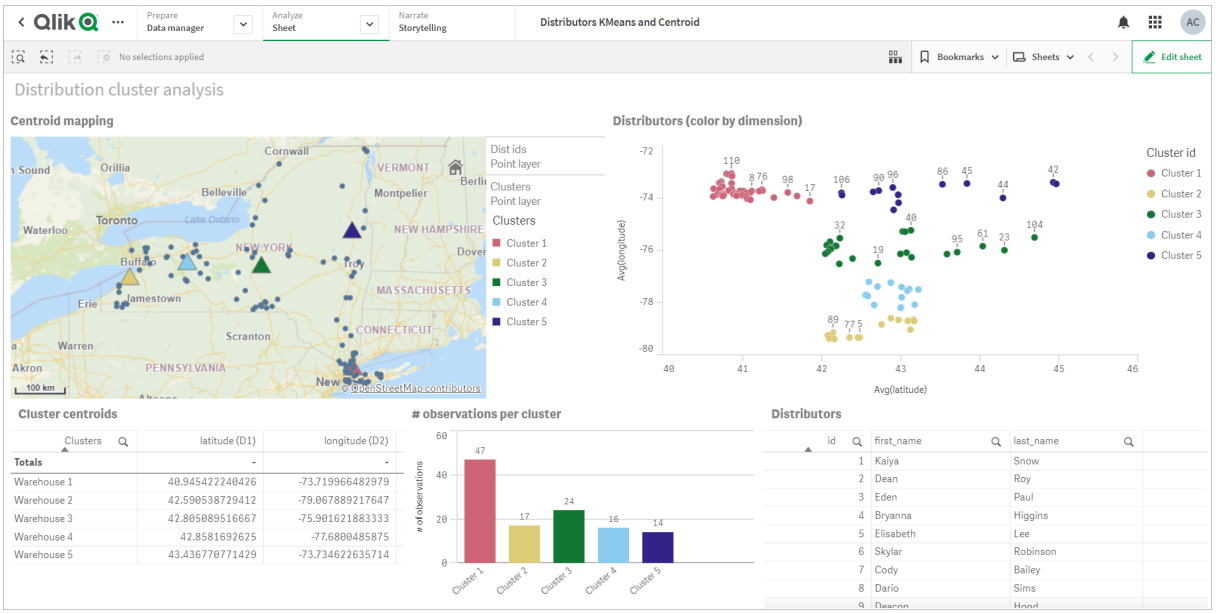
地图按群集映射的形心



结论

在这个真实场景中使用 K 均值函数，根据相似性将分销商分成相似的组或群集；在这种情况下，彼此接近。将形心函数应用于这些群集，识别出 5 个映射坐标。这些坐标提供了建造或定位仓库的初始中心位置。形心函数应用于**地图**图表，这样应用程序用户可以直观地看到形心相对于周围群集数据点的位置。由此产生的坐标表示潜在的仓库位置，这可以最大限度地降低纽约州分销商的交货成本。

应用程序：K 均值和形心分析示例



Distributor 数据集：Qlik Sense 中数据加载编辑器的内联加载

```
DistributorData: Load * Inline [ id,first_name,last_
name,telephone,address,city,state,zip,latitude,longitude 1,Kaiya,Snow,(716) 201-1212,6231
Tonawanda Creek Rd #APT 308,Lockport,NY,14094,43.08926,-78.69313 2,Dean,Roy,(716) 201-
1588,6884 E High St,Lockport,NY,14094,43.16245,-78.65036 3,Eden,Paul,(716) 202-4596,4647
Southwestern Blvd #APT 350,Hamburg,NY,14075,42.76003,-78.83194 4,Bryanna,Higgins,(716) 203-
7041,418 Park Ave,Dunkirk,NY,14048,42.48279,-79.33088 5,Elisabeth,Lee,(716) 203-7043,36 E
Courtney St,Dunkirk,NY,14048,42.48299,-79.31928 6,Skylar,Robinson,(716) 203-7166,26 Greco
Ln,Dunkirk,NY,14048,42.4612095,-79.3317925 7,Cody,Bailey,(716) 203-7201,114 Lincoln
Ave,Dunkirk,NY,14048,42.4801269,-79.322232 8,Dario,Sims,(408) 927-1606,N Castle
Dr,Armonk,NY,10504,41.11979,-73.714864 9,Deacon,Hood,(410) 244-6221,4856 44th
St,Woodside,NY,11377,40.748372,-73.905445 10,Zackery,Levy,(410) 363-8874,61 Executive
Blvd,Farmingdale,NY,11735,40.7197457,-73.430239 11,Rey,Hawkins,(412) 344-8687,4585 Shimerville
Rd,Clarence,NY,14031,42.972075,-78.6592452 12,Phillip,Howard,(413) 269-4049,464 Main St
#101,Port Washington,NY,11050,40.8273756,-73.7009971 13,Shirley,Tyler,(434) 985-8943,114 Glann
Rd,Apalachin,NY,13732,42.0482515,-76.1229725 14,Aniyah,Jarvis,(440) 244-1808,87 N Middletown
Rd,Pearl River,NY,10965,41.0629,-74.0159 15,Alayna,Woodard,(478) 335-3704,70 W Red Oak Ln,West
Harrison,NY,10604,41.0162722,-73.7234926 16,Jermaine,Lambert,(508) 561-9836,24 Kellogg Rd,New
Hartford,NY,13413,43.0555739,-75.2793197 17,Harper,Gibbs,(239) 466-0238,Po Box
33,Cottekill,NY,12419,41.853392,-74.106082 18,Osvaldo,Graham,(252) 246-0816,6878 Sand Hill
Rd,East Syracuse,NY,13057,43.073215,-76.081448 19,Roberto,Wade,(270) 469-1211,3936 Holley
Rd,Moravia,NY,13118,42.713044,-76.481227 20,Kate,Mcguire,(270) 788-3080,6451 State 64 Rte
#3,Naples,NY,14512,42.707366,-77.380489 21,Dale,Andersen,(281) 480-5690,205 W Service
Rd,Champlain,NY,12919,44.9645392,-73.4470831 22,Lorelai,Burch,(302) 644-2133,1 Brewster
St,Glen Cove,NY,11542,40.865177,-73.633019 23,Amiyah,Flowers,(303) 223-0055,46600 Us
Interstate 81 Rte,Alexandria Bay,NY,13607,44.309626,-75.988365 24,Mckinley,Clements,(303) 918-
3230,200 Summit Lake Dr,Valhalla,NY,10595,41.101145,-73.778298 25,Marc,Gibson,(607) 203-
1233,25 Robinson St,Binghamton,NY,13901,42.107416,-75.901614 26,Kali,Norman,(607) 203-1400,1
Ely Park Blvd #APT 15,Binghamton,NY,13905,42.125866,-75.925026 27,Laci,Cain,(607) 203-1437,16
Zimmer Road,Kirkwood,NY,13795,42.066516,-75.792627 28,Mohammad,Perez,(607) 203-1652,71
Endicott Ave #APT 12,Johnson City,NY,13790,42.111894,-75.952187 29,Izabelle,Pham,(607) 204-
0392,434 State 369 Rte,Port Crane,NY,13833,42.185838,-75.823074 30,Kiley,Mays,(607) 204-
0870,244 Ballyhack Rd #14,Port Crane,NY,13833,42.175612,-75.814917 31,Peter,Trevino,(607) 205-
1374,125 Melbourne St.,Vestal,NY,13850,42.080254,-76.051124 32,Ani,Francis,(607) 208-4067,48
Caswell St,Afton,NY,13730,42.232065,-75.525674 33,Jared,Sheppard,(716) 386-3002,4709 430th
Rte,Bemus Point,NY,14712,42.162175,-79.39176 34,Dulce,Atkinson,(914) 576-2266,501 Pelham
Rd,New Rochelle,NY,10805,40.895449,-73.782602 35,Jayla,Beasley,(716) 526-1054,5010 474th
Rte,Ashville,NY,14710,42.096859,-79.375561 36,Dane,Donovan,(718) 545-3732,5014 31st
Ave,Woodside,NY,11377,40.756967,-73.909506 37,Brendon,Clay,(585) 322-7780,133 Cummings
Ave,Gainesville,NY,14066,42.664309,-78.085651 38,Asia,Nunez,(718) 426-1472,2407 Gilmore ,East
Elmhurst,NY,11369,40.766662,-73.869185 39,Dawson,Odonnell,(718) 342-2179,5019 H
Ave,Brooklyn,NY,11234,40.633245,-73.927591 40,Kyle,Collins,(315) 733-7078,502 Rockhaven
Rd,Utica,NY,13502,43.129184,-75.226726 41,Eliza,Hardin,(315) 331-8072,502 Sladen Place,West
Point,NY,10996,41.3993,-73.973003 42,Kasen,Klein,(518) 298-4581,2407 Lake Shore
Rd,Chazy,NY,12921,44.925561,-73.387373 43,Reuben,Bradford,(518) 298-4581,33 Lake Flats
Dr,Champlain,NY,12919,44.928092,-73.387884 44,Henry,Grimes,(518) 523-3990,2407 Main St,Lake
Placid,NY,12946,44.291487,-73.98474 45,Kyan,Livingston,(518) 585-7364,241 Alexandria
Ave,Ticonderoga,NY,12883,43.836553,-73.43155 46,Kaitlyn,Short,(516) 678-3189,241 Chance
Dr,Oceanside,NY,11572,40.638534,-73.63079 47,Damaris,Jacobs,(914) 664-5331,241 Claremont
Ave,Mount Vernon,NY,10552,40.919852,-73.827848 48,Alivia,Schroeder,(315) 469-4473,241
Lafayette Rd,Syracuse,NY,13205,42.996446,-76.12957 49,Bridget,Strong,(315) 298-4355,241 Maltby
Rd,Pulaski,NY,13142,43.584966,-76.136317 50,Francis,Lee,(585) 201-7021,166 Ross
St,Batavia,NY,14020,43.0031502,-78.17487 51,Makaila,Phelps,(585) 201-7422,58 S Main
St,Batavia,NY,14020,42.99941,-78.1939285 52,Jazlynn,Stephens,(585) 203-1087,1 Sinclair
Dr,Pittsford,NY,14534,43.084157,-77.545452 53,Ryann,Randolph,(585) 203-1519,331 Eaglehead
```

Rd,East Rochester,NY,14445,43.10785,-77.475552 54,Rosa,Baker,(585) 204-4011,42 Ossian St,Dansville,NY,14437,42.560761,-77.70088 55,Marcel,Barry,(585) 204-4013,42 Jefferson St,Dansville,NY,14437,42.557735,-77.702983 56,Dennis,Schmitt,(585) 204-4061,750 Dansville Mount Morris Rd,Dansville,NY,14437,42.584458,-77.741648 57,Cassandra,Kim,(585) 204-4138,3 Perine Ave APT1,Dansville,NY,14437,42.562865,-77.69661 58,Kolton,Jacobson,(585) 206-5047,4925 Upper Holly Rd,Holley,NY,14470,43.175957,-78.074465 59,Nathanael,Donovan,(718) 393-3501,9604 57th Ave,Corona,NY,11373,40.736077,-73.864858 60,Robert,Frazier,(718) 271-3067,300 56th Ave,Corona,NY,11373,40.735304,-73.873997 61,Jessie,Mora,(315) 405-8991,9607 Forsyth Loop,Watertown,NY,13603,44.036466,-75.833437 62,Martha,Rollins,(347) 242-2642,22 Main St,Corona,NY,11373,40.757727,-73.829331 63,Emely,Townsend,(718) 699-0751,60 Sanford Ave,Corona,NY,11373,40.755466,-73.831029 64,Kylie,Cooley,(347) 561-7149,9608 95th Ave,Ozone Park,NY,11416,40.687564,-73.845715 65,Wendy,Cameron,(585) 571-4185,9608 Union St,Scottsville,NY,14546,43.013327,-77.7907839 66,Kayley,Peterson,(718) 654-5027,961 E 230th St,Bronx,NY,10466,40.889275,-73.850555 67,Camden,Ochoa,(718) 760-8699,59 Vark St,Yonkers,NY,10701,40.929322,-73.89957 68,Priscilla,Castillo,(910) 326-7233,9359 Elm St,Chadwicks,NY,13319,43.024902,-75.26886 69,Dana,Schultz,(913) 322-4580,99 Washington Ave,Hastings on Hudson,NY,10706,40.99265,-73.879748 70,Blaze,Medina,(914) 207-0015,60 Elliott Ave,Yonkers,NY,10705,40.921498,-73.896682 71,Finnegan,Tucker,(914) 207-0015,90 Hillside Drive,Yonkers,NY,10705,40.922514,-73.892911 72,Pranav,Palmer,(914) 214-8376,5 Bruce Ave,Harrison,NY,10528,40.970916,-73.711493 73,Kolten,Wong,(914) 218-8268,70 Barker St,Mount Kisco,NY,10549,41.211993,-73.723202 74,Jasiah,Vazquez,(914) 231-5199,30 Broadway,Dobbs Ferry,NY,10522,41.004629,-73.879825 75,Lamar,Pierce,(914) 232-0380,68 Ridge Rd,Katonah,NY,10536,41.256662,-73.707964 76,Carla,Coffey,(914) 232-0469,197 Beaver Dam Rd,Katonah,NY,10536,41.247934,-73.664363 77,Brooklynn,Harmon,(716) 595-3227,8084 Glasgow Rd,Cassadega,NY,14718,42.353861,-79.329558 78,Raquel,Hodges,(585) 398-8125,809 County Road ,Victor,NY,14564,43.011745,-77.398806 79,Jerimiah,Gardner,(585) 787-9127,809 Houston Rd,Webster,NY,14580,43.224204,-77.491353 80,Clarence,Hammond,(720) 746-1619,809 Pierpont Ave,Piermont,NY,10968,41.0491181,-73.918622 81,Rhys,Gill,(518) 427-7887,81 Columbia St,Albany,NY,12210,42.652824,-73.752096 82,Edith,Parrish,(845) 452-7621,81 Glenwood Ave,Poughkeepsie,NY,12603,41.691058,-73.910829 83,Kobe,Mcintosh,(845) 371-1101,81 Heitman Dr,Spring Valley,NY,10977,41.103227,-74.054396 84,Ayden,Waters,(516) 796-2722,81 Kingfisher Rd,Levittown,NY,11756,40.738939,-73.52826 85,Francis,Rogers,(631) 427-7728,81 Knollwood Ave,Huntington,NY,11743,40.864905,-73.426107 86,Jaden,Landry,(716) 496-4038,12839 39th Rte,Chaffee,NY,14030,43.527396,-73.462786 87,Giancarlo,Campos,(518) 885-5717,1284 Saratoga Rd,Ballston Spa,NY,12020,42.968594,-73.862847 88,Eduardo,Contreras,(716) 285-8987,1285 Saunders Sett Rd,Niagara Falls,NY,14305,43.122963,-79.029274 89,Gabriela,Davidson,(716) 267-3195,1286 Mee Rd,Falconer,NY,14733,42.147339,-79.137976 90,Evangeline,Case,(518) 272-9435,1287 2nd Ave,Watervliet,NY,12189,42.723132,-73.703818 91,Tyrone,Ellison,(518) 843-4691,1287 Midline Rd,Amsterdam,NY,12010,42.9730876,-74.1700608 92,Bryce,Bass,(518) 943-9549,1288 Leeds Athens Rd,Athens,NY,12015,42.259381,-73.876897 93,Londyn,Butler,(518) 922-7095,129 Argersinger Rd,Fultonville,NY,12072,42.910969,-74.441917 94,Graham,Becker,(607) 655-1318,129 Baker Rd,Windsor,NY,13865,42.107271,-75.66408 95,Rolando,Fitzgerald,(315) 465-4166,17164 County 90 Rte,Mannsville,NY,13661,43.713443,-76.06232 96,Grant,Hoover,(518) 692-8363,1718 County 113 Rte,Schaghticote,NY,12154,42.900648,-73.585036 97,Mark,Goodwin,(631) 584-6761,172 Cambon Ave,Saint James,NY,11780,40.871152,-73.146032 98,Deacon,Cantu,(845) 221-7940,172 Carpenter Rd,Hopewell Junction,NY,12533,41.57388,-73.77609 99,Tristian,Walsh,(516) 997-4750,172 E Cabot Ln,Westbury,NY,11590,40.7480397,-73.54819 100,Abram,Alexander,(631) 588-3817,172 Lorenzo Cir,Ronkonkoma,NY,11779,40.837123,-73.09367 101,Lesly,Bush,(516) 489-3791,172 Nassau Blvd,Garden City,NY,11530,40.71147,-73.660753 102,Pamela,Espinoza,(716) 201-1520,172 Niagara St ,Lockport,NY,14094,43.169871,-78.70093 103,Bryanna,Newton,(914) 328-4332,172 Warren Ave,White Plains,NY,10603,41.047207,-73.79572 104,Marcelo,Schmitt,(315) 393-4432,319 Mansion Ave,Ogdensburg,NY,13669,44.690246,-75.49992 105,Layton,Valenzuela,(631) 676-2113,319 Singingwood Dr,Holbrook,NY,11741,40.801391,-73.058993 106,Roderick,Rocha,(518) 671-6037,319 Warren St,Hudson,NY,12534,42.252527,-73.790629 107,Camryn,Terrell,(315) 635-1680,3192 Olive Dr,Baldinsville,NY,13027,43.136843,-76.260303 108,Summer,Callahan,(585) 394-4195,3192 Smith Road,Canandaigua,NY,14424,42.875457,-77.228039 109,Pierre,Novak,(716) 665-2524,3194 Falconer

```
Kimball Stand Rd,Falconer,NY,14733,42.138439,-79.211091 110,Kennedi,Fry,(315) 543-2301,32
College Rd,Selden,NY,11784,40.861624,-73.04757 111,Wyatt,Pruitt,(716) 681-4042,277 Ransom
Rd,Lancaster ,NY,14086,42.87702,-78.591302 112,Lilly,Jensen,(631) 841-0859,2772 Schliegel
Blvd,Amityville,NY,11701,40.708021,-73.413015 113,Tristin,Hardin,(631) 920-0927,278 Fulton
Street,West Babylon,NY,11704,40.733578,-73.357321 114,Tanya,Stafford,(716) 484-0771,278
Sampson St,Jamestown,NY,14701,42.0797,-79.247805 115,Paris,Cordova,(607) 589-4857,278 Washburn
Rd,Spencer,NY,14883,42.225046,-76.510257 116,Alfonso,Morse,(718) 359-5582,200 Colden
St,Flushing,NY,11355,40.750403,-73.822752 117,Maurice,Hooper,(315) 595-6694,4435 Italy Hill
Rd,Branchport,NY,14418,42.597957,-77.199267 118,Iris,Wolf,(607) 539-7288,444 Harford
Rd,Brooktondale,NY,14817,42.392164,-76.30756 ];
```

KMeans2D - 图表函数

KMeans2D() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的集群 id。集群算法使用的列分别由参数 coordinate_1 和 coordinate_2 确定。二者都是聚合型。创建的集群数由 num_clusters 参数确定。数据可以通过规范参数进行规范化。

KMeans2D 每个数据点返回一个值。返回值是一个双重值，是与每个数据点分配到的集群相对应的整数值。

语法：

```
KMeans2D(num_clusters, coordinate_1, coordinate_2 [, norm])
```

返回数据类型： 双

参数：

参数

参数	说明
num_clusters	指定集群数的整数。
coordinate_1	计算第一个坐标的聚合，通常是散点图的 x 轴，可以从图表中生成。另外的参数 coordinate_2 计算第二个坐标。
norm	在 K-均值聚类之前应用于数据集的可选规范化方法。 可能的值： 对于规范化为 0 或 'none' 对于 z-score 规范化为 1 或 'zscore' 对于 min-max 规范化为 2 或 'minmax' 如果未提供参数或提供的参数不正确，则不应用规范化。 Z-score 基于特征均值和标准差对数据进行标准化。Z-score 并不能保证每个特征具有相同的尺度，但在处理异常值时，它是一种比 min-max 更好的方法。 Min-max 规范化通过获取每个数据点的最小值和最大值并重新计算每个数据点，确保特征具有相同的比例。

示例：图表表达式
在本示例中，我们使用 *Iris* 数据集创建散点图，然后使用 KMeans 按表达式为数据着色。

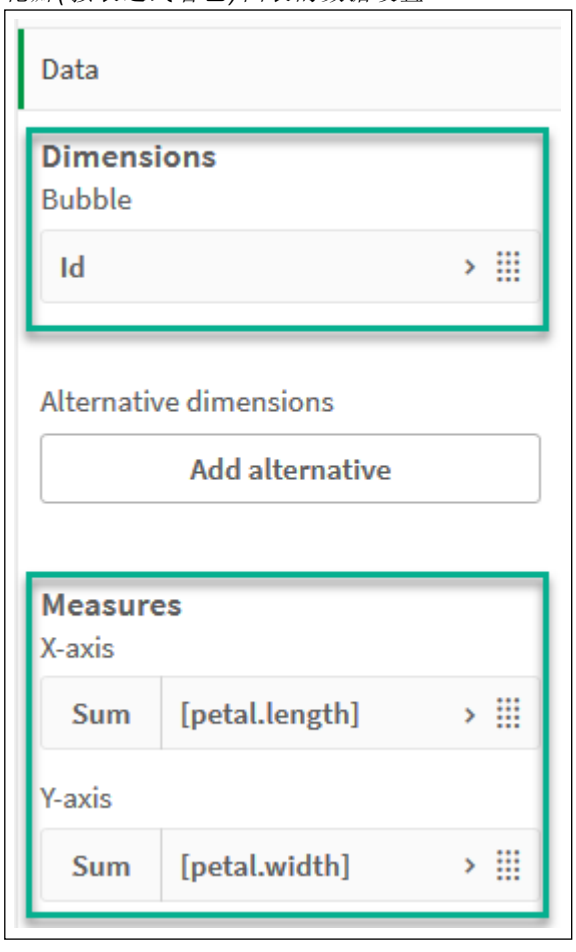
我们还为 *num_clusters* 参数创建一个变量，然后使用变量输入框来更改集群的数量。

Iris 数据集以多种格式公开可用。我们已将数据作为内联表提供，以便使用 Qlik Sense 中的数据加载编辑器进行加载。注意，我们在本例的数据表中添加了一个 *Id* 列。

在 Qlik Sense 中加载数据后，我们将执行以下操作：

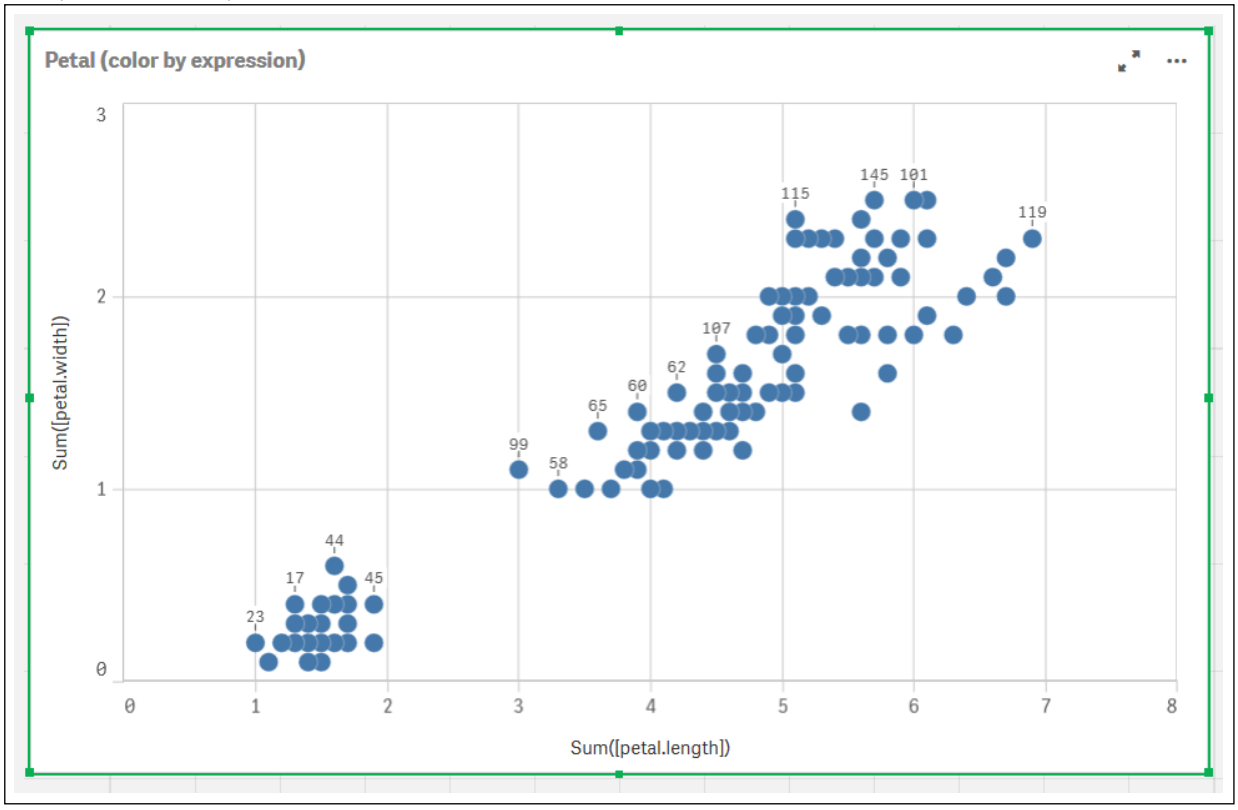
1. 将**散点图**图表拖动至新的工作表。将图表命名为*花瓣(按表达式着色)*。
2. 创建变量以指定集群数。对于变量 **Name**，输入 *KmeansPetalClusters*。对于变量 **Definition**，输入 *=2*。
3. 配置图表的**数据**：
    i. 在**维度**下，选择**气泡**的字段的 *id*。输入标签的集群 Id。
    ii. 在**度量**下，选择 **X 轴**表达式的 *Sum([petal.length])*。
    iii. 在**度量**下，选择 **Y 轴**表达式的 *Sum([petal.width])*。

    花瓣(按表达式着色)图表的数据设置



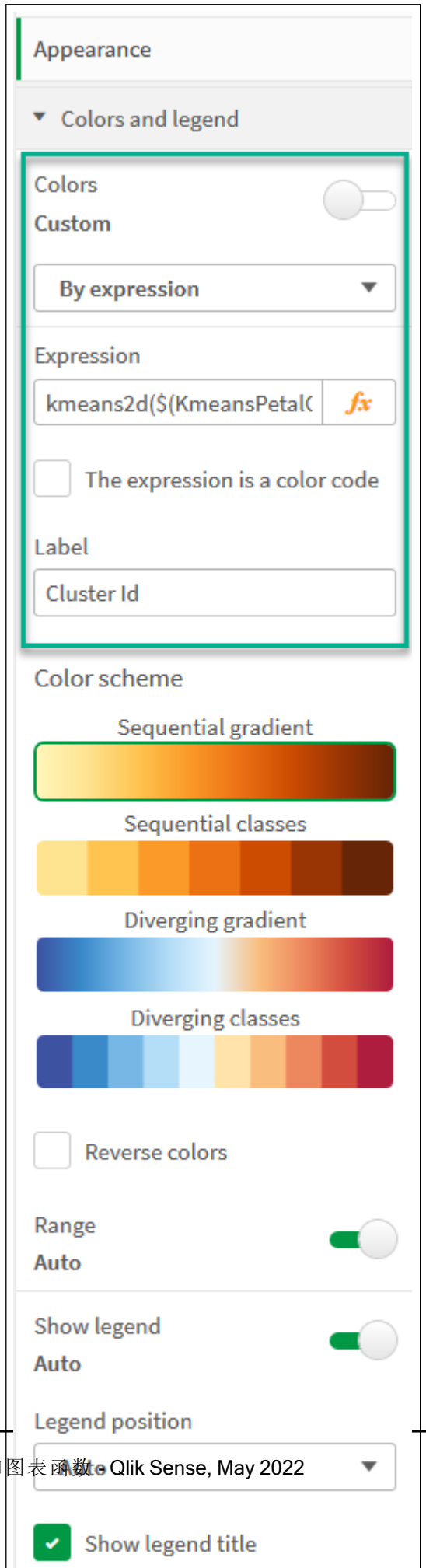
数据点在图表上绘制。

花瓣(按表达式着色)图表上的数据点

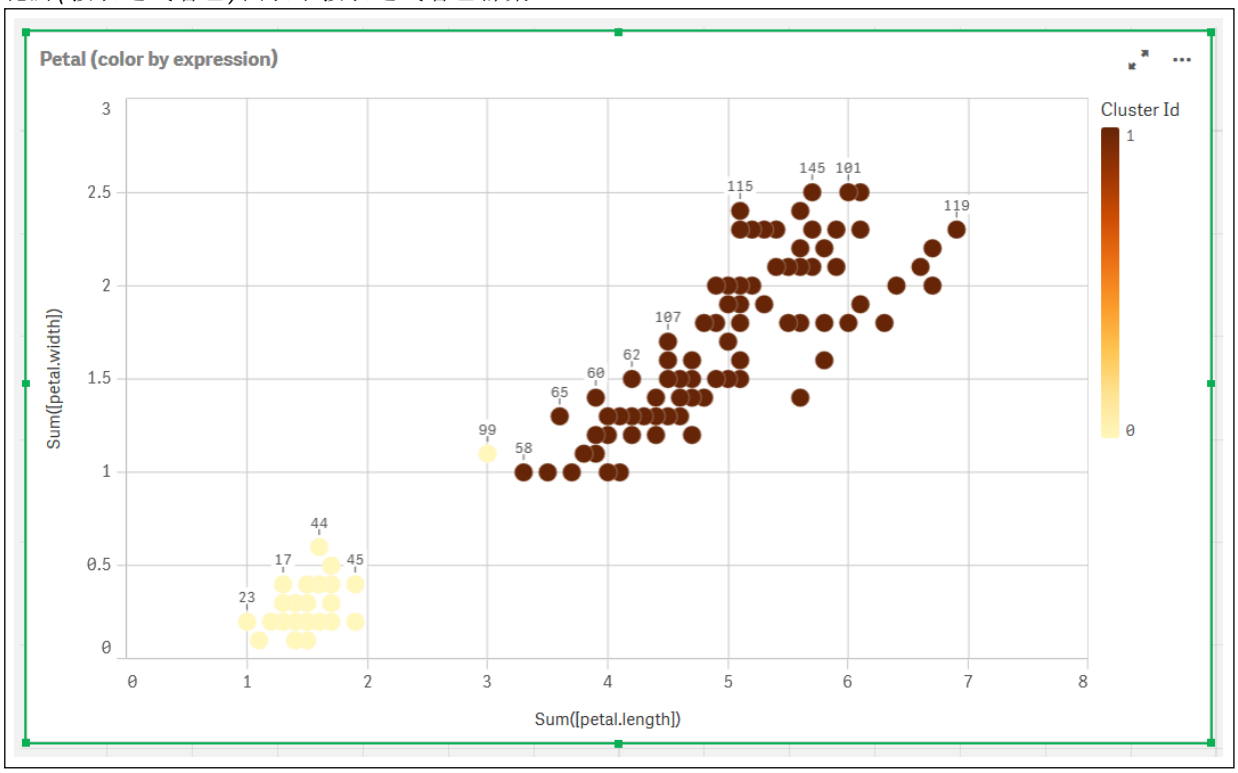


4. 配置图表的**外观**:
   i. 在**颜色和图例**下,为**颜色**选择**自定义**。
   ii. 选择侧向以**按表达式**对图表着色。
   iii. 为**表达式**输入以下内容: *kmeans2d($(KmeansPetalClusters), Sum([petal.length]), Sum([petal.width]))*
   注意 *KmeansPetalClusters* 是我们设置为 *2* 的变量。
   或者输入以下内容: *kmeans2d(2, Sum([petal.length]), Sum([petal.width]))*
   iv. 取消选择**表达式是一个颜色代码**。
   v. 为**标签**输入以下内容: *Cluster Id*

花瓣(按表达式着色)图表的外观设置

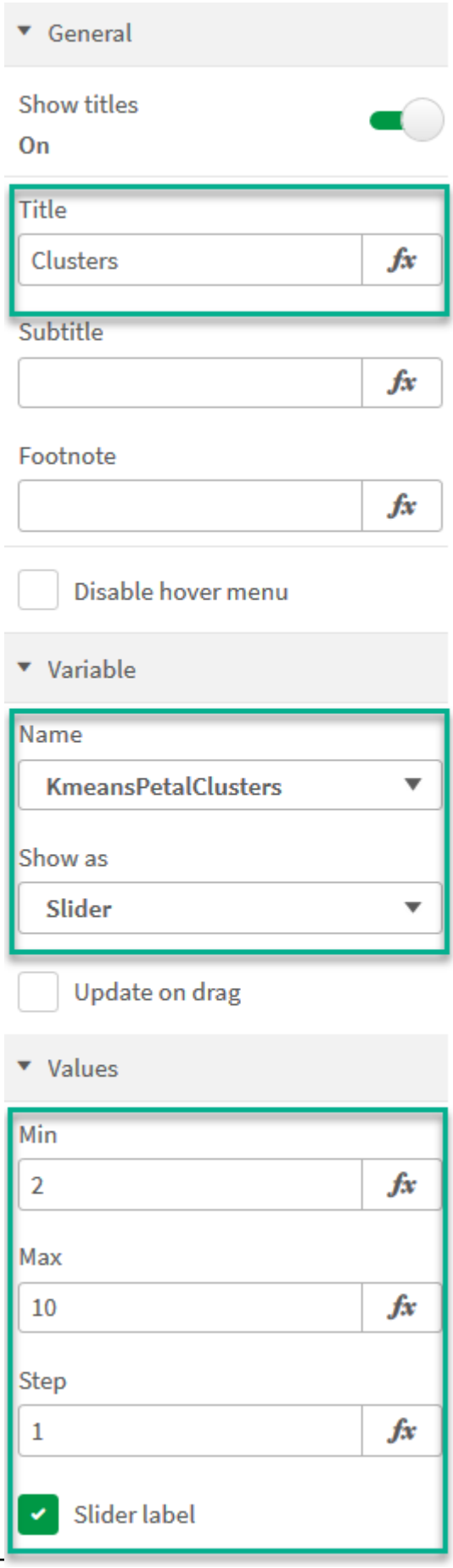

Appearance
Colors and legend
Colors
Custom
By expression
Expression
kmeans2d($(KmeansPetal(
The expression is a color code
Label
Cluster Id
Color scheme
Sequential gradient
Sequential classes
Diverging gradient
Diverging classes
Reverse colors
Range
Auto
Show legend
Auto
Legend position
Auto
Show legend title

图表上的两个集群按 KMeans 表达式着色。
花瓣(按表达式着色)图表中按表达式着色群集。



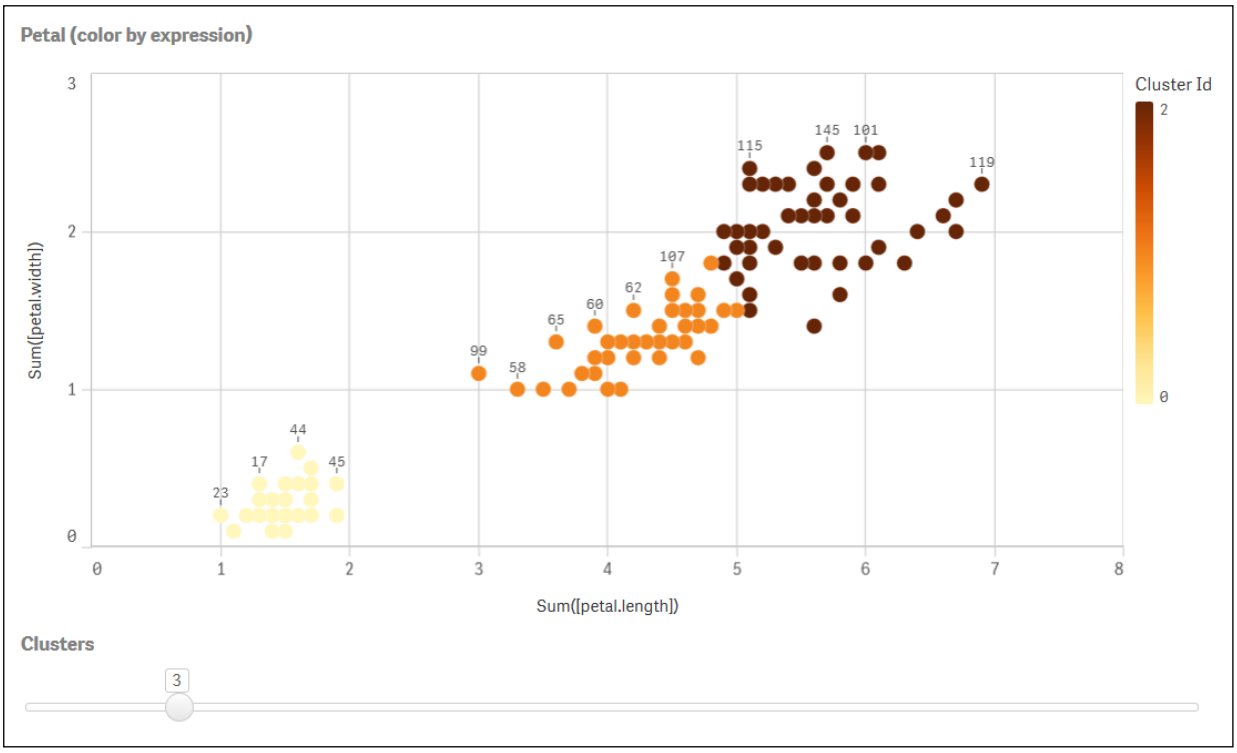
5. 对集群的数目添加**变量输入**框。
    i. 在**资产**面板中的**自定义对象**下，选择 **Qlik 仪表板捆绑**。如果我们不能访问仪表板捆绑，我们仍然可以使用我们创建的变量更改集群的数量，或者直接作为表达式中的整数。
    ii. 将**变量输入**框拖动到工作表上。
    iii. 在**外观**下，单击**常规**。
    iv. 为**标题**输入以下内容：*集群*
    v. 单击**变量**。
    vi. 为**名称**选择以下变量：*KmeansPetalClusters*。
    vii. 为**显示方式**选择**滑块**。
    viii. 选择**值**，并根据需要配置设置。

Clusters 变量输入框的外观。


General
Show titles
On
Title
Clusters
Subtitle
Footnote
Disable hover menu
Variable
Name
KmeansPetalClusters
Show as
Slider
Update on drag
Values
Min
2
Max
10
Step
1
Slider label

完成编辑后，我们现在可以使用 *Clusters* 变量输入框中的滑块更改集群的数量。

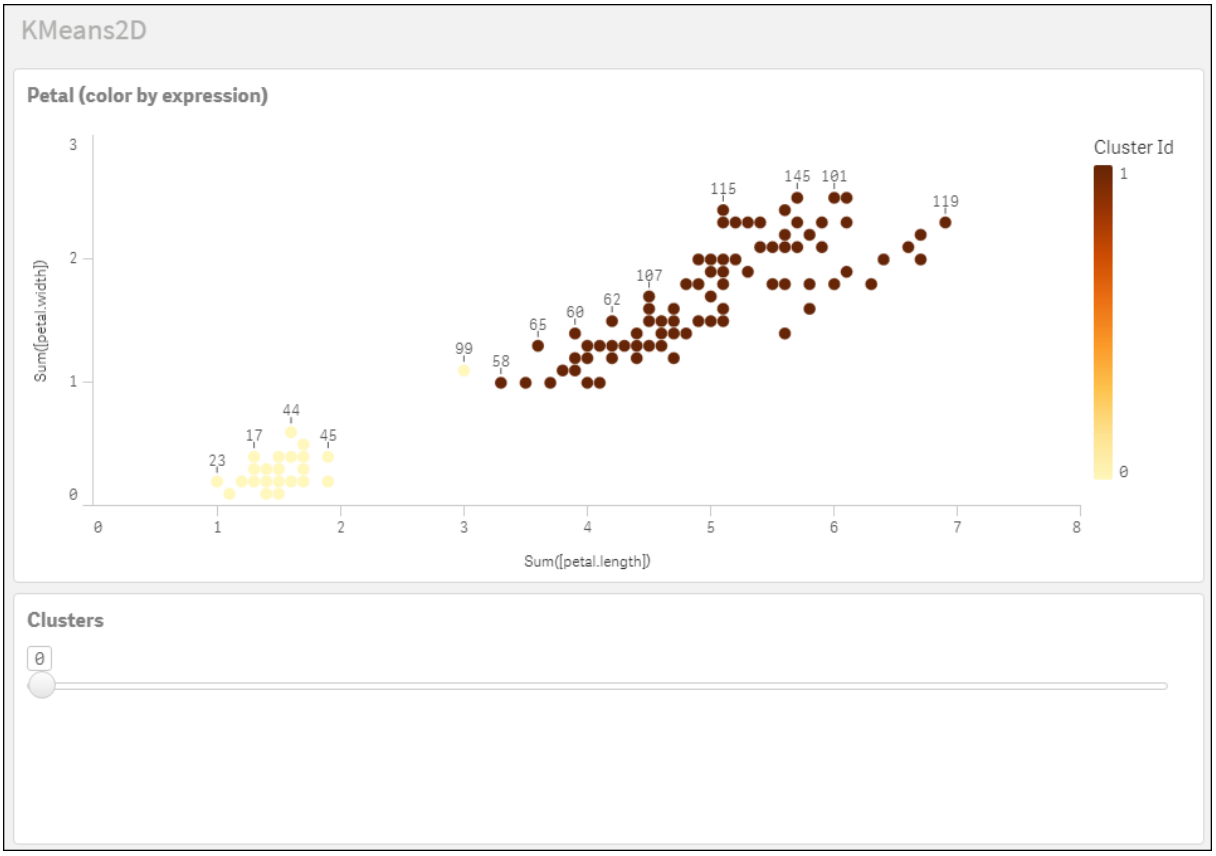
花瓣(按表达式着色)图表中按表达式着色群集。



自动聚合

均值函数支持使用名为深度差 (DeD) 的方法支持自动聚合。如果用户为集群数设置 0，则会为该数据集确定最优集群数。注意，虽然没有显式返回集群数 (*k*) 的整数，但它是在均值算法中计算的。例如，如果在函数中为 *KmeansPetalClusters* 的值指定了 0 或通过变量输入框进行设置，则会根据最佳的集群数自动计算数据集的簇分配。

当 (k) 设置为 0 时，均值深度差分方法确定最佳集群数



Iris 数据集：Qlik Sense 中数据加载编辑器的内联加载

```
IrisData: Load * Inline [ sepal.length, sepal.width, petal.length, petal.width, variety, id
5.1, 3.5, 1.4, 0.2, Setosa, 1 4.9, 3, 1.4, 0.2, Setosa, 2 4.7, 3.2, 1.3, 0.2, Setosa, 3 4.6,
3.1, 1.5, 0.2, Setosa, 4 5, 3.6, 1.4, 0.2, Setosa, 5 5.4, 3.9, 1.7, 0.4, Setosa, 6 4.6, 3.4,
1.4, 0.3, Setosa, 7 5, 3.4, 1.5, 0.2, Setosa, 8 4.4, 2.9, 1.4, 0.2, Setosa, 9 4.9, 3.1, 1.5,
0.1, Setosa, 10 5.4, 3.7, 1.5, 0.2, Setosa, 11 4.8, 3.4, 1.6, 0.2, Setosa, 12 4.8, 3, 1.4,
0.1, Setosa, 13 4.3, 3, 1.1, 0.1, Setosa, 14 5.8, 4, 1.2, 0.2, Setosa, 15 5.7, 4.4, 1.5, 0.4,
Setosa, 16 5.4, 3.9, 1.3, 0.4, Setosa, 17 5.1, 3.5, 1.4, 0.3, Setosa, 18 5.7, 3.8, 1.7, 0.3,
Setosa, 19 5.1, 3.8, 1.5, 0.3, Setosa, 20 5.4, 3.4, 1.7, 0.2, Setosa, 21 5.1, 3.7, 1.5, 0.4,
Setosa, 22 4.6, 3.6, 1, 0.2, Setosa, 23 5.1, 3.3, 1.7, 0.5, Setosa, 24 4.8, 3.4, 1.9, 0.2,
Setosa, 25 5, 3, 1.6, 0.2, Setosa, 26 5, 3.4, 1.6, 0.4, Setosa, 27 5.2, 3.5, 1.5, 0.2, Setosa,
28 5.2, 3.4, 1.4, 0.2, Setosa, 29 4.7, 3.2, 1.6, 0.2, Setosa, 30 4.8, 3.1, 1.6, 0.2, Setosa,
31 5.4, 3.4, 1.5, 0.4, Setosa, 32 5.2, 4.1, 1.5, 0.1, Setosa, 33 5.5, 4.2, 1.4, 0.2, Setosa,
34 4.9, 3.1, 1.5, 0.1, Setosa, 35 5, 3.2, 1.2, 0.2, Setosa, 36 5.5, 3.5, 1.3, 0.2, Setosa, 37
4.9, 3.1, 1.5, 0.1, Setosa, 38 4.4, 3, 1.3, 0.2, Setosa, 39 5.1, 3.4, 1.5, 0.2, Setosa, 40 5,
3.5, 1.3, 0.3, Setosa, 41 4.5, 2.3, 1.3, 0.3, Setosa, 42 4.4, 3.2, 1.3, 0.2, Setosa, 43 5,
3.5, 1.6, 0.6, Setosa, 44 5.1, 3.8, 1.9, 0.4, Setosa, 45 4.8, 3, 1.4, 0.3, Setosa, 46 5.1,
3.8, 1.6, 0.2, Setosa, 47 4.6, 3.2, 1.4, 0.2, Setosa, 48 5.3, 3.7, 1.5, 0.2, Setosa, 49 5,
3.3, 1.4, 0.2, Setosa, 50 7, 3.2, 4.7, 1.4, Versicolor, 51 6.4, 3.2, 4.5, 1.5, Versicolor, 52
6.9, 3.1, 4.9, 1.5, Versicolor, 53 5.5, 2.3, 4, 1.3, Versicolor, 54 6.5, 2.8, 4.6, 1.5,
Versicolor, 55 5.7, 2.8, 4.5, 1.3, Versicolor, 56 6.3, 3.3, 4.7, 1.6, Versicolor, 57 4.9, 2.4,
3.3, 1, Versicolor, 58 6.6, 2.9, 4.6, 1.3, Versicolor, 59 5.2, 2.7, 3.9, 1.4, Versicolor, 60
5, 2, 3.5, 1, Versicolor, 61 5.9, 3, 4.2, 1.5, Versicolor, 62 6, 2.2, 4, 1, Versicolor, 63
6.1, 2.9, 4.7, 1.4, Versicolor, 64 5.6, 2.9, 3.6, 1.3, Versicolor, 65 6.7, 3.1, 4.4, 1.4,
Versicolor, 66 5.6, 3, 4.5, 1.5, Versicolor, 67 5.8, 2.7, 4.1, 1, Versicolor, 68 6.2, 2.2,
4.5, 1.5, Versicolor, 69 5.6, 2.5, 3.9, 1.1, Versicolor, 70 5.9, 3.2, 4.8, 1.8, Versicolor, 71
6.1, 2.8, 4, 1.3, Versicolor, 72 6.3, 2.5, 4.9, 1.5, Versicolor, 73 6.1, 2.8, 4.7, 1.2,
```

```
Versicolor, 74 6.4, 2.9, 4.3, 1.3, Versicolor, 75 6.6, 3, 4.4, 1.4, Versicolor, 76 6.8, 2.8,
4.8, 1.4, Versicolor, 77 6.7, 3, 5, 1.7, Versicolor, 78 6, 2.9, 4.5, 1.5, Versicolor, 79 5.7,
2.6, 3.5, 1, Versicolor, 80 5.5, 2.4, 3.8, 1.1, Versicolor, 81 5.5, 2.4, 3.7, 1, Versicolor,
82 5.8, 2.7, 3.9, 1.2, Versicolor, 83 6, 2.7, 5.1, 1.6, Versicolor, 84 5.4, 3, 4.5, 1.5,
Versicolor, 85 6, 3.4, 4.5, 1.6, Versicolor, 86 6.7, 3.1, 4.7, 1.5, Versicolor, 87 6.3, 2.3,
4.4, 1.3, Versicolor, 88 5.6, 3, 4.1, 1.3, Versicolor, 89 5.5, 2.5, 4, 1.3, Versicolor, 90
5.5, 2.6, 4.4, 1.2, Versicolor, 91 6.1, 3, 4.6, 1.4, Versicolor, 92 5.8, 2.6, 4, 1.2,
Versicolor, 93 5, 2.3, 3.3, 1, Versicolor, 94 5.6, 2.7, 4.2, 1.3, Versicolor, 95 5.7, 3, 4.2,
1.2, Versicolor, 96 5.7, 2.9, 4.2, 1.3, Versicolor, 97 6.2, 2.9, 4.3, 1.3, Versicolor, 98 5.1,
2.5, 3, 1.1, Versicolor, 99 5.7, 2.8, 4.1, 1.3, Versicolor, 100 6.3, 3.3, 6, 2.5, Virginica,
101 5.8, 2.7, 5.1, 1.9, Virginica, 102 7.1, 3, 5.9, 2.1, Virginica, 103 6.3, 2.9, 5.6, 1.8,
Virginica, 104 6.5, 3, 5.8, 2.2, Virginica, 105 7.6, 3, 6.6, 2.1, Virginica, 106 4.9, 2.5,
4.5, 1.7, Virginica, 107 7.3, 2.9, 6.3, 1.8, Virginica, 108 6.7, 2.5, 5.8, 1.8, Virginica, 109
7.2, 3.6, 6.1, 2.5, Virginica, 110 6.5, 3.2, 5.1, 2, Virginica, 111 6.4, 2.7, 5.3, 1.9,
Virginica, 112 6.8, 3, 5.5, 2.1, Virginica, 113 5.7, 2.5, 5, 2, Virginica, 114 5.8, 2.8, 5.1,
2.4, Virginica, 115 6.4, 3.2, 5.3, 2.3, Virginica, 116 6.5, 3, 5.5, 1.8, Virginica, 117 7.7,
3.8, 6.7, 2.2, Virginica, 118 7.7, 2.6, 6.9, 2.3, Virginica, 119 6, 2.2, 5, 1.5, Virginica,
120 6.9, 3.2, 5.7, 2.3, Virginica, 121 5.6, 2.8, 4.9, 2, Virginica, 122 7.7, 2.8, 6.7, 2,
Virginica, 123 6.3, 2.7, 4.9, 1.8, Virginica, 124 6.7, 3.3, 5.7, 2.1, Virginica, 125 7.2, 3.2,
6, 1.8, Virginica, 126 6.2, 2.8, 4.8, 1.8, Virginica, 127 6.1, 3, 4.9, 1.8, Virginica, 128
6.4, 2.8, 5.6, 2.1, Virginica, 129 7.2, 3, 5.8, 1.6, Virginica, 130 7.4, 2.8, 6.1, 1.9,
Virginica, 131 7.9, 3.8, 6.4, 2, Virginica, 132 6.4, 2.8, 5.6, 2.2, Virginica, 133 6.3, 2.8,
5.1, 1.5, Virginica, 134 6.1, 2.6, 5.6, 1.4, Virginica, 135 7.7, 3, 6.1, 2.3, Virginica, 136
6.3, 3.4, 5.6, 2.4, Virginica, 137 6.4, 3.1, 5.5, 1.8, Virginica, 138 6, 3, 4.8, 1.8,
Virginica, 139 6.9, 3.1, 5.4, 2.1, Virginica, 140 6.7, 3.1, 5.6, 2.4, Virginica, 141 6.9, 3.1,
5.1, 2.3, Virginica, 142 5.8, 2.7, 5.1, 1.9, Virginica, 143 6.8, 3.2, 5.9, 2.3, Virginica, 144
6.7, 3.3, 5.7, 2.5, Virginica, 145 6.7, 3, 5.2, 2.3, Virginica, 146 6.3, 2.5, 5, 1.9,
Virginica, 147 6.5, 3, 5.2, 2, Virginica, 148 6.2, 3.4, 5.4, 2.3, Virginica, 149 5.9, 3, 5.1,
1.8, Virginica, 150 ];
```

KMeansND - 图表函数

KMeansND() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的集群 id。集群算法使用的列由参数 coordinate_1 和 coordinate_2 等确定（可达 n 列）。这些都是聚合型。创建的集群数由 num_clusters 参数确定。

KMeansND 每个数据点返回一个值。返回值是一个双重值，是与每个数据点分配到的集群相对应的整数值。

语法：

```
KMeansND(num_clusters, num_iter, coordinate_1, coordinate_2 [,coordinate_3 [,
...]])
```

返回数据类型： 双

参数：

参数

参数	说明
num_clusters	指定集群数的整数。

参数	说明
num_iter	使用重新初始化的集群中心进行集群的迭代次数。
coordinate_1	计算第一个坐标的聚合，通常是散点图的 x 轴，可以从图表中生成。另外的参数计算第二、第三和第四个坐标等。

示例：图表表达式
在本示例中，我们使用 *Iris* 数据集创建散点图，然后使用 KMeans 按表达式为数据着色。

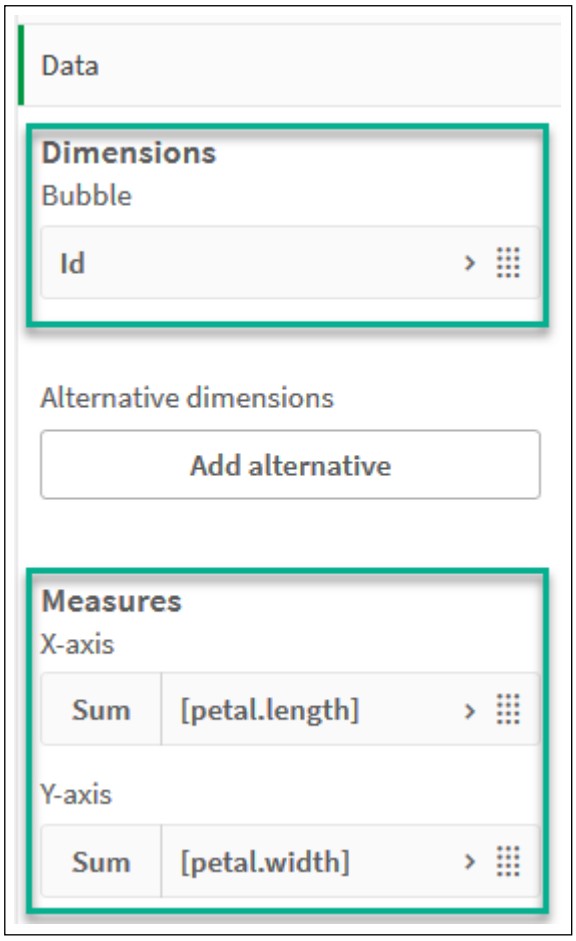
我们还为 *num_clusters* 参数创建一个变量，然后使用变量输入框来更改集群的数量。

我们还为 *num_iter* 参数创建一个变量，然后使用第二变量输入框来更改迭代的数量。

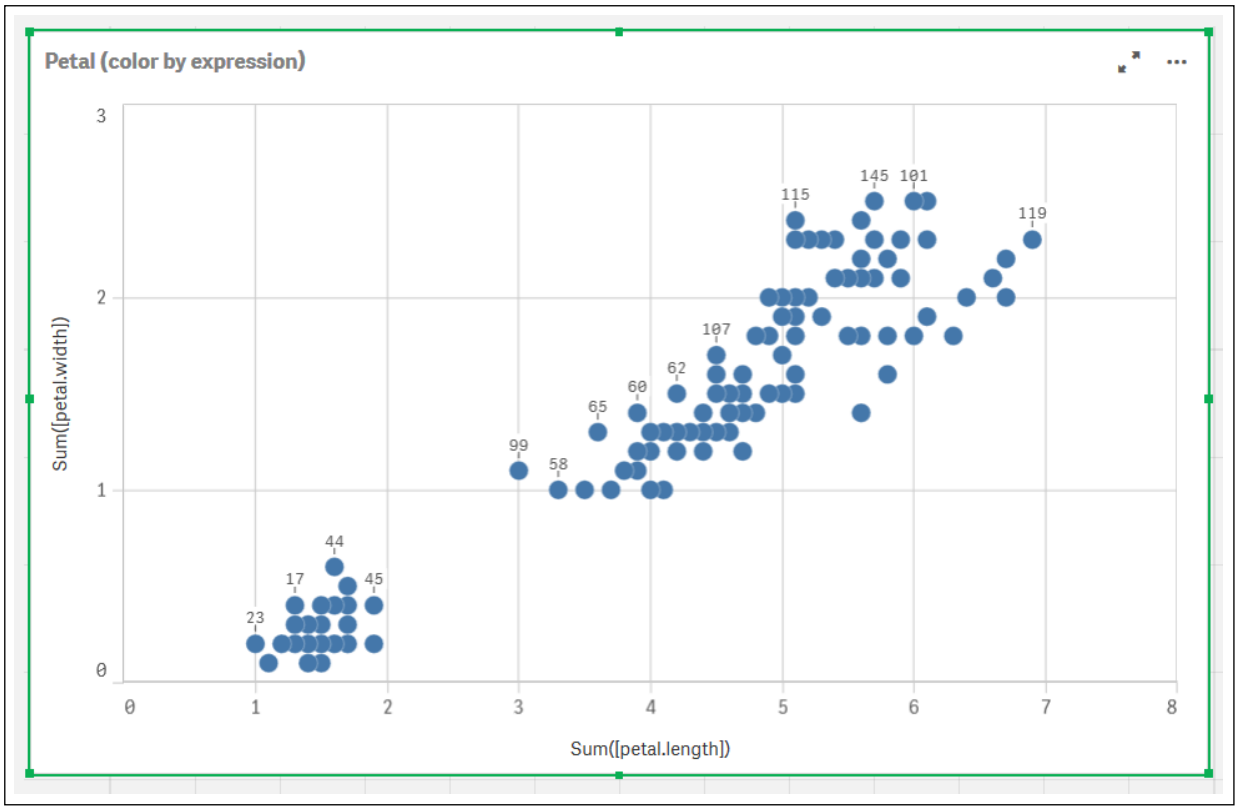
Iris 数据集以多种格式公开可用。我们已将数据作为内联表提供，以便使用 Qlik Sense 中的数据加载编辑器进行加载。注意，我们在本例的数据表中添加了一个 *Id* 列。

在 Qlik Sense 中加载数据后，我们将执行以下操作：

1. 将**散点图**图表拖动至新的工作表。将图表命名为*花瓣(按表达式着色)*。
2. 创建变量以指定集群数。对于变量 **Name**，输入 *KmeansPetalClusters*。对于变量 **Definition**，输入 *=2*。
3. 创建变量以指定迭代数。对于变量 **Name**，输入 *KmeansNumberIterations*。对于变量 **Definition**，输入 *=1*。
4. 配置图表的**数据**：
    i. 在**维度**下，选择**气泡**的字段的 *id*。输入标签的集群 Id。
    ii. 在**度量**下，选择 **X 轴**表达式的 *Sum([petal.length])*。
    iii. 在**度量**下，选择 **Y 轴**表达式的 *Sum([petal.width])*。
    花瓣(按表达式着色)图表的数据设置

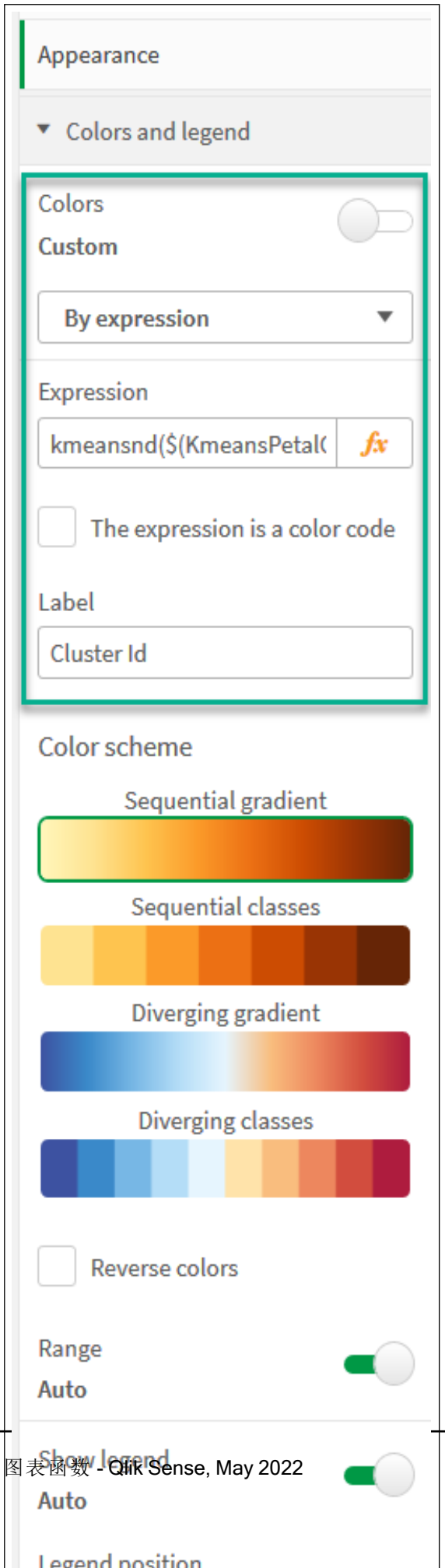


数据点在图表上绘制。
花瓣(按表达式着色)图表上的数据点

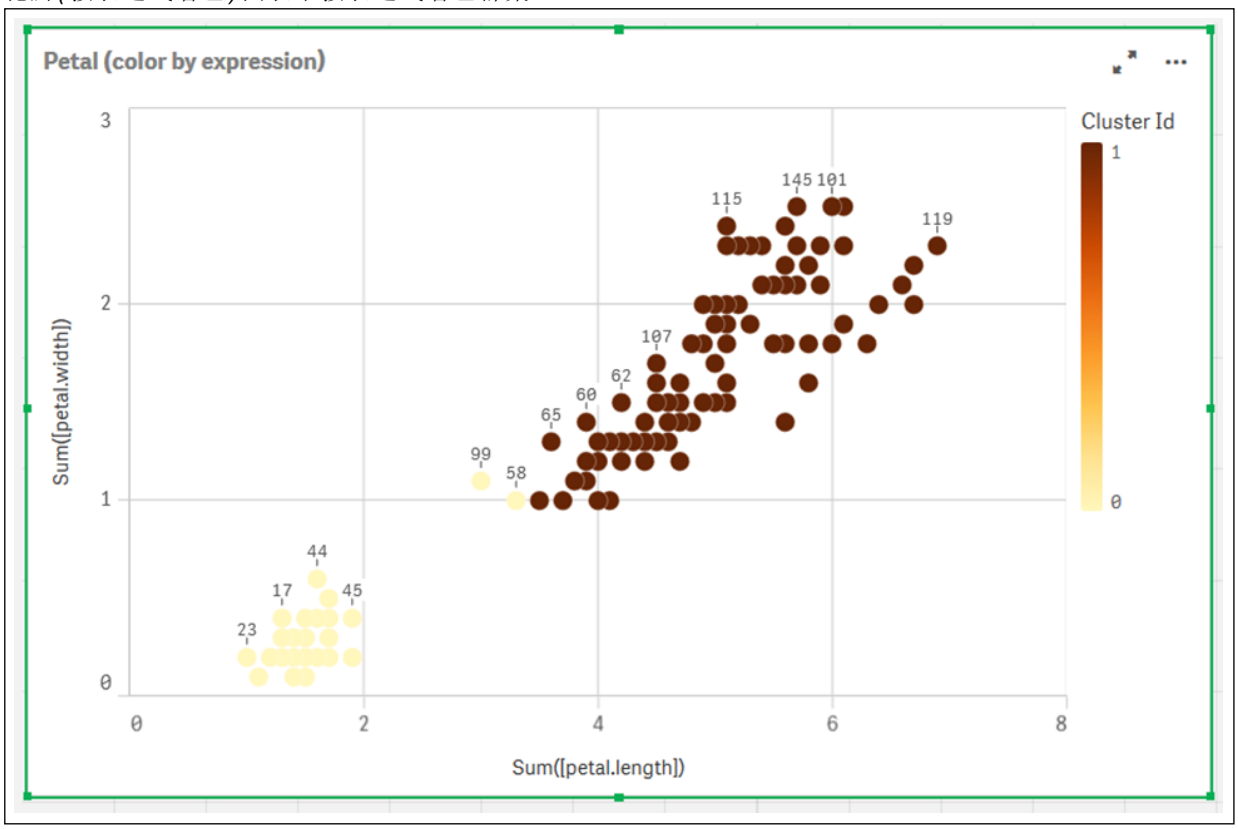


5. 配置图表的**外观**：
   i. 在**颜色和图例**下，为**颜色**选择**自定义**。
   ii. 选择侧向以**按表达式**对图表着色。
   iii. Enter the following for **Expression**: *kmeansnd ($(KmeansPetalClusters),$(KmeansNumberIterations), Sum([petal.length]), Sum ([petal.width]),Sum([sepal.length]), Sum([sepal.width]))*
   注意 *KmeansPetalClusters* 是我们设置为 *2* 的变量。*KmeansNumberIterations* 是设置为 *1* 的变量。
   或者输入以下内容：*kmeansnd(2, 2, Sum([petal.length]), Sum([petal.width]),Sum ([sepal.length]), Sum([sepal.width]))*
   iv. 取消选择**颜色代码的表达式**。
   v. 为**标签**输入以下内容：*Cluster Id*

花瓣(按表达式着色)图表的外观设置

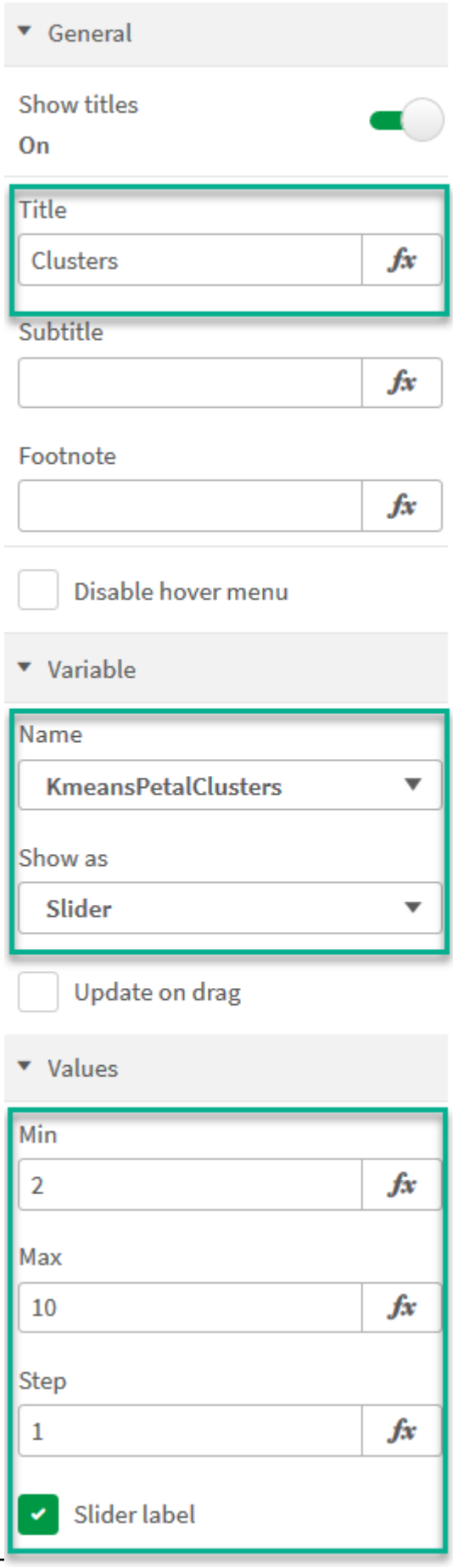

Appearance
Colors and legend
Colors
Custom
By expression
Expression
kmeansnd($(KmeansPetal(
fx
The expression is a color code
Label
Cluster Id
Color scheme
Sequential gradient
Sequential classes
Diverging gradient
Diverging classes
Reverse colors
Range
Auto
Show legend
Auto
Legend position

图表上的两个集群按 KMeans 表达式着色。
花瓣(按表达式着色)图表中按表达式着色群集。



6. 对集群的数目添加**变量输入**框。
    i. 在**资产**面板中的**自定义对象**下，选择 **Qlik 仪表板捆绑**。如果我们不能访问仪表板捆绑，我们仍然可以使用我们创建的变量更改集群的数量，或者直接作为表达式中的整数。
    ii. 将**变量输入**框拖动到工作表上。
    iii. 在**外观**下，单击**常规**。
    iv. 为**标题**输入以下内容：*集群*
    v. 单击**变量**。
    vi. 为**名称**选择以下变量：*KmeansPetalClusters*。
    vii. 为**显示方式**选择**滑块**。
    viii. 选择**值**，并根据需要配置设置。

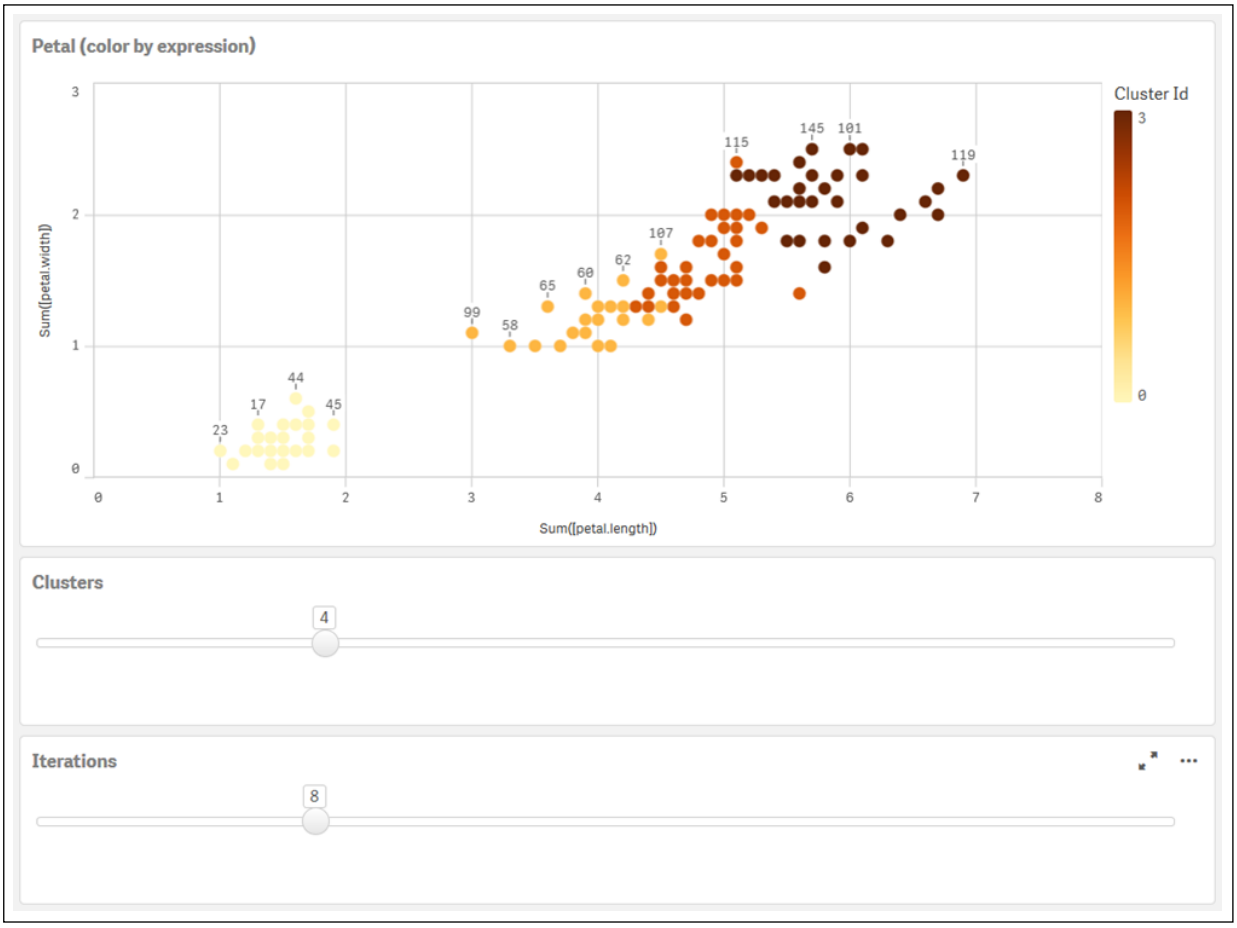
Clusters 变量输入框的外观。


General
Show titles
On
Title
Clusters
Subtitle
Footnote
Disable hover menu
Variable
Name
KmeansPetalClusters
Show as
Slider
Update on drag
Values
Min
2
Max
10
Step
1
Slider label

7. 对迭代的数目添加**变量输入**框。
    i. 将**变量输入**框拖动到工作表上。
    ii. 在**外观**下，选择**常规**。
    iii. 为**标题**输入以下内容：*迭代*
    iv. 在**外观**下，选择**变量**。
    v. 在**名称**下选择以下变量：*KmeansNumberIterations*。
    vi. 根据需要配置其他设置，

    我们现在可以使用变量输入框中的滑块更改集群和迭代的数量。

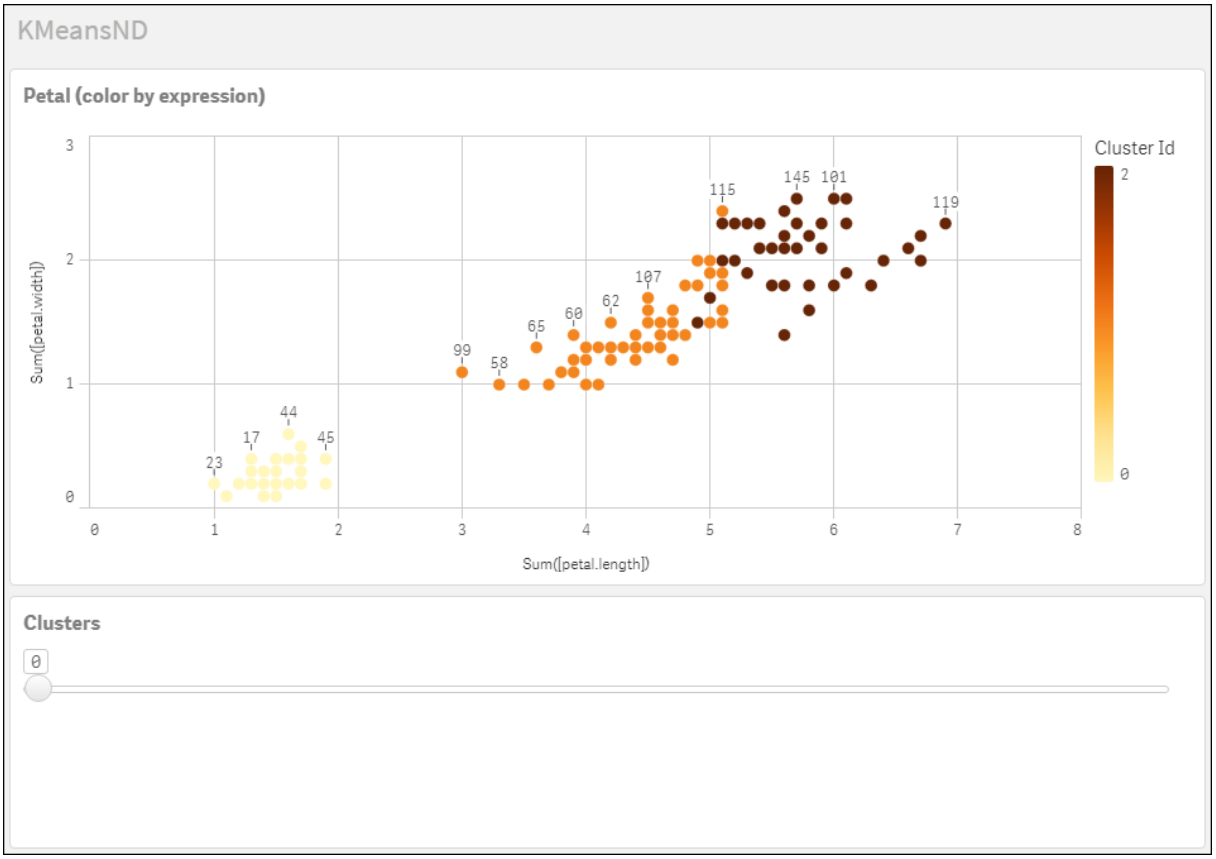
    花瓣（按表达式着色）图表中按表达式着色群集。



自动聚合

均值函数支持使用名为深度差 (DeD) 的方法支持自动聚合。如果用户为集群数设置 0，则会为该数据集确定最优集群数。注意，虽然没有显式返回集群数 (*k*) 的整数，但它是在均值算法中计算的。例如，如果在函数中为 *KmeansPetalClusters* 的值指定了 0 或通过变量输入框进行设置，则会根据最佳的集群数自动计算数据集的簇分配。给定 Iris 数据集，如果选择 0 作为集群数，则算法将确定（自动集群）此数据集的最佳集群数 (3)。

当 (k) 设置为 0 时，均值深度差分方法确定最佳集群数。



Iris 数据集：Qlik Sense 中数据加载编辑器的内联加载

```
IrisData: Load * Inline [ sepal.length, sepal.width, petal.length, petal.width, variety, id
5.1, 3.5, 1.4, 0.2, Setosa, 1 4.9, 3, 1.4, 0.2, Setosa, 2 4.7, 3.2, 1.3, 0.2, Setosa, 3 4.6,
3.1, 1.5, 0.2, Setosa, 4 5, 3.6, 1.4, 0.2, Setosa, 5 5.4, 3.9, 1.7, 0.4, Setosa, 6 4.6, 3.4,
1.4, 0.3, Setosa, 7 5, 3.4, 1.5, 0.2, Setosa, 8 4.4, 2.9, 1.4, 0.2, Setosa, 9 4.9, 3.1, 1.5,
0.1, Setosa, 10 5.4, 3.7, 1.5, 0.2, Setosa, 11 4.8, 3.4, 1.6, 0.2, Setosa, 12 4.8, 3, 1.4,
0.1, Setosa, 13 4.3, 3, 1.1, 0.1, Setosa, 14 5.8, 4, 1.2, 0.2, Setosa, 15 5.7, 4.4, 1.5, 0.4,
Setosa, 16 5.4, 3.9, 1.3, 0.4, Setosa, 17 5.1, 3.5, 1.4, 0.3, Setosa, 18 5.7, 3.8, 1.7, 0.3,
Setosa, 19 5.1, 3.8, 1.5, 0.3, Setosa, 20 5.4, 3.4, 1.7, 0.2, Setosa, 21 5.1, 3.7, 1.5, 0.4,
Setosa, 22 4.6, 3.6, 1, 0.2, Setosa, 23 5.1, 3.3, 1.7, 0.5, Setosa, 24 4.8, 3.4, 1.9, 0.2,
Setosa, 25 5, 3, 1.6, 0.2, Setosa, 26 5, 3.4, 1.6, 0.4, Setosa, 27 5.2, 3.5, 1.5, 0.2, Setosa,
28 5.2, 3.4, 1.4, 0.2, Setosa, 29 4.7, 3.2, 1.6, 0.2, Setosa, 30 4.8, 3.1, 1.6, 0.2, Setosa,
31 5.4, 3.4, 1.5, 0.4, Setosa, 32 5.2, 4.1, 1.5, 0.1, Setosa, 33 5.5, 4.2, 1.4, 0.2, Setosa,
34 4.9, 3.1, 1.5, 0.1, Setosa, 35 5, 3.2, 1.2, 0.2, Setosa, 36 5.5, 3.5, 1.3, 0.2, Setosa, 37
4.9, 3.1, 1.5, 0.1, Setosa, 38 4.4, 3, 1.3, 0.2, Setosa, 39 5.1, 3.4, 1.5, 0.2, Setosa, 40 5,
3.5, 1.3, 0.3, Setosa, 41 4.5, 2.3, 1.3, 0.3, Setosa, 42 4.4, 3.2, 1.3, 0.2, Setosa, 43 5,
3.5, 1.6, 0.6, Setosa, 44 5.1, 3.8, 1.9, 0.4, Setosa, 45 4.8, 3, 1.4, 0.3, Setosa, 46 5.1,
3.8, 1.6, 0.2, Setosa, 47 4.6, 3.2, 1.4, 0.2, Setosa, 48 5.3, 3.7, 1.5, 0.2, Setosa, 49 5,
3.3, 1.4, 0.2, Setosa, 50 7, 3.2, 4.7, 1.4, Versicolor, 51 6.4, 3.2, 4.5, 1.5, Versicolor, 52
6.9, 3.1, 4.9, 1.5, Versicolor, 53 5.5, 2.3, 4, 1.3, Versicolor, 54 6.5, 2.8, 4.6, 1.5,
Versicolor, 55 5.7, 2.8, 4.5, 1.3, Versicolor, 56 6.3, 3.3, 4.7, 1.6, Versicolor, 57 4.9, 2.4,
3.3, 1, Versicolor, 58 6.6, 2.9, 4.6, 1.3, Versicolor, 59 5.2, 2.7, 3.9, 1.4, Versicolor, 60
5, 2, 3.5, 1, Versicolor, 61 5.9, 3, 4.2, 1.5, Versicolor, 62 6, 2.2, 4, 1, Versicolor, 63
6.1, 2.9, 4.7, 1.4, Versicolor, 64 5.6, 2.9, 3.6, 1.3, Versicolor, 65 6.7, 3.1, 4.4, 1.4,
Versicolor, 66 5.6, 3, 4.5, 1.5, Versicolor, 67 5.8, 2.7, 4.1, 1, Versicolor, 68 6.2, 2.2,
```

```
4.5, 1.5, Versicolor, 69 5.6, 2.5, 3.9, 1.1, Versicolor, 70 5.9, 3.2, 4.8, 1.8, Versicolor, 71
6.1, 2.8, 4, 1.3, Versicolor, 72 6.3, 2.5, 4.9, 1.5, Versicolor, 73 6.1, 2.8, 4.7, 1.2,
Versicolor, 74 6.4, 2.9, 4.3, 1.3, Versicolor, 75 6.6, 3, 4.4, 1.4, Versicolor, 76 6.8, 2.8,
4.8, 1.4, Versicolor, 77 6.7, 3, 5, 1.7, Versicolor, 78 6, 2.9, 4.5, 1.5, Versicolor, 79 5.7,
2.6, 3.5, 1, Versicolor, 80 5.5, 2.4, 3.8, 1.1, Versicolor, 81 5.5, 2.4, 3.7, 1, Versicolor,
82 5.8, 2.7, 3.9, 1.2, Versicolor, 83 6, 2.7, 5.1, 1.6, Versicolor, 84 5.4, 3, 4.5, 1.5,
Versicolor, 85 6, 3.4, 4.5, 1.6, Versicolor, 86 6.7, 3.1, 4.7, 1.5, Versicolor, 87 6.3, 2.3,
4.4, 1.3, Versicolor, 88 5.6, 3, 4.1, 1.3, Versicolor, 89 5.5, 2.5, 4, 1.3, Versicolor, 90
5.5, 2.6, 4.4, 1.2, Versicolor, 91 6.1, 3, 4.6, 1.4, Versicolor, 92 5.8, 2.6, 4, 1.2,
Versicolor, 93 5, 2.3, 3.3, 1, Versicolor, 94 5.6, 2.7, 4.2, 1.3, Versicolor, 95 5.7, 3, 4.2,
1.2, Versicolor, 96 5.7, 2.9, 4.2, 1.3, Versicolor, 97 6.2, 2.9, 4.3, 1.3, Versicolor, 98 5.1,
2.5, 3, 1.1, Versicolor, 99 5.7, 2.8, 4.1, 1.3, Versicolor, 100 6.3, 3.3, 6, 2.5, Virginica,
101 5.8, 2.7, 5.1, 1.9, Virginica, 102 7.1, 3, 5.9, 2.1, Virginica, 103 6.3, 2.9, 5.6, 1.8,
Virginica, 104 6.5, 3, 5.8, 2.2, Virginica, 105 7.6, 3, 6.6, 2.1, Virginica, 106 4.9, 2.5,
4.5, 1.7, Virginica, 107 7.3, 2.9, 6.3, 1.8, Virginica, 108 6.7, 2.5, 5.8, 1.8, Virginica, 109
7.2, 3.6, 6.1, 2.5, Virginica, 110 6.5, 3.2, 5.1, 2, Virginica, 111 6.4, 2.7, 5.3, 1.9,
Virginica, 112 6.8, 3, 5.5, 2.1, Virginica, 113 5.7, 2.5, 5, 2, Virginica, 114 5.8, 2.8, 5.1,
2.4, Virginica, 115 6.4, 3.2, 5.3, 2.3, Virginica, 116 6.5, 3, 5.5, 1.8, Virginica, 117 7.7,
3.8, 6.7, 2.2, Virginica, 118 7.7, 2.6, 6.9, 2.3, Virginica, 119 6, 2.2, 5, 1.5, Virginica,
120 6.9, 3.2, 5.7, 2.3, Virginica, 121 5.6, 2.8, 4.9, 2, Virginica, 122 7.7, 2.8, 6.7, 2,
Virginica, 123 6.3, 2.7, 4.9, 1.8, Virginica, 124 6.7, 3.3, 5.7, 2.1, Virginica, 125 7.2, 3.2,
6, 1.8, Virginica, 126 6.2, 2.8, 4.8, 1.8, Virginica, 127 6.1, 3, 4.9, 1.8, Virginica, 128
6.4, 2.8, 5.6, 2.1, Virginica, 129 7.2, 3, 5.8, 1.6, Virginica, 130 7.4, 2.8, 6.1, 1.9,
Virginica, 131 7.9, 3.8, 6.4, 2, Virginica, 132 6.4, 2.8, 5.6, 2.2, Virginica, 133 6.3, 2.8,
5.1, 1.5, Virginica, 134 6.1, 2.6, 5.6, 1.4, Virginica, 135 7.7, 3, 6.1, 2.3, Virginica, 136
6.3, 3.4, 5.6, 2.4, Virginica, 137 6.4, 3.1, 5.5, 1.8, Virginica, 138 6, 3, 4.8, 1.8,
Virginica, 139 6.9, 3.1, 5.4, 2.1, Virginica, 140 6.7, 3.1, 5.6, 2.4, Virginica, 141 6.9, 3.1,
5.1, 2.3, Virginica, 142 5.8, 2.7, 5.1, 1.9, Virginica, 143 6.8, 3.2, 5.9, 2.3, Virginica, 144
6.7, 3.3, 5.7, 2.5, Virginica, 145 6.7, 3, 5.2, 2.3, Virginica, 146 6.3, 2.5, 5, 1.9,
Virginica, 147 6.5, 3, 5.2, 2, Virginica, 148 6.2, 3.4, 5.4, 2.3, Virginica, 149 5.9, 3, 5.1,
1.8, Virginica, 150 ];
```

KMeansCentroid2D - 图表函数

KMeansCentroid2D() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的所需坐标。集群算法使用的列分别由参数 coordinate_1 和 coordinate_2 确定。二者都是聚合型。创建的集群数由 num_clusters 参数确定。数据可以通过规范参数进行规范化。

KMeansCentroid2D 每个数据点返回一个值。返回值是一个双重值，是与数据点分配到的集群中心相对应的位置坐标之一。

语法：

```
KMeansCentroid2D(num_clusters, coordinate_no, coordinate_1, coordinate_2 [,
norm])
```

返回数据类型：双

参数：

参数

参数	说明
num_clusters	指定集群数的整数。
coordinate_no	所需的质心坐标数字(例如，对应于 x、y 或 z 轴)。
coordinate_1	计算第一个坐标的聚合，通常是散点图的 x 轴，可以从图表中生成。另外的参数 coordinate_2 计算第二个坐标。
norm	在 K-均值聚类之前应用于数据集的可选规范化方法。 可能的值： 对于规范化为 0 或 'none' 对于 z-score 规范化为 1 或 'zscore' 对于 min-max 规范化为 2 或 'minmax' 如果未提供参数或提供的参数不正确，则不应用规范化。 Z-score 基于特征均值和标准差对数据进行标准化。Z-score 并不能保证每个特征具有相同的尺度，但在处理异常值时，它是一种比 min-max 更好的方法。 Min-max 规范化通过获取每个数据点的最小值和最大值并重新计算每个数据点，确保特征具有相同的比例。

自动聚合

均值函数支持使用名为深度差 (DeD) 的方法支持自动聚合。如果用户为集群数设置 0，则会为该数据集确定最优集群数。注意，虽然没有显式返回集群数 (*k*) 的整数，但它是在均值算法中计算的。例如，如果在函数中为 *KmeansPetalClusters* 的值指定了 0 或通过变量输入框进行设置，则会根据最佳的集群数自动计算数据集的簇分配。

KMeansCentroidND- 图表函数

KMeansCentroidND() 通过应用 k 均值集群计算图表的行，并且对于每个图表行，显示此数据点已分配到的集群的所需坐标。集群算法使用的列由参数 coordinate_1 和 coordinate_2 等确定(可达 n 列)。这些都是聚合型。创建的集群数由 num_clusters 参数确定。

KMeansCentroidND 每行返回一个值。返回值是一个双重值，是与数据点分配到的集群中心相对应的位置坐标之一。

语法：

```
KMeansCentroidND((num_clusters, num_iter, coordinate_no, coordinate_1,
coordinate_2 [,coordinate_3 [, ...]])
```

返回数据类型：双

参数：

参数

参数	说明
num_clusters	指定集群数的整数。
num_iter	使用重新初始化的集群中心进行集群的迭代次数。
coordinate_no	所需的质心坐标数字(例如，对应于 x、y 或 z 轴)。
coordinate_1	计算第一个坐标的聚合，通常是散点图的 x 轴，可以从图表中生成。另外的参数计算第二、第三和第四个坐标等。

自动聚合

均值函数支持使用名为深度差 (DeD) 的方法支持自动聚合。如果用户为集群数设置 0，则会为该数据集确定最优集群数。注意，虽然没有显式返回集群数 (*k*) 的整数，但它是在均值算法中计算的。例如，如果在函数中为 *KmeansPetalClusters* 的值指定了 0 或通过变量输入框进行设置，则会根据最佳的集群数自动计算数据集的簇分配。

5.23 统计分布函数

统计分布 DIST 函数用于度量提供值给定分布点的分布函数概率。INV 函数用于在给定分布概率的情况下计算值。相反，统计聚合函数组用于计算各种统计假设检验系列统计检验值的聚合值。

以下描述的统计分布函数都是通过使用 Cephes 函数库在 Qlik Sense 中执行。有关所用算法、精确度等的参考及详情，请访问：Cephes library。Cephes 函数库经获得许可使用。

所有函数均可用于数据加载脚本和图表表达式。

统计分布函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

CHIDIST
CHIDIST() 用于返回单尾 chi^2 分布概率。chi^2 分布与 ch^{i2} 检验相关联。

```
CHIDIST (value, degrees_freedom)
```

CHIINV
CHIINV() 用于返回单尾 chi^2 分布概率的相反值。

```
CHIINV (prob, degrees_freedom)
```

NORMDIST

NORMDIST() 用于返回指定方式及标准误差的累积正态分布。如果 mean = 0 和 standard_dev = 1，则此函数返回标准正态分布。

```
NORMDIST (value, mean, standard_dev)
```

NORMINV

NORMINV() 用于返回指定方式及标准差的累积正态分布的相反值。

```
NORMINV (prob, mean, standard_dev)
```

TDIST

TDIST() 用于返回学生 t 分布的概率，其中数值是一个将要为其计算概率的 t 的计算值。

```
TDIST (value, degrees_freedom, tails)
```

TINV

TINV() 用于作为一个概率和自由度函数返回学生 t 分布的 t 值。

```
TINV (prob, degrees_freedom)
```

FDIST

FDIST() 用于返回 F 概率分布。

```
FDIST (value, degrees_freedom1, degrees_freedom2)
```

FINV

FINV() 用于返回 F 概率分布的相反值。

```
FINV (prob, degrees_freedom1, degrees_freedom2)
```

另请参见：

统计聚合函数 (page 240)

CHIDIST

CHIDIST() 用于返回单尾 chi^2 分布概率。chi^2 分布与 ch^{i2} 检验相关联。

语法：

```
CHIDIST(value, degrees_freedom)
```

返回数据类型：数字

参数：

参数

参数	说明
value	您想要用于评估分布的值。值必须不能为负数。
degrees_freedom	表示自由度数的正整数。

此函数通过以下方式与 **CHIINV** 函数关联：

```
If prob = CHIDIST(value,df), then CHIINV(prob, df) = value
```

限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
CHIDIST(8, 15)	返回 0.9238

CHIINV

CHIINV() 用于返回单尾 chi^2 分布概率的相反值。

语法：

```
CHIINV(prob, degrees_freedom)
```

返回数据类型：数字

参数：

参数

参数	说明
prob	与 chi^2 分布相关联的概率。必须为一个介于 0 和 1 之间的值。
degrees_freedom	表示自由度数的整数。

此函数通过以下方式与 **CHIDIST** 函数关联：

```
If prob = CHIDIST(value,df), then CHIINV(prob, df) = value
```

限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
CHIINV(0.9237827, 15)	返回 8.0000

FDIST

FDIST() 用于返回 F 概率分布。

语法：

```
FDIST(value, degrees_freedom1, degrees_freedom2)
```

返回数据类型： 数字

参数：

参数

参数	说明
value	您想要用于评估分布的值。**Value** 不能为负数。
degrees_freedom1	表示自由的分子度数的正整数。
degrees_freedom2	表示自由的分母度数的正整数。

此函数通过以下方式与 **FINV** 函数关联：

```
If prob = FDIST(value, df1, df2), then FINV(prob, df1, df2) = value
```

限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
FDIST(15, 8, 6)	返回 0.0019

FINV

FINV() 用于返回 F 概率分布的相反值。

语法：

```
FINV(prob, degrees_freedom1, degrees_freedom2)
```

返回数据类型：数字

参数：

参数

参数	说明
prob	与 F 概率分布相关联的概率，必须是一个介于 0 和 1 之间的数字。
degrees_freedom	表示自由度数的整数。

此函数通过以下方式与 **FDIST** 函数关联：
`If prob = FDIST(value, df1, df2), then FINV(prob, df1, df2) = value`

限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
FINV(0.0019369, 8, 6)	返回 15.0000

NORMDIST

NORMDIST() 用于返回指定方式及标准误差的累积正态分布。如果 mean = 0 和 standard_dev = 1，则此函数返回标准正态分布。

语法：

```
NORMDIST(value, [mean], [standard_dev], [cumulative])
```

返回数据类型：数字

参数：

参数

参数	说明
value	您想要用于评估分布的值。
mean	用于表示分布的算术平均值的可选值。 如果您没有声明该参数，则默认值为 0。
standard_dev	用于表示分布的标准偏差的可选正值。 如果您没有声明该参数，则默认值为 1。

参数	说明
cumulative	您可选择使用标准分布或累积分布。 0 = 标准正态分布 1 = 累积分布(默认)

此函数通过以下方式与 **NORMINV** 函数关联:
`If prob = NORMDIST(value, m, sd), then NORMINV(prob, m, sd) = value`

限制:

所有参数均必须为数字, 如不是则会返回 NULL 值。

示例和结果:

示例	结果
NORMDIST(0.5, 0, 1)	返回 0.6915

NORMINV

NORMINV() 用于返回指定方式及标准差的累积正态分布的相反值。

语法:

```
NORMINV(prob, mean, standard_dev)
```

返回数据类型: 数字

参数:

参数

参数	说明
prob	与正态分布相关联的概率。必须为一个介于 0 和 1 之间的值。
mean	用于表示分布的算术平均值的值。
standard_dev	用于表示分布的标准偏差的正值。

此函数通过以下方式与 **NORMDIST** 函数关联:
`If prob = NORMDIST(value, m, sd), then NORMINV(prob, m, sd) = value`

限制:

所有参数均必须为数字, 如不是则会返回 NULL 值。

示例和结果：

示例	结果
NORMINV(0.6914625, 0, 1)	返回 0.5000

TDIST

TDIST() 用于返回学生 t 分布的概率，其中数值是一个将要为其计算概率的 t 的计算值。

语法：

```
TDIST(value, degrees_freedom, tails)
```

返回数据类型： 数字

参数：

参数

参数	说明
value	您想要用于评估分布的值，且不能为负数。
degrees_freedom	表示自由度数的正整数。
tails	必须为 1(单尾分布) 或 2(双尾分布) 。

此函数通过以下方式与 **TINV** 函数关联：

```
If prob = TDIST(value, df ,2), then TINV(prob, df) = value
```

限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
TDIST(1, 30, 2)	返回 0.3253

TINV

TINV() 用于作为一个概率和自由度函数返回学生 t 分布的 t 值。

语法：

```
TINV(prob, degrees_freedom)
```

返回数据类型： 数字

参数：

参数

参数	说明
prob	与 T 分布相关联的双尾概率。必须为一个介于 0 和 1 之间的值。
degrees_freedom	表示自由度数的整数。

限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

此函数通过以下方式与 **TDIST** 函数关联：
`If prob = TDIST(value, df ,2), then TINV(prob, df) = value`。

示例和结果：

示例	结果
TINV(0.3253086, 30)	返回 1.0000

5.24 字符串函数

本节介绍用于处理和操作字符串的函数。

所有函数均可用于数据加载脚本和图表表达式，但 **Evaluate** 只能在数据加载脚本中使用。

字符串函数概述

每个函数都在概述后面进行了详细描述。也可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Capitalize

Capitalize() 用于返回包含首字母大写的所有单词的字符串。

```
Capitalize (text)
```

Chr

Chr() 用于返回与输入整数对应的 Unicode 字符。

```
Chr (int)
```

Evaluate

Evaluate() 用于确定是否可将输入文本字符串作为有效的 Qlik Sense 表达式来计算值，如果可以，则以字符串形式返回该表达式的值。如果输入字符串不是有效的表达式，则返回 NULL。

```
Evaluate (expression_text)
```

FindOneOf

FindOneOf() 用于搜索字符串，以便从一组提供的字符中找到任意字符出现的位置。如果不提供第三个参数(值大于 1)，则返回任意字符在搜索集合中首次出现的位置。如果未找到匹配值，则返回 **0**。

```
FindOneOf (text, char_set[, count])
```

Hash128

Hash128() 用于返回 128 位哈希的组合输入表达式值。结果为 22 个字符的字符串。

```
Hash128 (expr{, expression})
```

Hash160

Hash160() 用于返回 160 位哈希的组合输入表达式值。结果为 27 个字符的字符串。

```
Hash160 (expr{, expression})
```

Hash256

Hash256() 用于返回 256 位哈希的组合输入表达式值。结果为 43 个字符的字符串。

```
Hash256 (expr{, expression})
```

Index

Index() 用于搜索字符串，以便找到所提供子字符串第 n 次出现的开始位置。可选的第三个参数用于提供值 n，如果省略，则值为 1。如果为负值，则从字符串的结尾开始搜索。字符串中的位置从 **1** 开始编号。

```
Index  (text, substring[, count])
```

KeepChar

KeepChar() 用于返回包含第一个字符串“text”，但不包含第二个字符串“keep_chars”所包含的任何字符的字符串。

```
KeepChar (text, keep_chars)
```

Left

Left() 用于返回特定字符串，其中包含输入字符串的第一个 (leftmost) 字符，其中字符数量由第二个参数决定。

```
Left (text, count)
```

Len

Len() 用于返回输入字符串的长度。

```
Len (text)
```

LevenshteinDist

LevenshteinDist() 返回两个字符串之间的 Levenshtein 距离。它定义为将一个字符串更改为另一个字符串所需的最小单字符编辑次数(插入、删除或替换)。该函数用于模糊字符串比较。

```
LevenshteinDist (text1, text2)
```

Lower

Lower() 用于将输入字符串中的所有字符转换为小写字符。

```
Lower (text)
```

LTrim

LTrim() 用于返回由任何前导空格剪裁的输入字符串。

```
LTrim (text)
```

Mid

Mid() 返回从第二个参数“start”定义的字符位置开始的输入字符串的一部分，并返回第三个参数“count”定义的字符数量。如果省略“count”，则返回输入字符串的剩余部分。输入字符串的第一个字符的编号为 1。

```
Mid (text, start[, count])
```

Ord

Ord() 用于返回输入字符串第一个字符的 Unicode 代码点数。

```
Ord (text)
```

PurgeChar

PurgeChar() 返回包含输入字符串（“text”）中的字符，但不包括第二个参数（“remove_chars”）中的字符的字符串。

```
PurgeChar (text, remove_chars)
```

Repeat

Repeat() 用于构成特定字符串，其中包含重复的输入字符串，重复次数由第二个参数定义。

```
Repeat (text[, repeat_count])
```

Replace

Replace() 用于使用另一个子字符串替换输入字符串内出现的所有给定子字符串后，返回一个字符串。该函数为非递归函数，从左至右工作。

```
Replace (text, from_str, to_str)
```

Right

Right() 用于返回特定字符串，其中包含输入字符串末尾（最右边）的字符，其中字符数量由第二个参数决定。

```
Right (text, count)
```

RTrim

RTrim() 用于返回由任何尾部空格剪裁的输入字符串。

```
RTrim (text)
```

SubField

SubField() 用于从父字符串字段提取子字符串组成部分，其中原始记录字段由两个或更多用分隔符分隔的部分构成。

```
SubField (text, delimiter[, field_no ])
```

SubStringCount

SubStringCount() 用于返回指定子字符串在输入字符串文本中出现的次数。如果不匹配，则返回 0。

```
SubStringCount (text, substring)
```

TextBetween

TextBetween() 用于返回输入字符串中作为分隔符出现在指定字符之间的文本。

```
TextBetween (text, delimiter1, delimiter2[, n])
```

Trim

Trim() 用于返回由任何前导和尾部空格剪裁的输入字符串。

```
Trim (text)
```

Upper

Upper() 用于将输入字符串中表达式所定义的所有文本字符转换为大写。忽略数字和符号。

```
Upper (text)
```

Capitalize

Capitalize() 用于返回包含首字母大写的所有单词的字符串。

语法：

```
Capitalize(text)
```

返回数据类型： 字符串

示例：图表表达式

示例	结果
Capitalize ('star trek')	返回 'Star Trek'
Capitalize ('AA bb cC Dd')	返回 'Aa Bb Cc Dd'

示例：加载脚本

```
Load  String, Capitalize(String) Inline [String rHode iSland washingTon d.C. new york];
```

结果

字符串	Capitalize(String)
rHode iSland	Rhode Island
washingTon d.C.	Washington D.C.
new york	New York

Chr

Chr() 用于返回与输入整数对应的 Unicode 字符。

语法：

```
Chr(int)
```

返回数据类型： 字符串

示例和结果：

示例	结果
Chr(65)	返回字符串 'A'
Chr(163)	返回字符串 '£'
Chr(35)	返回字符串 '#'

Evaluate

Evaluate() 用于确定是否可将输入文本字符串作为有效的 Qlik Sense 表达式来计算值，如果可以，则以字符串形式返回该表达式的值。如果输入字符串不是有效的表达式，则返回 NULL。

语法：

```
Evaluate(expression_text)
```

返回数据类型： 双

此字符串函数不可用于图表表达式。

示例和结果：

函数示例	结果
Evaluate (5 * 8)	返回 '40'

加载脚本示例

```
Load  Evaluate(String) as Evaluated, String Inline [String 4 5+3 0123456789012345678 Today()
];
```

结果

字符串	已评估
4	4
5+3	8
0123456789012345678	0123456789012345678
Today()	2022-02-02

FindOneOf

FindOneOf() 用于搜索字符串，以便从一组提供的字符中找到任意字符出现的位置。如果不提供第三个参数(值大于 1)，则返回任意字符在搜索集合中首次出现的位置。如果未找到匹配值，则返回 **0**。

语法：

```
FindOneOf(text, char_set[, count])
```

返回数据类型： 整数

参数：

参数

参数	说明
text	原始字符串。
char_set	在 text 中搜索的字符集。
count	定义搜索哪一次出现的任何字符。例如，值为 2，则搜索第二次出现的。

示例：图表表达式

示例	结果
`FindOneOf( 'my example text string', 'et%s')`	返回“4”，因为“e”是示例字符串中的第四个字符。
`FindOneOf( 'my example text string', 'et%s', 3)`	返回 '12'，因为是搜索以下任何字符：e、t、% 或 s，"t"是第三次出现的位置，因此是在示例字符串的位置 12。
`FindOneOf( 'my example text string', '¤%&')`	返回“0”，因为示例字符串中不存在 ¤、% 或 & 中的任何字符。

示例:加载脚本

```
Load * Inline [SearchFor, Occurrence et%s,1 et%s,3 ¤%&,1]
```

结果

SearchFor	Occurrence	FindOneOf('my example text string', SearchFor, Occurrence)
et%s	1	4
et%s	3	12
¤%&	1	0

Hash128

Hash128() 用于返回 128 位哈希的组合输入表达式值。结果为 22 个字符的字符串。

语法:

```
Hash128(expr{, expression})
```

返回数据类型: 字符串

示例:图表表达式

示例	结果
Hash128 ('abc', 'xyz', '123')	返回 'MA&5]6+3=:>:>G%S<U*S2+'。
Hash128 (Region, Year, Month) Note: Region, Year, and Month are table fields.	返回 'G7*=6GKPJ(Z+)^KM?<$'A+'。

示例:加载脚本

```
Hash_128: Load *,  Hash128(Region, Year, Month) as Hash128; Load * inline [ Region, Year, Month abc, xyz, 123 EU, 2022, 01 UK, 2022, 02 US, 2022, 02 ];
```

结果

区域	年	月	Hash128
abc	xyz	123	MA&5]6+3=:>;>G%S<U*S2+
EU	2022	01	B40^K&[T@!;VB'XR]<5=/$
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!
US	2022	02	C6@#]4#_G-(]J7EQY#KRW0

Hash160

Hash160() 用于返回 160 位哈希的组合输入表达式值。结果为 27 个字符的字符串。

语法：

```
Hash160(expr{, expression})
```

返回数据类型： 字符串

示例：图表表达式

示例	结果
Hash160 ('abc', 'xyz', '123')	返回 'MA&5]6+3=:>;>G%S<U*S2I:`=X*'。
Hash160 (Region, Year, Month) Note: Region, Year, and Month are table fields.	返回 'G7*=6GKPJ (Z+)^KM?<$'AI.)?U$'。

示例：加载脚本

```
Hash_160: Load *,  Hash160(Region, Year, Month) as Hash160; Load * inline [ Region, Year, Month abc, xyz, 123 EU, 2022, 01 UK, 2022, 02 US, 2022, 02 ];
```

结果

区域	年	月	Hash160
abc	xyz	123	MA&5]6+3=:>;>G%S<U*S2I:`=X*
EU	2022	01	B40^K&[T@!;VB'XR]<5=//_F853
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!T"FWZ
US	2022	02	C6@#]4#_G-(]J7EQY#KRW`@KF+W

Hash256

Hash256() 用于返回 256 位哈希的组合输入表达式值。结果为 43 个字符的字符串。

语法：

```
Hash256(expr{, expression})
```

返回数据类型： 字符串

示例：图表表达式

示例	结果
Hash256 ('abc', 'xyz', '123')	返回 'MA&5]6+3=:>;>G%S<U*S2I:`=X*A.IO*8N\%Y7Q;YEJ'。
Hash256 (Region, Year, Month) Note: Region, Year, and Month are table fields.	返回 'G7*=6GKPJ(Z+)^KM?<$'AI.)?U$#X2RB [:0ZP=+Z`F:'。

示例:加载脚本

```
Hash_256: Load *,  Hash256(Region, Year, Month) as Hash256; Load * inline [ Region, Year, Month abc, xyz, 123 EU, 2022, 01 UK, 2022, 02 US, 2022, 02 ];
```

结果

区域	年	月	Hash256
abc	xyz	123	MA&5]6+3=:>;>G%S<U*S2I:`=X*A.IO*8N\%Y7Q;YEJ
EU	2022	01	B40^K&[T@!;VB'XR]<5=//_F853?BE6'G&,YH*T'MF)
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!T"FWZT=4\#V`M%6_\0C>4
US	2022	02	C6@#]4#_G-(]J7EQY#KRW`@KF+W-0]`[Z8R+#'")=+0

Index

Index() 用于搜索字符串,以便找到所提供子字符串第 n 次出现的开始位置。可选的第三个参数用于提供值 n,如果省略,则值为 1。如果为负值,则从字符串的结尾开始搜索。字符串中的位置从 **1** 开始编号。

语法:

```
Index(text, substring[, count])
```

返回数据类型: 整数

参数:

参数

参数	说明
text	原始字符串。
substring	在 text 中搜索的字符串。
count	定义搜索哪一次出现的 **substring**。例如,值为 2,则搜索第二次出现的。

示例和结果:

示例	结果
Index('abcdefg', 'cd')	返回 3
Index('abcdabcd', 'b', 2)	返回 6(“b”第二次出现的位置)
Index('abcdabcd', 'b',-2)	返回 2(“b”从结尾开始第二次出现的位置)
`Left( Date, Index( Date,'-') -1)` where **Date** = 1997-07-14	返回 1997

示例	结果
`Mid( Date, Index( Date, '-', 2 ) -2, 2 )` where **Date** = 1997-07-14	返回 07

示例：脚本

```
T1: Load  *,  index(String, 'cd') as Index_CD,              // returns 3  in Index_CD index
(String, 'b') as Index_B,               // returns 2 in Index_B index(String, 'b', -1) as
Index_B2;           // returns 2 or 6 in Index_B2 Load * inline [ String abcdefg abcdabcd ];
```

KeepChar

KeepChar() 用于返回包含第一个字符串“text”，但不包含第二个字符串“keep_chars”所包含的任何字符的字符串。

语法：

```
KeepChar(text, keep_chars)
```

返回数据类型：字符串

参数：

参数

参数	说明
text	原始字符串。
keep_chars	包含 text 中要保留的字符的字符串。

示例：图表表达式

示例	结果
`KeepChar ( 'a1b2c3','123' )`	返回“123”。
`KeepChar ( 'a1b2c3','1234' )`	返回“123”。
`KeepChar ( 'a1b22c3','1234' )`	返回“1223”。
`KeepChar ( 'a1b2c3','312' )`	返回“123”。

示例：加载脚本

```
T1: Load  *,  keepchar(String1, String2) as KeepChar;   Load * inline [ String1, String2
'a1b2c3', '123' ];
```

结果

Qlik Sense 表显示了在加载脚本中使用 *KeepChar* 函数的输出。

String1	String2	KeepChar
a1b2c3	123	123

另请参见:

- *PurgeChar (page 740)*

Left

Left() 用于返回特定字符串，其中包含输入字符串的第一个 (leftmost) 字符，其中字符数量由第二个参数决定。

语法:

```
Left(text, count)
```

返回数据类型: 字符串

参数:

参数	说明
text	原始字符串。
count	定义从字符串 **text** 左侧开始包含的字符数。

示例:图表表达式

示例	结果
`Left('abcdef', 3)`	返回 'abc'

示例:加载脚本

```
T1: Load  *,  left(Text,Start) as Left;        Load * inline [ Text, Start 'abcdef', 3 '2021-07-14', 4 '2021-07-14', 2 ];
```

结果

Qlik Sense 表显示了在加载脚本中使用 *Left* 函数的输出。

文本	开始	Left
abcdef	3	abc
2021-07-14	4	2021
2021-07-14	2	20

- 另请参见 *Index (page 732)*，允许分析更复杂的字符串。

Len

Len() 用于返回输入字符串的长度。

语法：

```
Len(text)
```

返回数据类型： 整数

示例：图表表达式

示例	结果
Len('Peter')	返回“5”

示例：加载脚本

```
T1: Load String, First&Second as NewString; Load *, mid(String,len(First)+1) as Second; Load *, upper(left(String,1)) as First; Load  * inline [ String this is a sample text string capitalize first letter only ];
```

结果

字符串	新字符串
this is a sample text string	This is a sample text string
capitalize first letter only	Capitalize first letter only

LevenshteinDist

LevenshteinDist() 返回两个字符串之间的 Levenshtein 距离。它定义为将一个字符串更改为另一个字符串所需的最小单字符编辑次数(插入、删除或替换)。该函数用于模糊字符串比较。

语法：

```
LevenshteinDist(text1, text2)
```

返回数据类型： 整数

示例：图表表达式

示例	结果
LevenshteinDist('Kitten','Sitting')	返回 '3'

示例:加载脚本

加载脚本

```
T1: Load *, recno() as ID; Load 'Silver' as String_1,* inline [ String_2 Sliver SSiver SSiveer
];  T1: Load *, recno()+3 as ID; Load 'Gold' as String_1,* inline [ String_2 Bold Bool Bond ];
 T1: Load *, recno()+6 as ID; Load 'Ove' as String_1,* inline [ String_2 Ove Uve Üve ];  T1:
Load *, recno()+9 as ID; Load 'ABC' as String_1,* inline [ String_2 DEFG abc ビビビ ];  set
nullinterpret = '<NULL>'; T1: Load *, recno()+12 as ID; Load 'X' as String_1,* inline [
String_2 '' <NULL> 1 ];  R1: Load  ID,  String_1,  String_2,  LevenshteinDist(String_1,
String_2) as LevenshteinDistance  resident T1;  Drop table T1;
```

结果

ID	String_1	String_2	LevenshteinDistance
1	Silver	Sliver	2
2	Silver	SSiver	2
3	Silver	SSiveer	3
4	Gold	Bold	1
5	Gold	Bool	3
6	Gold	Bond	2
7	Ove	Ove	0
8	Ove	Uve	1
9	Ove	Üve	1
10	ABC	DEFG	4
11	ABC	abc	3
12	ABC	ビビビ	3
13	X		1
14	X	-	1
15	X	1	1

Lower

Lower() 用于将输入字符串中的所有字符转换为小写字符。

语法:

```
Lower(text)
```

返回数据类型：字符串

示例：图表表达式

示例	结果
`Lower('abcD')`	返回 'abcd'

示例：加载脚本

```
Load  String, Lower(String) Inline [String rHode iSland washingTon d.C. new york];
```

结果

字符串	Lower(String)
rHode iSland	rhode island
washingTon d.C.	washington d.c.
new york	new york

LTrim

LTrim() 用于返回由任何前导空格剪裁的输入字符串。

语法：

```
LTrim(text)
```

返回数据类型：字符串

示例：图表表达式

示例	结果
`LTrim( ' abc' )`	返回 'abc'
`LTrim( 'abc ' )`	返回 'abc '

示例：加载脚本

```
Set verbatim=1;  T1:  Load *, len(LtrimString) as LtrimStringLength;       Load *, ltrim
(String) as LtrimString; Load *, len(String) as StringLength;                Load * Inline [
String '   abc    ' ' def  '];
```

示例中包含"Set verbatim=1"语句，以确保在演示 ltrim 函数之前不会自动修剪空间。有关更多信息，请参阅Verbatim (page 135)。

结果

字符串	StringLength	LtrimStringLength
def	6	5
abc	10	7

另请参见:

RTrim (page 742)

Mid

Mid() 返回从第二个参数“start”定义的字符位置开始的输入字符串的一部分，并返回第三个参数“count”定义的字符数量。如果省略“count”，则返回输入字符串的剩余部分。输入字符串的第一个字符的编号为 1。

语法:

```
Mid(text, start[, count])
```

返回数据类型: 字符串

参数:

参数

参数	说明
text	原始字符串。
start	定义 text 中要包含的第一个字符的位置的整数。
count	定义输出字符串的字符串长度。如果省略，则包含从 **start** 所定义位置开始的所有字符。

示例:图表表达式

示例	结果
`Mid('abcdef',3 )`	返回“cdef”
`Mid('abcdef',3, 2 )`	返回“cd”

示例:加载脚本

```
T1: Load *,  mid(Text,Start) as Mid1,            mid(Text,Start,Count) as Mid2;     Load *
inline [ Text, Start, Count 'abcdef', 3, 2 'abcdef', 2, 3 '210714', 3, 2 '210714', 2, 3 ];
```

结果

Qlik Sense 表显示了在加载脚本中使用 *Mid* 函数的输出。

文本	开始	Mid1	计数	Mid2
abcdef	2	bcdef	3	bcd
abcdef	3	cdef	2	cd
210714	2	10714	3	107
210714	3	0714	2	07

另请参见：

Index (page 732)

Ord

Ord() 用于返回输入字符串第一个字符的 Unicode 代码点数。

语法：

```
Ord(text)
```

返回数据类型： 整数

示例和结果：

示例：图表表达式

示例	结果
`Ord('A')`	返回整数 65。
`Ord('Ab')`	返回整数 65。

示例：加载脚本

```
//Guqin (Chinese: 古琴) – 7-stringed zithers T2: Load *,  ord(Chinese) as OrdUnicode,
        ord(Western) as OrdASCII;                    Load * inline [ Chinese, Western 古琴,
Guqin ];
```

结果：

中文	Western	OrdASCII	OrdUnicode
古琴	Guqin	71	21476

PurgeChar

PurgeChar() 返回包含输入字符串(“text”)中的字符，但不包括第二个参数(“remove_chars”)中的字符的字符串。

语法：

```
PurgeChar(text, remove_chars)
```

返回数据类型：字符串

参数：

参数

参数	说明
text	原始字符串。
remove_chars	包含 text 中要移除的字符的字符串。

返回数据类型：字符串

示例：图表表达式

示例	结果
`PurgeChar ( 'a1b2c3','123' )`	返回 'abc'。
`PurgeChar ( 'a1b2c3','312' )`	返回 'abc'。

示例：加载脚本

```
T1: Load  *,  purgechar(String1, String2) as PurgeChar;   Load * inline [ String1, String2
'a1b2c3', '123' ];
```

结果

Qlik Sense 表显示了在加载脚本中使用 *PurgeChar* 函数的输出。

String1	String2	PurgeChar
a1b2c3	123	abc

另请参见：

- *KeepChar (page 733)*

Repeat

Repeat() 用于构成特定字符串，其中包含重复的输入字符串，重复次数由第二个参数定义。

语法：

```
Repeat(text[, repeat_count])
```

返回数据类型：字符串

参数：

参数

参数	说明
text	原始字符串。
repeat_count	定义字符串 **text** 的字符在输出字符串中重复的次数。

示例：图表表达式

示例	结果
`Repeat( ' * ', rating )` when **rating** = 4	返回 '****'

示例：加载脚本

```
T1: Load *, repeat(String,2) as Repeat; Load * inline [ String hello world! hOw aRe you? ];
```

结果

字符串	重复
hello world!	hello world!hello world!
hOw aRe you?	hOw aRe you?hOw aRe you?

Replace

Replace() 用于使用另一个子字符串替换输入字符串内出现的所有给定子字符串后，返回一个字符串。该函数为非递归函数，从左至右工作。

语法：

```
Replace(text, from_str, to_str)
```

返回数据类型：字符串

参数：

参数

参数	说明
text	原始字符串。
from_str	在输入字符串 **text** 内可能出现一次或多次的字符串。
to_str	替换在字符串 **text** 内出现的所有 **from_str** 的字符串。

示例和结果:

示例	结果
`Replace('abccde','cc','xyz')`	返回 'abxyzde'

另请参见:

Right

Right() 用于返回特定字符串,其中包含输入字符串末尾(最右边)的字符,其中字符数量由第二个参数决定。

语法:

```
Right(text, count)
```

返回数据类型: 字符串

参数:

参数

参数	说明
text	原始字符串。
count	定义从字符串 **text** 最右侧开始包含的字符数。

示例:图表表达式

示例	结果
`Right('abcdef', 3)`	返回 'def'

示例:加载脚本

```
T1: Load  *,  right(Text,Start) as Right;        Load * inline [ Text, Start 'abcdef', 3
'2021-07-14', 4 '2021-07-14', 2 ];
```

结果

Qlik Sense 表显示了在加载脚本中使用 *Right* 函数的输出。

文本	开始	Right
abcdef	3	def
2021-07-14	4	7-14
2021-07-14	2	14

RTrim

RTrim() 用于返回由任何尾部空格剪裁的输入字符串。

语法：

```
RTrim(text)
```

返回数据类型：字符串

示例：图表表达式

示例	结果
RTrim(' abc')	返回 ' abc'
RTrim('abc ')	返回 'abc'

示例：加载脚本

```
Set verbatim=1;    T1:  Load *, len(RtrimString) as RtrimStringLength;         Load *, rtrim
(String) as RtrimString; Load *, len(String) as StringLength;              Load * Inline [
String '   abc    ' ' def  '];
```

示例中包含“Set verbatim=1”语句，以确保在演示 ltrim 函数之前不会自动修剪空间。有关更多信息，请参阅 Verbatim (page 135)。

结果

字符串	StringLength	RtrimStringLength
def	6	4
abc	10	6

另请参见：

- *LTrim (page 737)*

SubField

SubField() 用于从父字符串字段提取子字符串组成部分，其中原始记录字段由两个或更多用分隔符分隔的部分构成。

Subfield() 函数可用于（例如）从由全名、路径名的组成部分构成的记录的列表中提取名字和姓氏，或用于从逗号分隔的表格中提取数据。

如果在忽略可选 field_no 参数的 **LOAD** 语句中使用 **Subfield()** 函数，则会为每个子字符串生成一个完整记录。如果使用 **Subfield()** 加载多个字段，则会创建所有组合的 Cartesian 产品。

语法：

```
SubField(text, delimiter[, field_no ])
```

返回数据类型：字符串

参数：

参数

参数	说明
text	原始字符串。可以是硬编码文本、变量、货币符号扩展或其他表达式。
delimiter	输入 **text** 中将字符串分成各组成部分的字符。
field_no	可选的第三个参数是整数，用于指定返回父字符串 **text** 的哪些子字符串。使用值 1 可以返回第一个子字符串，使用值 2 可以返回第二个字符串，以此类推。 • 如果 **field_no** 为正值，子字符串是自左至右提取的。 • 如果 **field_no** 为负值，子字符串是自右至左提取的。

可以使用 SubField() 代替复杂的函数组合（例如 Len()、Right()、Left()、Mid()）和其他字符串函数。

示例：使用 SubField 的脚本和图表表达式

示例 - 脚本和图表表达式

基本示例

示例	结果
`SubField(S, ';' ,2)`	如果 **S** 为 'abc;cde;efg'，则返回 'cde'。
`SubField(S, ';' ,1)`	如果 **S** 为空字符串，则返回一个空字符串。
`SubField(S, ';' ,1)`	如果 **S** 为 ';'，则返回一个空字符串。
假设您有一个变量，其值为路径名 vMyPath， `Set vMyPath=\Users\ext_jrb\Documents\Qlik\Sense\Apps;`。	在文本和图像图表中，可添加度量，诸如：`SubField(vMyPath, '\',-3)`，结果返回 'Qlik'，因为它是从变量 vMyPath 右端开始的第三个子字符串。

脚本示例 1

加载脚本

在数据加载编辑器中加载以下脚本表达式和数据。

```
FullName:                                    LOAD * inline [ Name 'Dave Owen' 'Joe Tem' ];  SepNames:  Lo
(Name, ' ',1) as FirstName, SubField(Name, ' ',-1) as Surname Resident FullName; Drop Table
FullName;
```

创建可视化

在 Qlik Sense 工作表中创建以 **Name**、**FirstName** 和 **SurName** 为维度的表格可视化。

结果

Name	FirstName	SurName
Dave Owen	Dave	Owen
Joe Tem	Joe	Tem

解释

SubField() 函数的作用是将 **field_no** 参数设置为 1，从而提取 **Name** 的第一个子字符串。由于 **field_no** 的值为正值，因此从左到右的顺序用于提取子字符串。第二个函数调用通过将 **field _no** 参数设置为 -1 来提取第二个子字符串，该字段按照从右到左的顺序提取子字符串。

脚本示例 2

加载脚本

在数据加载编辑器中加载以下脚本表达式和数据。

```
LOAD DISTINCT Instrument, SubField(Player,',') as Player, SubField(Project,',') as Project;
Load * inline [ Instrument|Player|Project Guitar|Neil,Mike|Music,Video Guitar|Neil|Music,OST
Synth|Neil,Jen|Music,Video,OST Synth|Jo|Music Guitar|Neil,Mike|Music,OST ] (delimiter is '|');
```

创建可视化

在 Qlik Sense 工作表中创建表格可视化，将 **Instrument**、**Player** 和 **Project** 作为维度。

结果

Instrument	Player	Project
Guitar	Mike	Music
Guitar	Mike	Video
Guitar	Mike	OST
Guitar	Neil	Music
Guitar	Neil	Video
Guitar	Neil	OST
Synth	Jen	Music
Synth	Jen	Video
Synth	Jen	OST
Synth	Jo	Music

Instrument	Player	Project
Synth	Neil	Music
Synth	Neil	Video
Synth	Neil	OST

解释

此示例演示了如何使用 **Subfield()** 函数的多个实例，每个实例都不考虑 field_no 参数，其中相同的 **LOAD** 语句会创建所有组合的 Cartesian 产品。**DISTINCT** 选项用于避免创建重复记录。

SubStringCount

SubStringCount() 用于返回指定子字符串在输入字符串文本中出现的次数。如果不匹配，则返回 0。

语法：

```
SubStringCount(text, sub_string)
```

返回数据类型： 整数

参数：

参数	说明
text	原始字符串。
sub_string	在输入字符串 **text** 内可能出现一次或多次的字符串。

示例：图表表达式

示例	结果
`SubStringCount ( 'abcdefgcdxyz', 'cd' )`	返回“2”
`SubStringCount ( 'abcdefgcdxyz', 'dc' )`	返回“0”

示例：加载脚本

```
T1:
Load *,
substringcount(upper(Strings),'AB') as SubStringCount_AB;
Load * inline [
Strings
ABC:DEF:GHI:AB:CD:EF:GH
aB/cd/ef/gh/Abc/abandoned ];
```

结果

字符串	SubStringCount_AB
aB/cd/ef/gh/Abc/abandoned	3
ABC:DEF:GHI:AB:CD:EF:GH	2

TextBetween

TextBetween() 用于返回输入字符串中作为分隔符出现在指定字符之间的文本。

语法:

```
TextBetween(text, delimiter1, delimiter2[, n])
```

返回数据类型: 字符串

参数:

参数	说明
text	原始字符串。
delimiter1	指定要在 **text** 中搜索的第一个分隔符(或字符串)。
delimiter2	指定要在 **text** 中搜索的第二个分隔符(或字符串)。
n	定义搜索哪一次出现的分隔符对之间的字符。例如,值为 2,则返回第二次出现的 delimiter1 和第二次出现的 delimiter2 之间的字符。

示例:图表表达式

示例	结果
`TextBetween('<abc>', '<', '>')`	返回 'abc'
`TextBetween('<abc><de>', '<', '>',2)`	返回 'de'
`TextBetween('abc', '<', '>')` `TextBetween('<a<b', '<', '>')`	两个示例都返回 NULL。 如果在字符串中未找到任何一个分隔符,则会返回 NULL。
`TextBetween('<>', '<', '>')`	返回零长度字符串。
`TextBetween('<abc>', '<', '>', 2)`	返回 NULL,因为 n 大于分隔符的出现数。

示例:加载脚本

```
Load *,
textbetween(Text,'<','>') as TextBetween,
textbetween(Text,'<','>',2) as SecondTextBetween;
Load * inline [
```

```
Text
<abc><de>
<def><ghi><jkl> ];
```

结果

文本	TextBetween	SecondTextBetween
<abc><de>	abc	de
<def><ghi><jkl>	def	ghi

Trim

Trim() 用于返回由任何前导和尾部空格剪裁的输入字符串。

语法：

```
Trim(text)
```

返回数据类型： 字符串

示例和结果：

示例：图表表达式

示例	结果
`Trim( ' abc' )`	返回 'abc'
`Trim( 'abc ' )`	返回 'abc'
`Trim( ' abc ' )`	返回 'abc'

示例：加载脚本

```
Set verbatim=1;                                          T1: Load *, len(TrimString) as TrimStringLength;
(String) as TrimString; Load *, len(String) as StringLength;                Load * inline [
String '   abc    ' ' def  '](delimiter is '\t');
```

示例中包含"Set verbatim=1"语句，以确保在演示 ltrim 函数之前不会自动修剪空间。有关更多信息，请参阅Verbatim (page 135)。

结果：

字符串	StringLength	TrimStringLength
def	6	3
abc	10	3

Upper

Upper() 用于将输入字符串中表达式所定义的所有文本字符转换为大写。忽略数字和符号。

语法：

```
Upper(text)
```

返回数据类型： 字符串

示例：图表表达式

示例	结果
Upper(' abcD')	返回 'ABCD'

示例：加载脚本

```
Load  String,Upper(String) Inline [String rHode iSland washingTon d.C. new york];
```

结果

字符串	Upper(String)
rHode iSland	RHODE ISLAND
washingTon d.C.	WASHINGTON D.C.
new york	NEW YORK

5.25 系统函数

系统函数可提供用于访问系统、设备和 Qlik Sense 应用程序属性的函数。

系统函数概述

一部分函数在概述后面进行了详细描述。对于这些函数，可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

Author()

此函数返回一个包含当前应用程序的 author 属性的字符串。此函数均可用于数据加载脚本和图表表达式。

> *在当前版本的 Qlik Sense 中无法设置 author 属性。如果迁移 QlikView 文档，将保留 author 属性。*

ClientPlatform()

此函数返回客户端浏览器的用户代理字符串。此函数均可用于数据加载脚本和图表表达式。

示例：

```
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36
```

ComputerName

此函数返回操作系统返回的包含计算机名称的字符串。此函数均可用于数据加载脚本和图表表达式。

如果计算机的名称超过 15 个字符，字符串将仅包含前 15 个字符。

```
ComputerName( )
```

DocumentName

此函数返回一个包含当前 Qlik Sense 应用程序名称的字符串，不包括路径，但包括扩展名。此函数均可用于数据加载脚本和图表表达式。

```
DocumentName( )
```

DocumentPath

此函数用于返回一个包含至当前 Qlik Sense 应用程序完整路径的字符串。此函数均可用于数据加载脚本和图表表达式。

```
DocumentPath( )
```

在标准模式下不支持此函数。

DocumentTitle

此函数用于返回一个包含当前 Qlik Sense 应用程序标题的字符串。此函数均可用于数据加载脚本和图表表达式。

```
DocumentTitle( )
```

EngineVersion

此函数以字符串形式返回完整的 Qlik Sense 引擎版本。

```
EngineVersion ()
```

GetCollationLocale

此脚本函数返回所使用的排序规则区域设置的区域性名称。如果未设置变量 CollationLocale，则返回实际的用户计算机区域设置。

```
GetCollationLocale( )
```

GetObjectField

GetObjectField() 返回维度的名称。**Index**(索引) 是一个可选整数，表明应返回的维度。

```
GetObjectField - 图表函数([index])
```

GetRegistryString

此函数返回 Windows 注册表项的值。此函数均可用于数据加载脚本和图表表达式。

```
GetRegistryString(path, key)
```

在标准模式下不支持此函数。

IsPartialReload

此函数返回 - 如果当前部分重新加载，则返回1 (True)，否则为 0 (False)。

```
IsPartialReload ()
```

OSUser

此函数返回包含当前连接的用户的名称的字符串。此函数均可用于数据加载脚本和图表表达式。

```
OSUser( )
```

在 Qlik Sense Desktop 和 Qlik Sense Mobile Client Managed 中，此函数始终返回 “Personal\Me”。

ProductVersion

此函数用于返回完整的 Qlik Sense 版本和内部版本号作为一个字符串。

该函数已弃用并由 **EngineVersion()** 替代。

```
ProductVersion ()
```

ReloadTime

此函数返回上次完成数据加载的时间戳。此函数均可用于数据加载脚本和图表表达式。

```
ReloadTime( )
```

StateName

StateName() 会返回所使用的可视化的替代状况的名称。例如，StateName 可用于创建具有动态文本和颜色的可视化以反映可视化状态发生的更改。此函数可用于图表表达式，但不能用于确定该表达式所指的状态。

```
StateName - 图表函数()
```

EngineVersion

此函数以字符串形式返回完整的 Qlik Sense 引擎版本。

语法：

```
EngineVersion()
```

IsPartialReload

此函数返回 - 如果当前部分重新加载，则返回1 (True)，否则为 0 (False)。

语法：

```
IsPartialReload()
```

ProductVersion

此函数用于返回完整的 Qlik Sense 版本和内部版本号作为一个字符串。该函数已弃用并由 **EngineVersion()** 替代。

语法：

```
ProductVersion()
```

StateName - 图表函数

StateName() 会返回所使用的可视化的替代状况的名称。例如，StateName 可用于创建具有动态文本和颜色的可视化以反映可视化状态发生的更改。此函数可用于图表表达式，但不能用于确定该表达式所指的状态。

语法：

```
StateName ()
```

Example 1:

```
      动态文本
='Region - ' & if(StateName() = '$', 'Default', StateName())
```

Example 2:

```
      动态颜色
if(StateName() = 'Group 1', rgb(152, 171, 206),
      if(StateName() = 'Group 2', rgb(187, 200, 179),
            rgb(210, 210, 210)
      )
)
```

5.26 表格函数

表格函数会返回有关当前读取的数据表格的信息。如果未指定表格名，且该函数用于 **LOAD** 语句，则当前表格为假定表格。

所有函数均可用于数据加载脚本，而只有 **NoOfRows** 可用于图表表达式。

表格函数概述

一部分函数在概述后面进行了详细描述。对于这些函数，可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

FieldName

FieldName 脚本函数用于返回带有以前加载表格内指定数字的字段的名称。如果在 **LOAD** 语句内使用此函数，则它不必引用当前正在加载的表格。

```
FieldName (field_number ,table_name)
```

FieldNumber

FieldNumber 脚本函数用于返回以前加载表格内指定字段的数量。如果在 **LOAD** 语句内使用此函数，则它不必引用当前正在加载的表格。

```
FieldNumber (field_name ,table_name)
```

NoOfFields

NoOfFields 脚本函数用于返回以前加载表格内字段的数量。如果在 **LOAD** 语句内使用此函数，则它不必引用当前正在加载的表格。

```
NoOfFields (table_name)
```

NoOfRows

NoOfRows 函数用于返回以前加载表格内行(记录)的数量。如果在 **LOAD** 语句内使用此函数，则它不必引用当前正在加载的表格。

```
NoOfRows (table_name)
```

NoOfTables

此脚本函数返回以前加载表格的数量。

```
NoOfTables()
```

TableName

此脚本函数返回带有指定数量的表格的名称。

```
TableName(table_number)
```

TableNumber

此脚本函数返回指定表格的数量。第一个表格的编号为 0。

如果 table_name 不存在，则返回 NULL。

```
TableNumber(table_name)
```

示例：

在此例中，我们想要使用有关已经加载的表格和字段的信息创建表格。

首先，我们加载一部分样本数据。这可以创建两个用于说明此部分所介绍的表格函数的表格。

```
Characters:
Load Chr(RecNo()+Ord('A')-1) as Alpha, RecNo() as Num autogenerate 26;

ASCII:
Load
 if(RecNo()>=65 and RecNo()<=90,RecNo()-64) as Num,
 Chr(RecNo()) as AsciiAlpha,
```

```
 RecNo() as AsciiNum
autogenerate 255
 Where (RecNo()>=32 and RecNo()<=126) or RecNo()>=160 ;
```

接下来，我们使用 **NoOfTables** 函数迭代已经加载的表格，然后使用 **NoOfFields** 函数迭代每个表格的字段，并使用表格函数加载信息。

```
//Iterate through the loaded tables
For t = 0 to NoOfTables() - 1

//Iterate through the fields of table
 For f = 1 to NoOfFields(TableName($(t)))
  Tables:
  Load
   TableName($(t)) as Table,
   TableNumber(TableName($(t))) as TableNo,
   NoOfRows(TableName($(t))) as TableRows,
   FieldName($(f),TableName($(t))) as Field,
   FieldNumber(FieldName($(f),TableName($(t))),TableName($(t))) as FieldNo
   Autogenerate 1;
 Next f
Next t;
```

最终生成的表格 Tables 如下所示：

Load table

Table	TableNo	TableRows	Field	FieldNo
Characters	0	26	Alpha	1
Characters	0	26	Num	2
ASCII	1	191	Num	1
ASCII	1	191	AsciiAlpha	2
ASCII	1	191	AsciiNum	3

FieldName

FieldName 脚本函数用于返回带有以前加载表格内指定数字的字段的名称。如果在 **LOAD** 语句内使用此函数，则它不必引用当前正在加载的表格。

语法：

```
FieldName(field_number ,table_name)
```

参数：

参数

参数	说明
field_number	您想要引用的字段的字段编号。
table_name	下表包含您想要引用的字段。

示例：

```
LET a = FieldName(4,'tab1');
```

FieldNumber

FieldNumber 脚本函数用于返回以前加载表格内指定字段的数量。如果在 **LOAD** 语句内使用此函数，则它不必引用当前正在加载的表格。

语法：

```
FieldNumber(field_name ,table_name)
```

参数：

参数

参数	说明
field_name	字段名。
table_name	包含字段的表格的名称。

如果字段 field_name 不在 table_name 中，或者 table_name 不存在，则函数返回 0。

示例：

```
LET a = FieldNumber('Customer','tab1');
```

NoOfFields

NoOfFields 脚本函数用于返回以前加载表格内字段的数量。如果在 **LOAD** 语句内使用此函数，则它不必引用当前正在加载的表格。

语法：

```
NoOfFields(table_name)
```

参数：

参数

参数	说明
table_name	表格的名称。

示例：

```
LET a = NoOfFields('tab1');
```

NoOfRows

NoOfRows 函数用于返回以前加载表格内行(记录)的数量。如果在 **LOAD** 语句内使用此函数,则它不必引用当前正在加载的表格。

语法:

```
NoOfRows(table_name)
```

参数:

参数

参数	说明
table_name	表格的名称。

示例:

```
LET a = NoOfRows('tab1');
```

5.27 三角函数和双曲函数

本节介绍执行三角和双曲运算的函数。在所有函数中,参数都是用来解算以弧度测量的角度的表达式,其中 **x** 应解释为实数。

所有角度都以弧度为单位。

所有函数均可用于数据加载脚本和图表表达式。

cos

x 的余弦。结果是介于 -1 与 1 之间的数字。

```
cos( x )
```

acos

x 的反余弦。仅在 -1≤**x**≤1 时才可定义此函数。结果是介于 0 和 π 之间的数字。

```
acos( x )
```

sin

x 的正弦。结果是介于 -1 与 1 之间的数字。

```
sin( x )
```

asin

x 的反正弦。仅在 -1≤**x**≤1 时才可定义此函数。结果是介于 -π/2 和 π/2 之间的数字。

```
asin( x )
```

tan

x 的正切。结果为实数。

```
tan( x )
```

atan

x 的反正切。结果是介于 -π/2 和 π/2 之间的数字。

```
atan( x )
```

atan2

反正切函数的二维广义形式。返回原点和 **x**, **y** 坐标所决定点之间的角度。结果是介于 -π 和 +π 之间的数字。

```
atan2( y,x )
```

cosh

x 的双曲余弦。结果为正实数。

```
cosh( x )
```

sinh

x 的双曲正弦。结果为实数。

```
sinh( x )
```

tanh

x 的双曲正切。结果为实数。

```
tanh( x )
```

acosh

x 的反双曲余弦。结果为正实数。

```
acosh( x )
```

asinh

x 的反双曲正弦。结果为实数。

```
asinh( x )
```

atanh

x 的反双曲正切。结果为实数。

```
atanh( x )
```

示例：

以下脚本代码用于加载示例表格，然后加载包含值计算的三角函数和双曲操作的表格。

```
SampleData:
LOAD * Inline
[Value
-1
```

```
0
1];

Results:
Load *,
cos(Value),
acos(Value),
sin(Value),
asin(Value),
tan(Value),
atan(Value),
atan2(Value, Value),
cosh(Value),
sinh(Value),
tanh(Value)
RESIDENT SampleData;

Drop Table SampleData;
```

6 文件系统访问限制

出于安全原因，标准模式下的 Qlik Sense 不支持数据加载脚本中的路径，或者展示文件系统的函数和变量。

但是，由于 QlikView 支持文件系统路径，因此可以禁用标准模式，使用旧模式，以便重复使用 QlikView 加载脚本。

禁用标准模式会展示文件系统，从而带来安全风险。

禁用标准模式 (page 764)

6.1 当连接到基于文件的 ODBC 和 OLE DB 数据连接时的安全性

使用基于文件的驱动程序的 ODBC 和 OLE DB 数据连接会在连接字符串中暴露指向已连接数据文件的路径。当在数据选择对话框或某些 SQL 查询中编辑连接时，可能会暴露路径。在标准模式和旧模式中都可能会发生这种情况。

如果暴露指向数据文件的路径已成为一个问题，则我们建议使用文件夹数据连接来连接到数据文件(如果可能)。

6.2 标准模式中的限制

在标准模式下，不能使用几种语句、变量和函数，或者有限制。如果在数据加载脚本中使用不支持的语句，则会在加载脚本运行时产生错误。错误信息可在脚本日志文件中找到。如果使用不支持的变量和函数，不会生成错误信息或日志文件条目。相反，函数会返回 NULL 值。

在编辑数据加载脚本时，没有任何指示表明不支持变量、语句或函数。

系统变量

系统变量

变量	标准模式	旧模式	定义
Floppy	不支持	支持	用于返回找到的第一个软盘驱动器的驱动器号，通常是 *a:*。

变量	标准模式	旧模式	定义
CD	不支持	支持	用于返回找到的第一个 CD-ROM 驱动器的驱动器号。如果未找到任何 CD-ROM，随后会返回 *c:*。
QvPath	不支持	支持	用于返回浏览字符串到可执行的 Qlik Sense 文件。
QvRoot	不支持	支持	用于返回可执行的 Qlik Sense 的根目录。
QvWorkPath	不支持	支持	用于返回浏览字符串到当前 Qlik Sense 应用程序。
QvWorkRoot	不支持	支持	用于返回当前 Qlik Sense 应用程序的根目录。
WinPath	不支持	支持	用于返回浏览字符串到 Windows。
WinRoot	不支持	支持	返回 Windows 的根目录。
$(include=...)	支持的输入：使用库连接的路径	支持的输入：使用库连接或文件系统的路径	**Include/Must_Include** 变量用于指定包含应包括在脚本中并作为脚本代码计算值的文本的文件。它不用于添加数据。您可以将部分脚本代码存储在单独的文本文件中，并可以在多个应用程序中重复使用它。这是用户定义的变量。

常规脚本语句

常规脚本语句

语句	标准模式	旧模式	定义
Binary	支持的输入：使用库连接的路径	支持的输入：使用库连接或文件系统的路径	**binary** 语句用于加载另一个应用程序中的数据。

语句	标准模式	旧模式	定义
Connect	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	**CONNECT** 语句用于定义 Qlik Sense 通过 OLE DB/ODBC 接口访问通用数据库。对于 ODBC, 首先需要用 ODBC 管理员指定数据源。
Directory	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	**Directory** 语句用于定义在后续 **LOAD** 语句中查找数据文件的目录, 直到出现新的 **Directory** 语句。
Execute	不支持	支持的输入:使用库连接或文件系统的路径	**Execute** 语句用于在 Qlik Sense 加载数据的同时运行其他程序。例如, 需要执行转换。
LOAD from ...	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	**LOAD** 语句可以加载以下来源的字段:文件、脚本中定义的数据、预先载入的输入表格、网页、后续 **SELECT** 语句产生的结果或自动生成的数据。
Store into ...	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	**Store** 语句创建 QVD、CSV 或 text 文件。

脚本控制语句

脚本控制语句

语句	标准模式	旧模式	定义
For each... filelist mask/dirlist mask	支持的输入:使用库连接的路径 返回的输出:库连接	支持的输入:使用库连接或文件系统的路径 返回的输出:库连接或文件系统路径, 具体取决于输入	filelist mask 语法会在匹配 **filelist mask** 的当前目录中生成逗号分隔的全部文件列表。**dirlist mask** 语法会在匹配目录名称掩码的当前目录中生成逗号分隔的全部目录列表。

文件函数

文件函数

函数	标准模式	旧模式	定义
Attribute()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	以文本形式返回不同媒体文件的元标签的值。
ConnectString()	返回的输出:库连接名称	库连接名称或实际连接,具体取决于输入	为 ODBC 或 OLE DB 连接返回激活连接字符串。
FileDir()	返回的输出:库连接	返回的输出:库连接或文件系统路径,具体取决于输入	**FileDir** 函数用于返回一个包含至当前阅读表格文件目录的路径。
FilePath()	返回的输出:库连接	返回的输出:库连接或文件系统路径,具体取决于输入	**FilePath** 函数用于返回一个包含至当前阅读表格文件的完整路径的字符串。
FileSize()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	**FileSize** 函数用于返回一个包含文件 filename 字节大小的整数,或如果未指定 filename,则返回一个包含当前阅读的表格文件字节大小的整数。
FileTime()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	**FileTime** 函数用于返回文件 filename 的上一次修改日期和时间的时间戳。如果未指定 filename,则此函数将参考当前阅读的表格文件。

函数	标准模式	旧模式	定义
GetFolderPath()	不支持	返回的输出:绝对路径	**GetFolderPath** 函数用于返回 Microsoft Windows *SHGetFolderPath* 函数的值。此函数用于输入 Microsoft Windows 文件夹的名称,并返回该文件夹的完整路径。
QvdCreateTime()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	此脚本函数用于返回 QVD 文件的 XML-标题时间戳(如果有),否则返回 NULL 值。
QvdFieldName()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	此脚本函数返回 QVD 文件中的字段编号 **fieldno**。如果字段不存在,则返回 NULL。
QvdNoOfFields()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	此脚本函数用于返回 QVD 文件中的字段数。
QvdNoOfRecords()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	此脚本函数用于返回 QVD 文件中的当前记录数。
QvdTableName()	支持的输入:使用库连接的路径	支持的输入:使用库连接或文件系统的路径	此脚本函数用于返回存储在 QVD 文件中的表格名称。

系统函数

系统函数

函数	标准模式	旧模式	定义
DocumentPath()	不支持	返回的输出:绝对路径	此函数用于返回一个包含至当前 Qlik Sense 应用程序完整路径的字符串。
GetRegistryString()	不支持	支持	返回一个称为注册关键字的值及一个给定的注册路径。此函数可用于图表及脚本等类似程序中。

6.3 禁用标准模式

您可以禁用标准模式，或换言之，设置旧模式，以便重复使用 QlikView 加载脚本，查阅绝对或相对文件路径以及库连接。

禁用标准模式会展示文件系统，从而带来安全风险。

Qlik Sense

对于 Qlik Sense，可以使用**标准模式**属性在 QMC 中禁用标准模式。

Qlik Sense Desktop

在 Qlik Sense Desktop 中，可以使用 *Settings.ini* 设置标准/旧模式。

如果您使用默认安装位置安装了 Qlik Sense Desktop，则 *Settings.ini* 位于 *C:\Users\{user}\Documents\Qlik\Sense\Settings.ini*。如果将 Qlik Sense Desktop 安装至您选择的文件夹，则 *Settings.ini* 位于安装路径的 *Engine* 文件夹。

执行以下操作：

1. 在文本编辑器中打开 *Settings.ini*。
2. 将 *StandardReload=1* 更改为 *StandardReload=0*。
3. 保存文件并启动 Qlik Sense Desktop。

Qlik Sense Desktop 现在以旧模式运行。

设置

StandardReload 的可用设置是：

- 1(标准模式)
- 0(旧模式)

7 Qlik Sense 不支持的 QlikView 函数和语句

在 Qlik Sense 中也会支持可用于 QlikView 加载脚本和图表表达式的大部分函数和语句，但有一些例外，如下所述。

7.1 Qlik Sense 不支持的脚本语句

Qlik Sense 中不支持的 QlikView 脚本语句

语句	注释
Command	使用 SQL。
InputField	

7.2 Qlik Sense 不支持的函数

下表介绍了 Qlik Sense 不支持的 QlikView 脚本和图表函数。

- GetCurrentField
- GetExtendedProperty
- Input
- InputAvg
- InputSum
- MsgBox
- NoOfReports
- ReportComment
- ReportId
- ReportName
- ReportNumber

7.3 Qlik Sense 不支持的前缀

下表介绍了 Qlik Sense 不支持的 QlikView 前缀。

- Bundle
- Image_Size
- Info

8 不推荐的函数和语句Qlik Sense

可在 QlikView 加载脚本和图表表达式中使用的大部分函数和语句在 Qlik Sense 中也受支持，但不建议将某些函数和语句用于 Qlik Sense。还存在已经弃用的在之前版本的 Qlik Sense 中可用的函数和语句。

由于兼容性原因，它们仍可按照预期方式起作用，但最好是根据本部分中的建议更新代码，因为可能会在未来的版本中将它们删除。

8.1 Qlik Sense 不推荐的脚本语句

该表介绍了不建议用于 Qlik Sense 的脚本语句。

不推荐的脚本语句

语句	推荐
Command	使用 **SQL**。
CustomConnect	使用 **Custom Connect**。

8.2 Qlik Sense 不推荐的脚本语句参数

该列表介绍了不建议用于 Qlik Sense 的脚本语句参数。

不推荐的脚本语句参数

语句	参数
Buffer	使用 **Incremental**： • **Inc**(不推荐) • **Incr**(不推荐)

语句	参数
LOAD	以下参数关键字由 QlikView 文件转换向导生成。在重新加载数据时保留功能，但 Qlik Sense 不会提供使用以下参数生成语句的指导支持/向导： • Bottom • Cellvalue • Col • Colmatch • Colsplit • Colxtr • Compound • Contain • Equal • Every • Expand • Filters • Intarray • Interpret • Length • Longer • Numerical • Pos • Remove • Rotate • Row • Rowcnd • Shorter • Start • Strcnd • Top • Transpose • Unwrap • XML: XMLSAX and Pattern is Path

8.3 Qlik Sense 不推荐的函数

该列表介绍了不建议用于 Qlik Sense 的脚本和图表函数。

不推荐的函数

函数	推荐
NumAvg **NumCount** **NumMax** **NumMin** **NumSum**	使用范围函数。 *范围函数 (page 643)*
Color() **QliktechBlue** **QliktechGray**	使用其他颜色函数。**QliktechBlue()** 由 **RGB(8, 18, 90)** 替代且 **QliktechGray** 由 **RGB(158, 148, 137)** 替代可以获得相同的颜色。 *颜色函数 (page 377)*
QlikViewVersion	使用 **EngineVersion**。 *EngineVersion (page 751)*
ProductVersion	使用 **EngineVersion**。 *EngineVersion (page 751)*
QVUser	
Year2Date	使用 **YearToDate**。
Vrank	使用 **Rank**。
WildMatch5	使用 **WildMatch**。

ALL 限定符

在 QlikView 中，**ALL** 限定符可能会先于表达式出现。这等同于使用 **{1} TOTAL**。计算会扩展到文档内字段的全部值，但忽略图表维度和当前选择。始终返回相同的值，不论文档逻辑状态为何。如果使用 **ALL** 限定符，则不可使用集合表达式，因为 **ALL** 限定符可自行定义集合。出于遗留原因，**ALL** 限定符在 Qlik Sense 版本内仍有效，但可能会在未来版本中删除。