脚本语法和图表函数

Qlik Sense®
November 2024

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 };
```

3 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。

3.1 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 的数据连接。

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。*
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。

3.2 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 加载前缀用于转置“交叉表”或“透视表”结构化数据。使用电子表格源时，通常会遇到以这种方式构造的数据。**crosstable** 加载前缀的输出和目的是将此类结构转换为规则的面向列的表等价物，因为这种结构通常更适合 Qlik Sense 中的分析。

```
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

Generic 加载前缀允许将实体-属性-值建模数据 (EAV) 转换为传统的规范化关系表结构。EAV 建模也称为“通用数据建模”或“开放模式”。

```
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 96) 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	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，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

Concatenate 是一个脚本加载前缀，使数据集能够附加到内存中已存在的表中。它通常用于将不同的事务数据集附加到单个中心事实表，或构建源自多个源的特定类型的公共引用数据集。它的功能类似于 SQL UNION 运算符。

concatenate 操作的结果表将包含原始数据集，新的数据行附加到该表的底部。源表和目标表可能存在不同的字段。如果字段不同，则结果表将加宽，以表示源表和目标表中所有字段的组合结果。

语法：

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

参数

参数	说明
tablename	现有表格的名称。命名表将是 Concatenate操作的目标，加载的任何数据记录都将附加到该表中。如果未使用 tablename 参数，则目标表将是此语句之前最后加载的表。
loadstatement/selectstatement	tablename 参数后面的 loadstatement/selectstatement 参数将连接到指定的表。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数

示例	结果
Concatenate (Transactions) Load …. ;	在 Concatenate 前缀下面的 LOAD 语句中加载的数据将附加到名为 Transactions 的现有内存表中(假定在加载脚本中的这一点之前已加载了名为 Transactions 的表)。

示例 1 – 将多组数据附加到具有级联加载前缀的目标表

加载脚本和结果

概述

在本例中，您将按顺序加载两个脚本。

- 第一个加载脚本包含一个初始数据集，其中包含发送到名为 Transactions 的表的日期和金额。
- 第二个加载脚本包含：
    - 通过使用 Concatenate 前缀附加到初始数据集的第二个数据集。此数据集有一个附加

字段 type，该字段不在初始数据集中。

- Concatenate 前缀。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

第一个加载脚本

```
Transactions:
Load * Inline [

id, date, amount
3750, 08/30/2018, 23.56
3751, 09/07/2018, 556.31
3752, 09/16/2018, 5.75
3753, 09/22/2018, 125.00
3754, 09/22/2018, 484.21
3756, 09/22/2018, 59.18
3757, 09/23/2018, 177.42
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount

第一个加载脚本结果表

id	日期	金额
3750	08/30/2018	23.56
3751	09/07/2018	556.31
3752	09/16/2018	5.75
3753	09/22/2018	125.00
3754	09/22/2018	484.21
3756	09/22/2018	59.18
3757	09/23/2018	177.42

该表显示了初始数据集。

第二个加载脚本

打开数据加载编辑器，并添加下面的加载脚本。

```
Concatenate(Transactions)
Load * Inline [
id, date, amount, type
```

```
3758, 10/01/2018, 164.27, Internal
3759, 10/03/2018, 384.00, External
3760, 10/06/2018, 25.82, Internal
3761, 10/09/2018, 312.00, Internal
3762, 10/15/2018, 4.56, Internal
3763, 10/16/2018, 90.24, Internal
3764, 10/18/2018, 19.32, External
];
```

结果

加载数据并转到工作表。将此字段创建为维度：

- type

第二个加载脚本结果表

id	日期	金额	类型
3750	08/30/2018	23.56	-
3751	09/07/2018	556.31	-
3752	09/16/2018	5.75	-
3753	09/22/2018	125.00	-
3754	09/22/2018	484.21	-
3756	09/22/2018	59.18	-
3757	09/23/2018	177.42	-
3758	10/01/2018	164.27	内部
3759	10/03/2018	384.00	外部
3760	10/06/2018	25.82	内部
3761	10/09/2018	312.00	内部
3762	10/15/2018	4.56	内部
3763	10/16/2018	90.24	内部
3764	10/18/2018	19.32	外部

请注意 type 字段中未定义 type 的加载的前七条记录的空值。

示例 2 – 使用隐式串联将多组数据附加到目标表

加载脚本和结果

概述

隐式附加数据的一个典型用例是，当您加载几个结构相同的数据文件并希望将它们全部附加到目标表中时可使用它。

例如，通过在文件名中使用 wildcards，语法为：

```
myTable:
Load * from [myFile_*.qvd] (qvd);
```

或使用结构如下的循环：

```
for each file in filelist('myFile_*.qvd')

myTable:
Load * from [$(file)] (qvd);

next file
```

隐式串联将发生在加载了同名字段的任何两个表之间，即使它们没有在脚本中逐个定义。这可能导致数据被无意地附加到表中。如果不希望以这种方式附加具有相同字段的辅助表，请使用 NoConcatenate 加载前缀。使用备用表名标记重命名表不足以防止发生隐式串联。有关更多信息，请参阅NoConcatenate (page 86)。

在本例中，您将按顺序加载两个脚本。

- 第一个加载脚本包含一个带四个字段的初始数据集，该数据集被发送到名为 Transactions 的表。
- 第二个加载脚本包含与第一个数据集具有相同字段的数据集。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

第一个加载脚本

```
Transactions:
Load * Inline [
id, date, amount, type
3758, 10/01/2018, 164.27, Internal
3759, 10/03/2018, 384.00, External
3760, 10/06/2018, 25.82, Internal
3761, 10/09/2018, 312.00, Internal
3762, 10/15/2018, 4.56, Internal
3763, 10/16/2018, 90.24, Internal
3764, 10/18/2018, 19.32, External
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount
- type

第一个加载脚本结果表

id	日期	类型	金额
3758	10/01/2018	内部	164.27
3759	10/03/2018	外部	384.00
3760	10/06/2018	内部	25.82
3761	10/09/2018	内部	312.00
3762	10/15/2018	内部	4.56
3763	10/16/2018	内部	90.24
3764	10/18/2018	外部	19.32

该表显示了初始数据集。

第二个加载脚本

打开数据加载编辑器，并添加下面的加载脚本。

```
Load * Inline [
id, date, amount, type
3765, 11/03/2018, 129.40, Internal
3766, 11/05/2018, 638.50, External
];
```

结果

加载数据并转到工作表。

第二个加载脚本结果表

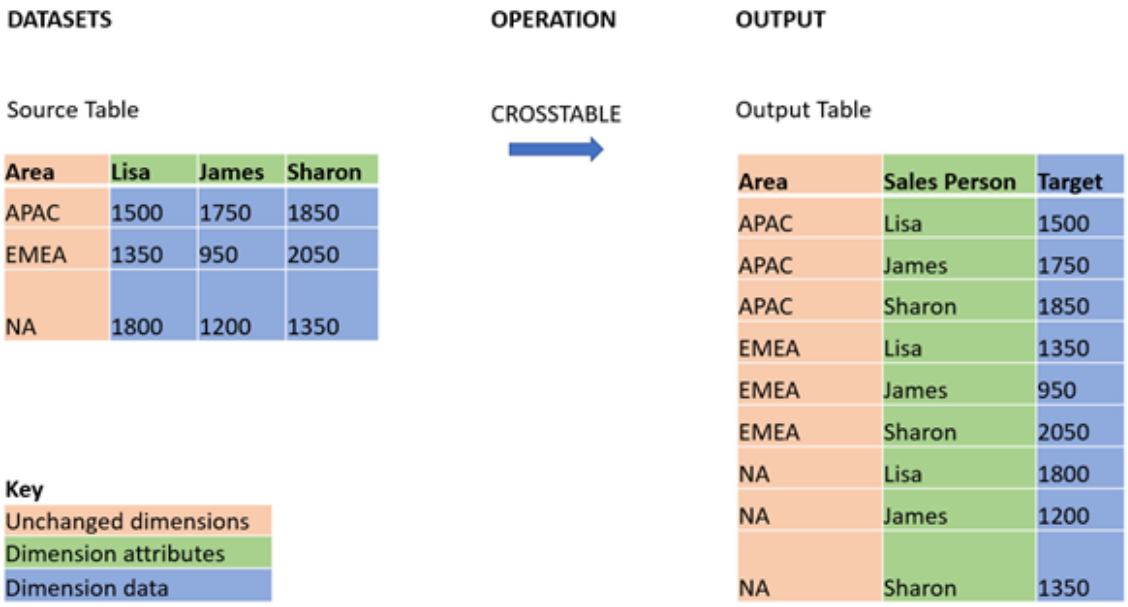
id	日期	类型	金额
3758	10/01/2018	内部	164.27
3759	10/03/2018	外部	384.00
3760	10/06/2018	内部	25.82
3761	10/09/2018	内部	312.00
3762	10/15/2018	内部	4.56
3763	10/16/2018	内部	90.24
3764	10/18/2018	外部	19.32
3765	11/03/2018	内部	129.40
3766	11/05/2018	外部	638.50

第二个数据集隐式串联到初始数据集，因为它们具有相同的字段。

Crosstable

crosstable 加载前缀用于转置“交叉表”或“透视表”结构化数据。使用电子表格源时，通常会遇到以这种方式构造的数据。**crosstable** 加载前缀的输出和目的是将此类结构转换为规则的面向列的表等价物，因为这种结构通常更适合 Qlik Sense 中的分析。

交叉表转换后数据结构为交叉表及其等效结构的示例



语法：

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

参数

参数	说明
attribute field name	描述要转置的水平方向维度的所需输出字段名称(标题行)。
data field name	所需的输出字段名称，用于描述要转置的维度的水平方向数据(标题行下方的数据值矩阵)。
n	要被转换成常规形式的表格前面的限定符字段数量或未更改的维度。默认值为 1。

此脚本函数与以下函数相关：

相关函数

函数	交互
Generic (page 55)	一种转换加载前缀，用于获取实体属性值结构化数据集，并将其转换为常规关系表结构，将遇到的每个属性分离为新的字段或数据列。

示例 1–转换数据透视销售数据(简单)

加载脚本和结果

概述

打开数据加载编辑器,并将下面的第一加载脚本添加到新选项卡。

第一个加载脚本包含一个数据集,crosstable 脚本前缀稍后将应用于该数据集,其中应用 crosstable 的部分已注释掉。这意味着注释语法用于在加载脚本中禁用此部分。

第二个加载脚本与第一个加载脚本相同,但应用了未注释 crosstable(通过删除注释语法启用)。脚本以这种方式显示,以突出显示该脚本函数在转换数据中的价值。

第二加载脚本(应用的函数)

```
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

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- Product
- Jan 2021
- Feb 2021
- Mar 2021
- Apr 2021
- May 2021
- Jun 2021

结果表

产品	2021 年 1 月	2021 年 2 月	2021 年 3 月	2021 年 4 月	2021 年 5 月	2021 年 6 月
A	100	98	103	63	108	82
B	284	279	297	305	294	292
C	50	53	50	54	49	51

该脚本允许创建一个交叉表，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，放在一个三列表格中。下一节将解释如何对交叉表进行这种转换。

第二加载脚本(应用的函数)

通过删除 // 来取消对脚本的注释。加载的脚本应如下所示：

```
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

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- Product
- Month
- Sales

结果表

产品	月	销售额值
A	Jan 2021	100
A	Feb 2021	98
A	Mar 2021	103

产品	月	销售额值
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

应用脚本前缀后，交叉表将转换为一个垂直表，其中一列用于 Month，另一列用于 Sales。这提高了数据的可读性。

示例 2 – 将数据透视销售目标数据转换为垂直表结构(中等)

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为目标的表中的数据集。
- crosstable 加载前缀，将数据透视的销售人员姓名转换为其自己的字段，标记为 Sales Person。
- 关联的销售目标数据，该数据被结构化为一个名为 Target 的字段。

加载脚本

```
SalesTargets:
CROSSTABLE([Sales Person],Target,1)
LOAD
*
```

```
INLINE [
Area, Lisa, James, Sharon
APAC, 1500, 1750, 1850
EMEA, 1350, 950, 2050
NA, 1800, 1200, 1350
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- Area
- Sales Person

添加该度量：

```
=Sum(Target)
```

结果表

面积图	销售员	=Sum(Target)
APAC	James	1750
APAC	Lisa	1500
APAC	Sharon	1850
EMEA	James	950
EMEA	Lisa	1350
EMEA	Sharon	2050
NA	James	1200
NA	Lisa	1800
NA	Sharon	1350

如果要将数据的显示复制为数据透视输入表，可以在工作表中创建等效的数据透视表。

执行以下操作：

1. 将刚刚创建的表复制并粘贴到工作表中。
2. 将**透视表**图表对象拖动到新创建的表副本的顶部。选择**转换**。
3. 单击 ✔ **完成编辑**。
4. 将 Sales Person 字段从垂直列架拖动到水平列架。

下表显示了初始表格形式的数据，如 Qlik Sense 中所示：

原始结果表，如 Qlik Sense 中所示

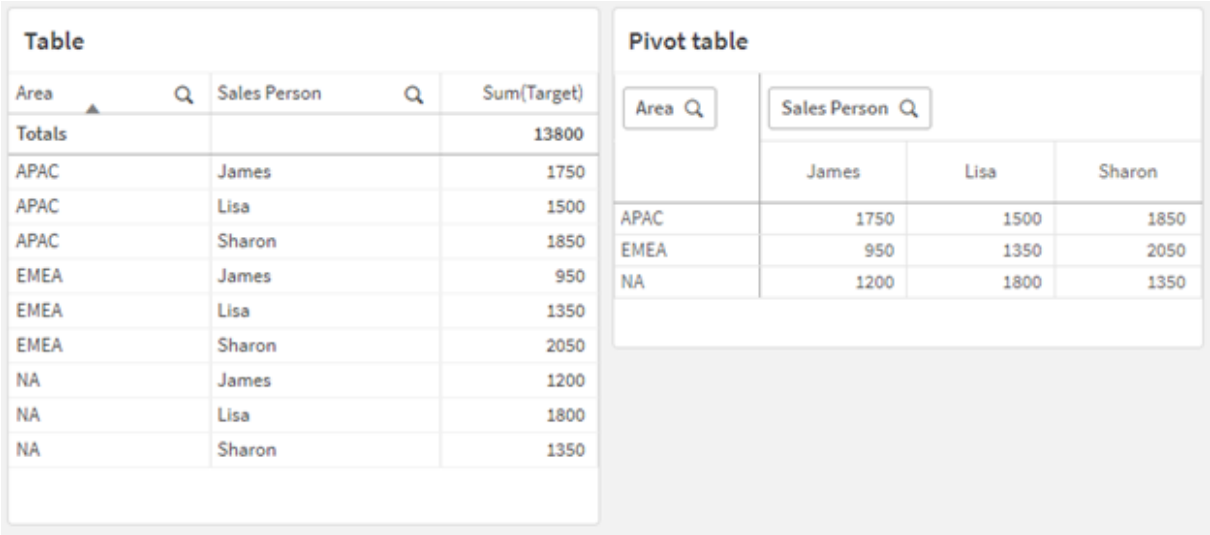
面积图	销售员	=Sum(Target)
总计	-	13800
APAC	James	1750
APAC	Lisa	1500
APAC	Sharon	1850
EMEA	James	950
EMEA	Lisa	1350
EMEA	Sharon	2050
NA	James	1200
NA	Lisa	1800
NA	Sharon	1350

等效数据透视表类似如下，每个销售人员姓名的列包含在较大的 `Sales Person` 行中：

带水平透视 `Sales Person` 字段的等效透视表

面积图	James	Lisa	Sharon
APAC	1750	1500	1850
EMEA	950	1350	2050
NA	1350	1350	1350

显示为表和等效数据透视表的数据示例，该透视表具有水平透视的 `Sales Person`



示例 3–将数据透视销售和目标数据转换为垂直表结构(高级)

加载脚本和图表表达式

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 表示销售和目标数据的数据集,按地区和月份组织。这将加载到名为 SalesAndTargets 的表中。
- crosstable 加载前缀。这用于将 Month Year 维度取消透视到专用字段中,以及将销售和目标金额矩阵转换到名为 Amount 的专用字段中。
- 使用文本到日期转换特性 date# 将 Month Year 字段从文本转换为适当的日期。此日期转换 Month Year 字段通过 Join 加载前缀连接回 SalesAndTarget 表中。

加载脚本

```
SalesAndTargets:
CROSSTABLE(MonthYearAsText,Amount,2)
LOAD
       *
INLINE [
Area    Type    Jan-22  Feb-22  Mar-22  Apr-22  May-22  Jun-22  Jul-22  Aug-22  Sep-22  Oct-22  Nov-22  Dec-22
APAC    Target  425     425     425     425     425     425     425     425     425     425     425     425
APAC    Actual  435     434     397     404     458     447     413     458     385     421     448     397
EMEA    Target  362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5
EMEA    Actual  363.5   359.5   337.5   361.5   341.5   337.5   379.5   352.5   327.5   337.5   360.5   334.5
NA      Target  375     375     375     375     375     375     375     375     375     375     375     375
NA      Actual  378     415     363     356     403     343     401     365     393     340     360     405
] (delimiter is '\t');

tmp:
LOAD DISTINCT MonthYearAsText,date#(MonthYearAsText,'MMM-YY') AS [Month Year]
RESIDENT SalesAndTargets;

JOIN (SalesAndTargets)
LOAD * RESIDENT tmp;

DROP TABLE tmp;
DROP FIELD MonthYearAsText;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- Area
- Month Year

使用标签 Actual 创建以下度量值:

```
=Sum({<Type={'Actual'}>} Amount)
```

同时创建此度量，具有标签 `Target`：

```
=Sum({<Type={'Target'}>} Amount)
```

结果表(裁剪)

面积图	年度月份	实际	目标
APAC	Jan-22	435	425
APAC	Feb-22	434	425
APAC	Mar-22	397	425
APAC	Apr-22	404	425
APAC	May-22	458	425
APAC	Jun-22	447	425
APAC	Jul-22	413	425
APAC	Aug-22	458	425
APAC	Sep-22	385	425
APAC	Oct-22	421	425
APAC	Nov-22	448	425
APAC	Dec-22	397	425
EMEA	Jan-22	363.5	362.5
EMEA	Feb-22	359.5	362.5

如果要将数据的显示复制为数据透视输入表，可以在工作表中创建等效的数据透视表。

执行以下操作：

1. 将刚刚创建的表复制并粘贴到工作表中。
2. 将**透视表**图表对象拖动到新创建的表副本的顶部。选择**转换**。
3. 单击 ✔ **完成编辑**。
4. 将 `Month Year` 字段从垂直列架拖动到水平列架。
5. 将 `Values` 项目从水平列架拖动到垂直列架。

下表显示了初始表格形式的数据，如 Qlik Sense 中所示：

原始结果表(裁剪)，如 Qlik Sense 中所示

面积图	年度月份	实际	目标
总计	-	13812	13950

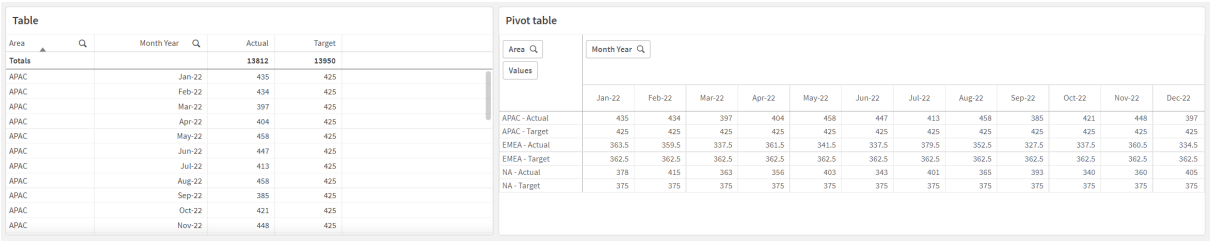
面积图	年度月份	实际	目标
APAC	Jan-22	435	425
APAC	Feb-22	434	425
APAC	Mar-22	397	425
APAC	Apr-22	404	425
APAC	May-22	458	425
APAC	Jun-22	447	425
APAC	Jul-22	413	425
APAC	Aug-22	458	425
APAC	Sep-22	385	425
APAC	Oct-22	421	425
APAC	Nov-22	448	425
APAC	Dec-22	397	425
EMEA	Jan-22	363.5	362.5
EMEA	Feb-22	359.5	362.5

等效数据透视表类似如下，每个单独年度月份的列包含在较大的 Month Year 行中：

带水平透视 Month Year 字段的等效透视表(裁剪)

面积(值)	Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
APAC - 实际	435	434	397	404	458	447	413	458	385	421	448	397
APAC - 目标	425	425	425	425	425	425	425	425	425	425	425	425
EMEA - 实际	363.5	359.5	337.5	361.5	341.5	337.5	379.5	352.5	327.5	337.5	360.5	334.5
EMEA - 目标	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5
NA - 实际	378	415	363	356	403	343	401	365	393	340	360	405
NA - 目标	375	375	375	375	375	375	375	375	375	375	375	375

显示为表和等效数据透视表的数据示例，该透视表具有水平透视的 Month Year



Table

Area	Month Year	Actual	Target
Totals		13812	13950
APAC	Jan-22	435	425
APAC	Feb-22	434	425
APAC	Mar-22	397	425
APAC	Apr-22	404	425
APAC	May-22	458	425
APAC	Jun-22	447	425
APAC	Jul-22	413	425
APAC	Aug-22	458	425
APAC	Sep-22	385	425
APAC	Oct-22	421	425
APAC	Nov-22	448	425

Pivot table

Area / Values / Month Year

	Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
APAC - Actual	435	434	397	404	458	447	413	458	385	421	448	397
APAC - Target	425	425	425	425	425	425	425	425	425	425	425	425
EMEA - Actual	363.5	359.5	337.5	361.5	341.5	337.5	379.5	352.5	327.5	337.5	360.5	334.5
EMEA - Target	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5
NA - Actual	378	415	363	356	403	343	401	365	393	340	360	405
NA - Target	375	375	375	375	375	375	375	375	375	375	375	375

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。使用 First 前缀的一个典型用例是，当您希望从大型和/或慢速数据加载步骤中检索一小部分记录时。一旦加载了定义的“n”数量的记录，加载步骤就会过早终止，其余脚本执行将正常继续。

语法：

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

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 也可以用括号括起来：(n)。
loadstatement | selectstatement	n 参数后面的 load statement/select statement 将定义必须加载设置的最大数目记录的指定表。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
FIRST 10 LOAD * from abc.csv;	本示例将从 excel 文件中检索前十行。
FIRST (1) SQL SELECT * from Orders;	此示例将从 Orders 数据集中检索第一个选定行。

示例 – 加载前五行

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 2020 年前两周的数据集。
- 指示应用程序仅加载前五条记录的 First 变量。

加载脚本

```
Sales:
FIRST 5
LOAD
*
Inline [
date,sales
01/01/2020,6000
01/02/2020,3000
01/03/2020,6000
01/04/2020,8000
01/05/2020,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
01/12/2020,7000
01/13/2020,7000
01/14/2020,7000
];
```

结果

加载数据并打开工作表。创建新表并添加 Date 为字段并添加 sum(sales) 为度量。

结果表

日期	sum(sales)
01/01/2020	6000
01/02/2020	3000
01/03/2020	6000
01/04/2020	8000
01/05/2020	5000

脚本只加载 Sales 表的前五条记录。

Generic

Generic 加载前缀允许将实体-属性-值建模数据 (EAV) 转换为传统的规范化关系表结构。EAV 建模也称为“通用数据建模”或“开放模式”。

EAV 建模数据和等效非规范化关系表示例

Product ID	Attribute	Value
13	Status	Discontinued
13	Colour	Brown
20	Colour	White
13	Size	13-15
20	Size	16-18

Product ID	Status	Colour	Size
13	Discontinued	Brown	13-15
20		White	16-18

EAV 建模数据和一组等效的规范化关系表示例

Product ID	Attribute	Value
13	Status	Discontinued
13	Colour	Brown
20	Colour	White
13	Size	13-15
20	Size	16-18

Product ID	Status
13	Discontinued

Product ID	Colour
13	Brown
20	White

Product ID	Size
13	13-15
20	16-18

虽然在技术上可以加载和分析 Qlik 中 EAV建模的数据，但使用等效的传统关系数据结构通常更容易。

语法：

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

这些主题可以帮助您使用此函数：

相关主题

主题	说明
Crosstable (page 44)	Crosstable 加载前缀将水平方向的数据转换为垂直方向的数据。从纯特性的角度来看，它执行与 Generic 加载前缀相反的转换，尽管前缀通常服务于完全不同的用例。
通用数据库，位于 *管理数据*	这里进一步描述了 EAV 结构化数据模型。

示例 1 – 使用通用加载前缀转换 EAV 结构化数据

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含一个数据集，该数据集加载到名为 Transactions 的表中。数据表包括一个日期字段。使用了默认 MonthNames 定义。

加载脚本

```
Products:
Generic
Load * inline [
Product ID, Attribute, Value
13, Status, Discontinued
13, Color, Brown
20, Color, White
13, Size, 13-15
20, Size, 16-18
2, Status, Discontinued
5, Color, Brown
2, Color, White
44, Color, Brown
45, Size, 16-18
45, Color, Brown
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：Color。

添加该度量：

```
=Count([Product ID])
```

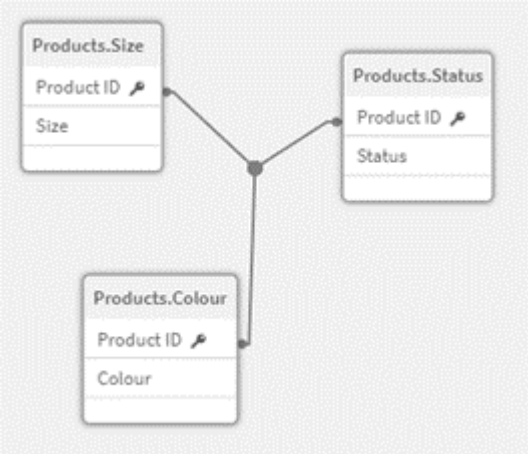
现在您可以按颜色检查产品数量。

结果表

颜色	=Count([Product ID])
棕色	4
白色	2

注意数据模型的形状，其中每个属性都被分解为一个单独的表，根据原始目标表标记 `Product` 命名。每个表都有一个属性作为后缀。这方面的一个示例为 `Product.Color`。生成的产品属性输出记录由 `Product ID` 关联。

结果的数据模型查看器表示



所得记录表：Products.Status

产品 ID	状态
13	中断
2	中断

所得记录表：Products.Size

产品 ID	大小
13	13-15
20	16-18
45	16-18

所得记录
表：Products.Col
or

产品 ID	颜色
13	棕色
5	棕色
44	棕色
45	棕色
20	白色
2	白色

示例 2 – 分析没有通用加载前缀的EAV结构化数据

加载脚本和图表表达式

概述

本示例演示如何分析原始形式的 EAV 结构化数据。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含一个数据集，该数据集加载到名为 Products 的采用 EAV 结构的表中。

在本例中，我们仍然按颜色属性计算产品数量。为了分析以这种方式结构化的数据，需要对带有属性值 Color 的产品应用表达式级筛选。

此外，单个属性不可选择为维度或字段，因此更难确定如何构建有效的可视化。

加载脚本

```
Products:
Load * Inline
[
Product ID, Attribute, Value
13, Status, Discontinued
13, Color, Brown
20, Color, White
13, Size, 13-15
20, Size, 16-18
2, Status, Discontinued
5, Color, Brown
2, Color, White
44, Color, Brown
45, Size, 16-18
45, Color, Brown
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：Value。

创建以下度量：

```
=Count({<Attribute={'Color'}>} [Product ID])
```

现在您可以按颜色检查产品数量。

所得记录表：Products.Status

值	=Count({<Attribute={'Color'}>} [Product ID])
棕色	4
白色	2

示例 3 – 从通用加载中去规范化结果输出表(高级)

加载脚本和图表表达式

概述

在本例中，我们展示了如何将 Generic 加载前缀生成的规范化数据结构反规范化回合并 Product 维度表。这是一种高级建模技术，可以作为数据模型性能调整的一部分使用。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本

```
Products:

Generic
Load * inline [
Product ID, Attribute, Value
13, Status, Discontinued
13, Color, Brown
20, Color, White
13, Size, 13-15
20, Size, 16-18
2, Status, Discontinued
5, Color, Brown
2, Color, White
44, Color, Brown
45, Size, 16-18
45, Color, Brown
];

RENAME TABLE Products.Color TO Products;

OUTER JOIN (Products)
LOAD * RESIDENT Products.Size;
```

```
OUTER JOIN (Products)
LOAD * RESIDENT Products.Status;
DROP TABLES Products.Size,Products.Status;
```

结果

打开数据模型查看器，并记录结果数据模型的形状。仅存在一个非规范化表。它是三个中间输出表的组合：Products.Size、Products.Status 和 Products.Color。

得到内部数据模型

产品
产品 ID
状态
颜色
大小

所得记录表：产品

产品 ID	**状态**	**颜色**	**大小**
13	中断	棕色	13-15
20	-	白色	16-18
2	中断	白色	-
5	-	棕色	-
44	-	棕色	-
45	-	棕色	16-18

加载数据并打开工作表。创建新表并将该字段添加为维度：Color。

添加该度量：

```
=Count([Product ID])
```

结果表

颜色	**=Count([Product ID])**
棕色	4
白色	2

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

列 1	列 2	列 3
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。

联接数据的效果是通过一组额外的字段或属性扩展目标表，即目标表中不存在的字段或特性。源数据集和目标表之间的任何公共字段名都用于确定如何关联新的传入记录。这通常被称为“自然联接”。根据联接关联的唯一性和使用的联接类型，Qlik 联接操作可能导致生成的目标表具有比开始时更多或更少的记录。

有四种类型的联接：

左侧联接

左联接是最常见的联接类型。例如，如果您有一个事务数据集，并希望将其与参考数据集组合，则通常会使用 `Left Join`。首先加载事务表，然后加载引用数据集，同时通过 `Left Join` 前缀将其连接到已加载的事务表。`Left Join` 将保持所有交易的原样，并添加找到匹配项的补充参考数据字段。

内部联接

当您有两个数据集，其中您只关心存在匹配关联的任何结果时，请考虑使用 `Inner Join`。如果未找到匹配项，这将从加载的源数据和目标表中删除所有记录。因此，这可能会使目标表中的记录比连接操作发生之前更少。

外部联接

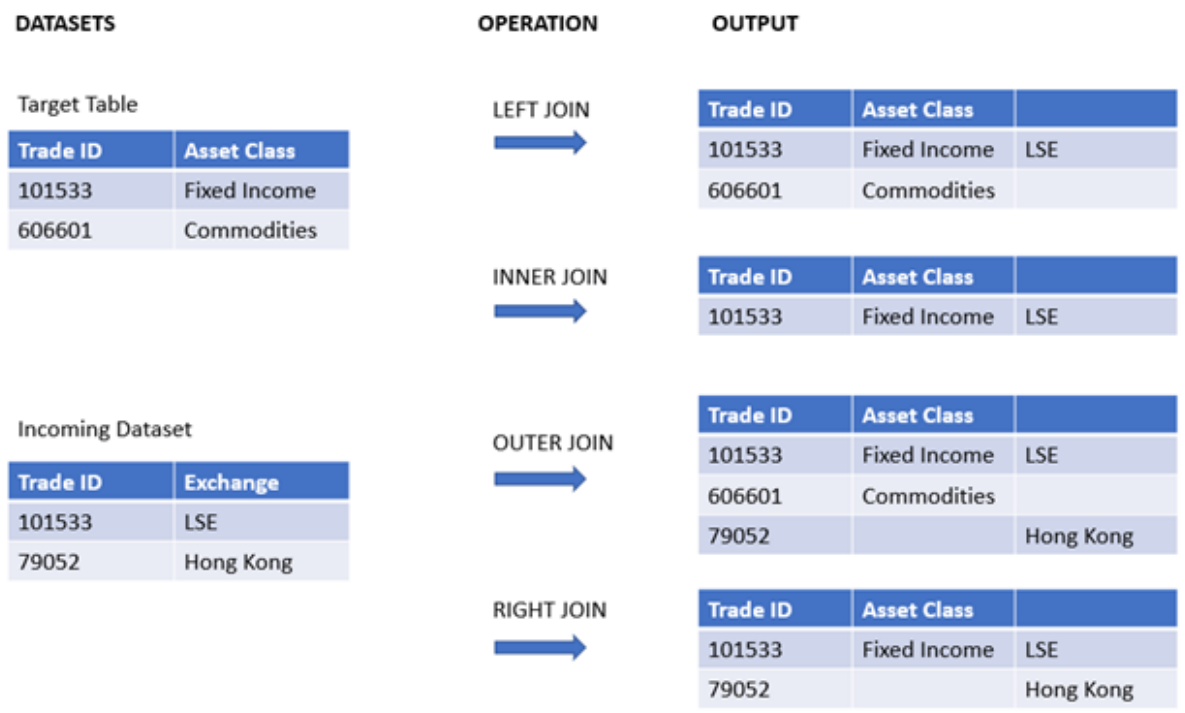
当您需要同时保留目标记录和所有传入记录时，请使用 `Outer Join`。如果未找到匹配项，则仍保留每组记录，而连接另一侧的字段将保持未填充(空)。

如果省略类型关键字，则默认联接类型为外部联接。

右侧联接

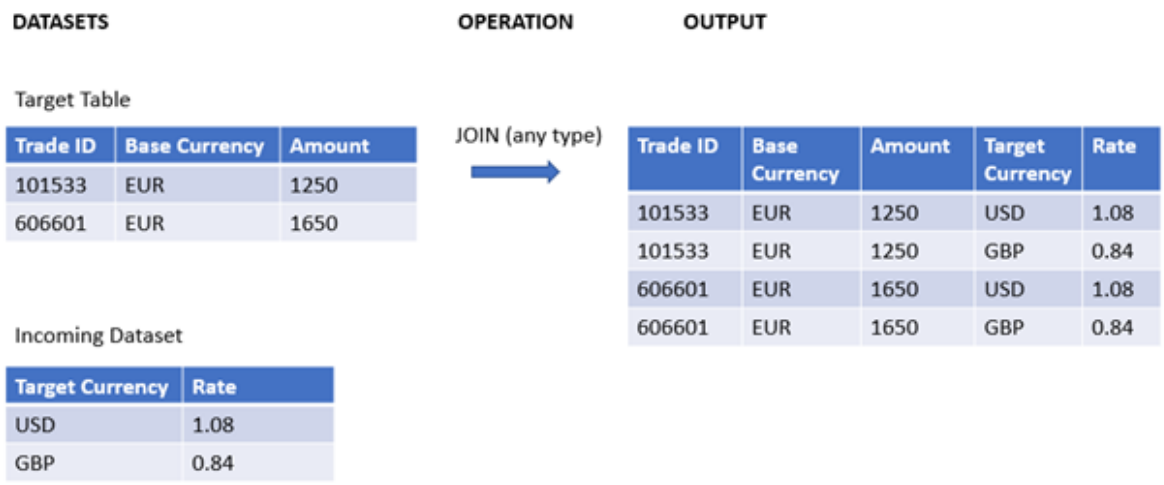
此联接类型保留所有将要加载的记录，同时将联接所针对的表中的记录减少到只有传入记录中存在关联匹配的记录。这是一种利基联接类型，有时用于将已预加载的记录表缩减为所需子集。

来自不同类型联接操作的示例结果集



如果联接操作的源和目标之间没有共同的字段名，则联接将导致所有行的笛卡尔乘积，这称为"交叉联接"。

"交叉联接"操作的示例结果集



语法：

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

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。

这些主题可以帮助您使用此函数:

相关主题

主题	说明
使用 Join 和 Keep 在 *管理数据* 中合并表格	本主题进一步解释了“连接”和“保留”数据集的概念。
Keep (page 78)	Keep 加载前缀类似于 Join 前缀，但它不组合源数据集和目标数据集。相反，它根据所采用的操作类型(内部、外部、左侧或右侧)修剪每个数据集。

示例 1 - 左联接:使用参考数据集丰富目标表

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 表示更改记录的数据集，该数据集加载到名为 Changes 的表中。它包括一个状态 ID 关键字段。
- 表示变更状态的第二数据集，通过将其与左 Join加载前缀连接来加载并与原始变更记录组合。

此左联接确保更改记录保持完整，同时添加状态属性，其中根据公共状态 ID 找到传入状态记录中的匹配项。

加载脚本

```
Changes:
Load * inline [
Change ID      Status ID       Scheduled Start Date    Scheduled End Date      Business Impact
10030  4       19/01/2022      23/02/2022      None
10015  3       04/01/2022      15/02/2022      Low
10103  1       02/04/2022      29/05/2022      Medium
10185  2       23/06/2022      08/09/2022      None
10323  1       08/11/2022      26/11/2022      High
10326  2       11/11/2022      05/12/2022      None
```

```
10138  2       07/05/2022      03/08/2022      None
10031  3       20/01/2022      25/03/2022      Low
10040  1       29/01/2022      22/04/2022      None
10134  1       03/05/2022      08/07/2022      Low
10334  2       19/11/2022      06/02/2023      Low
10220  2       28/07/2022      06/09/2022      None
10264  1       10/09/2022      17/10/2022      Medium
10116  1       15/04/2022      24/04/2022      None
10187  2       25/06/2022      24/08/2022      Low
] (delimiter is '\t');

Status:
Left Join (Changes)
Load * inline [
Status ID      Status  Sub Status
1      Open    Not Started
2      Open    Started
3      Closed  Completed
4      Closed  Cancelled
5      Closed  Obsolete
] (delimiter is '\t');
```

结果

打开数据模型查看器，并记录数据模型的形状。仅存在一个非规范化表。它是所有原始变更记录的组合，匹配的状态属性连接到每个变更记录上。

得到内部数据模型

更改
更改 ID
状态 ID
计划的开始日期
计划的结束日期
业务影响
状态
子状态

如果在数据模型查看器中展开预览窗口，您将看到整个结果集的一部分被组织到一个表中：

在数据模型查看器中预览更改表

更改 ID	状态 ID	计划的开始日期	计划的结束日期	业务影响	状态	子状态
10030	4	19/01/2022	23/02/2022	无	关闭	已取消
10031	3	20/01/2022	25/03/2022	低	关闭	已完成
10015	3	04/01/2022	15/02/2022	低	关闭	已完成

更改 ID	状态 ID	计划的开始日期	计划的结束日期	业务影响	状态	子状态
10103	1	02/04/2022	29/05/2022	中等	打开	未开始
10116	1	15/04/2022	24/04/2022	无	打开	未开始
10134	1	03/05/2022	08/07/2022	低	打开	未开始
10264	1	10/09/2022	17/10/2022	中等	打开	未开始
10040	1	29/01/2022	22/04/2022	无	打开	未开始
10323	1	08/11/2022	26/11/2022	高	打开	未开始
10187	2	25/06/2022	24/08/2022	低	打开	已开始
10185	2	23/06/2022	08/09/2022	无	打开	已开始
10220	2	28/07/2022	06/09/2022	无	打开	已开始
10326	2	11/11/2022	05/12/2022	无	打开	已开始
10138	2	07/05/2022	03/08/2022	无	打开	已开始
10334	2	19/11/2022	06/02/2023	低	打开	已开始

由于 Status 表中的第五行(Status ID:'5'、Status:'Closed'、Sub Status:'Obsolete')与 Changes 表中的任何记录都不对应,因此此行中的信息不会出现在上面的结果集中。

返回至数据加载编辑器。加载数据并打开工作表。创建新表并将该字段添加为维度:Status。

添加该度量:

```
=Count([Change ID])
```

现在您可以按状态检查更改的数量。

结果表

状态	=Count([Change ID])
打开	12
关闭	3

示例 2 – 内部联接:仅合并匹配记录

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 表示更改记录的数据集，该数据集加载到名为 Changes 的表中。
- 第二个数据集表示源系统 JIRA 中的变更记录。通过使用 Inner Join 加载前缀将其与原始记录连接，将其加载并合并。

该 Inner Join 确保只保留两个数据集中的五个变更记录。

加载脚本

```
Changes:
Load * inline [
Change ID      Status ID       Scheduled Start Date    Scheduled End Date      Business Impact
10030  4       19/01/2022      23/02/2022      None
10015  3       04/01/2022      15/02/2022      Low
10103  1       02/04/2022      29/05/2022      Medium
10185  2       23/06/2022      08/09/2022      None
10323  1       08/11/2022      26/11/2022      High
10326  2       11/11/2022      05/12/2022      None
10138  2       07/05/2022      03/08/2022      None
10031  3       20/01/2022      25/03/2022      Low
10040  1       29/01/2022      22/04/2022      None
10134  1       03/05/2022      08/07/2022      Low
10334  2       19/11/2022      06/02/2023      Low
10220  2       28/07/2022      06/09/2022      None
10264  1       10/09/2022      17/10/2022      Medium
10116  1       15/04/2022      24/04/2022      None
10187  2       25/06/2022      24/08/2022      Low
] (delimiter is '\t');

JIRA_changes:
Inner Join (Changes)
Load
  [Ticket ID] AS [Change ID],
  [Source System]
inline
[
Ticket ID      Source System
10000  JIRA
10030  JIRA
10323  JIRA
10134  JIRA
10334  JIRA
10220  JIRA
20000  TFS
] (delimiter is '\t');
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- Source System
- Change ID
- Business Impact

现在您可以检查五个结果记录。来自 Inner Join 的结果表将只包括两个数据集中具有匹配信息的记录。

结果表

源系统	更改 ID	业务影响
JIRA	10030	无
JIRA	10134	低
JIRA	10220	无
JIRA	10323	高
JIRA	10334	低

示例 3 – 外部联接:合并重叠记录集

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 表示更改记录的数据集,该数据集加载到名为 Changes 的表中。
- 第二个数据集表示源系统 JIRA 中的变更记录。通过使用 Outer Join 加载前缀将其与原始记录连接,将其加载并合并。

这确保了来自两个数据集的所有重叠更改记录都被保留。

加载脚本

```
// 8 Change records

Changes:
Load * inline [
Change ID      Status ID       Scheduled Start Date    Scheduled End Date      Business Impact
10030  4       19/01/2022      23/02/2022      None
10015  3       04/01/2022      15/02/2022      Low
10138  2       07/05/2022      03/08/2022      None
10031  3       20/01/2022      25/03/2022      Low
10040  1       29/01/2022      22/04/2022      None
10134  1       03/05/2022      08/07/2022      Low
10334  2       19/11/2022      06/02/2023      Low
10220  2       28/07/2022      06/09/2022      None
] (delimiter is '\t');

// 6 Change records
```

```
JIRA_changes:
Outer Join (Changes)
Load
  [Ticket ID] AS [Change ID],
  [Source System]
inline
[
Ticket ID      Source System
10030  JIRA
10323  JIRA
10134  JIRA
10334  JIRA
10220  JIRA
10597  JIRA
] (delimiter is '\t');
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- Source System
- Change ID
- Business Impact

现在您可以检查 10 个结果记录。

结果表

源系统	更改 ID	业务影响
JIRA	10030	无
JIRA	10134	低
JIRA	10220	无
JIRA	10323	-
JIRA	10334	低
JIRA	10597	-
-	10015	低
-	10031	低
-	10040	无
-	10138	无

示例 4 – 右联结:通过辅助主数据集修剪目标表

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 表示更改记录的数据集,该数据集加载到名为 Changes 的表中。
- 第二数据集表示源系统 Teamwork 的变更记录。通过将其与 Right Join 加载前缀连接,将其加载并与原始记录组合。

这确保只保留 Teamwork 更改记录,而如果目标表没有匹配 Change ID,则不会丢失任何 Teamwork 记录。

加载脚本

```
Changes:

Load * inline [
Change ID      Status ID       Scheduled Start Date    Scheduled End Date      Business Impact
10030  4       19/01/2022      23/02/2022      None
10015  3       04/01/2022      15/02/2022      Low
10103  1       02/04/2022      29/05/2022      Medium
10185  2       23/06/2022      08/09/2022      None
10323  1       08/11/2022      26/11/2022      High
10326  2       11/11/2022      05/12/2022      None
10138  2       07/05/2022      03/08/2022      None
10031  3       20/01/2022      25/03/2022      Low
10040  1       29/01/2022      22/04/2022      None
10134  1       03/05/2022      08/07/2022      Low
10334  2       19/11/2022      06/02/2023      Low
10220  2       28/07/2022      06/09/2022      None
10264  1       10/09/2022      17/10/2022      Medium
10116  1       15/04/2022      24/04/2022      None
10187  2       25/06/2022      24/08/2022      Low
] (delimiter is '\t');

Teamwork_changes:
Right Join (Changes)
Load
  [Ticket ID] AS [Change ID],
  [Source System]
inline
[
Ticket ID      Source System
10040  Teamwork
10015  Teamwork
10103  Teamwork
10031  Teamwork
```

```
50231  Teamwork
] (delimiter is '\t');
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- Source System
- Change ID
- Business Impact

现在您可以检查五个结果记录。

结果表

源系统	更改 ID	业务影响
团队工作	10015	低
团队工作	10031	低
团队工作	10040	无
团队工作	10103	中等
团队工作	50231	-

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 1399)

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

列 1	列 2	列 3
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	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，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 **Concatenate** 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：

Insert：**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。

Delete：缺少字段不会影响结果。相关记录仍将被删除。

Update：**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。

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

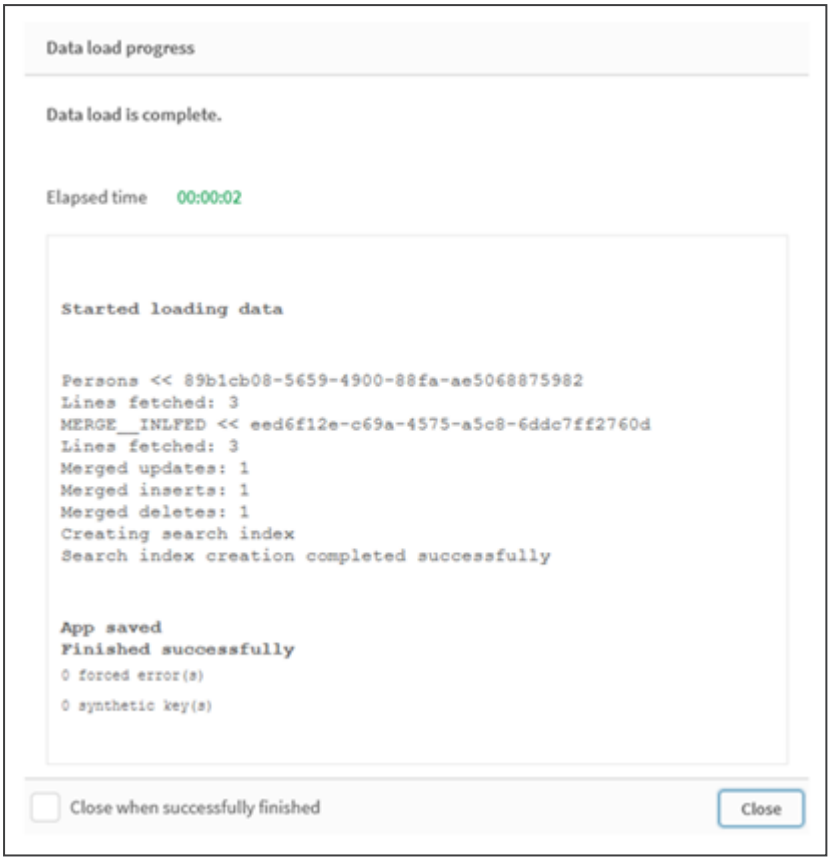
在 **Merge Load** 之后，表格如下显示：

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 )
```

默认情况下，如果加载的表包含相同数量的字段，并且字段名称与脚本中先前加载的表匹配，则 Qlik Sense 将自动串联这两个表。即使第二个表的名称不同，也会发生这种情况。

但是，如果脚本前缀 `NoConcatenate` 包含在第二个表的 Load 语句或 select 语句之前，则这两个表将分别加载。

NoConcatenate的一个典型用例是，您可能需要创建表的临时副本以对该副本执行一些临时转换，同时保留原始数据的副本。NoConcatenate 确保您可以创建该副本，而无需将其隐式添加回源表。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数

示例	结果
Source: LOAD A,B from file1.csv; CopyOfSource: NoConcatenate LOAD A,B resident Source;	加载带有 A 和 B 作为度量的表。使用 NoConcatenate 变量单独加载具有相同字段的第二个表。

示例 1 – 隐式串联

加载脚本和结果

概述

在本例中，您将按顺序加载两个加载脚本。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 一个初始数据集，其中包含发送到名为 Transactions 的表的日期和金额。

第一个加载脚本

```
Transactions:
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount

第一个结果表

id	日期	金额
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

第二个加载脚本

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 具有相同字段的第二个数据集被发送到名为 Sales 的表。

```
Sales:
LOAD
*
Inline [
id, date, amount
8, 10/01/2018, 164.27
9, 10/03/2018, 384.00
10, 10/06/2018, 25.82
11, 10/09/2018, 312.00
12, 10/15/2018, 4.56
13, 10/16/2018, 90.24
14, 10/18/2018, 19.32
];
```

结果

加载数据并转到表。

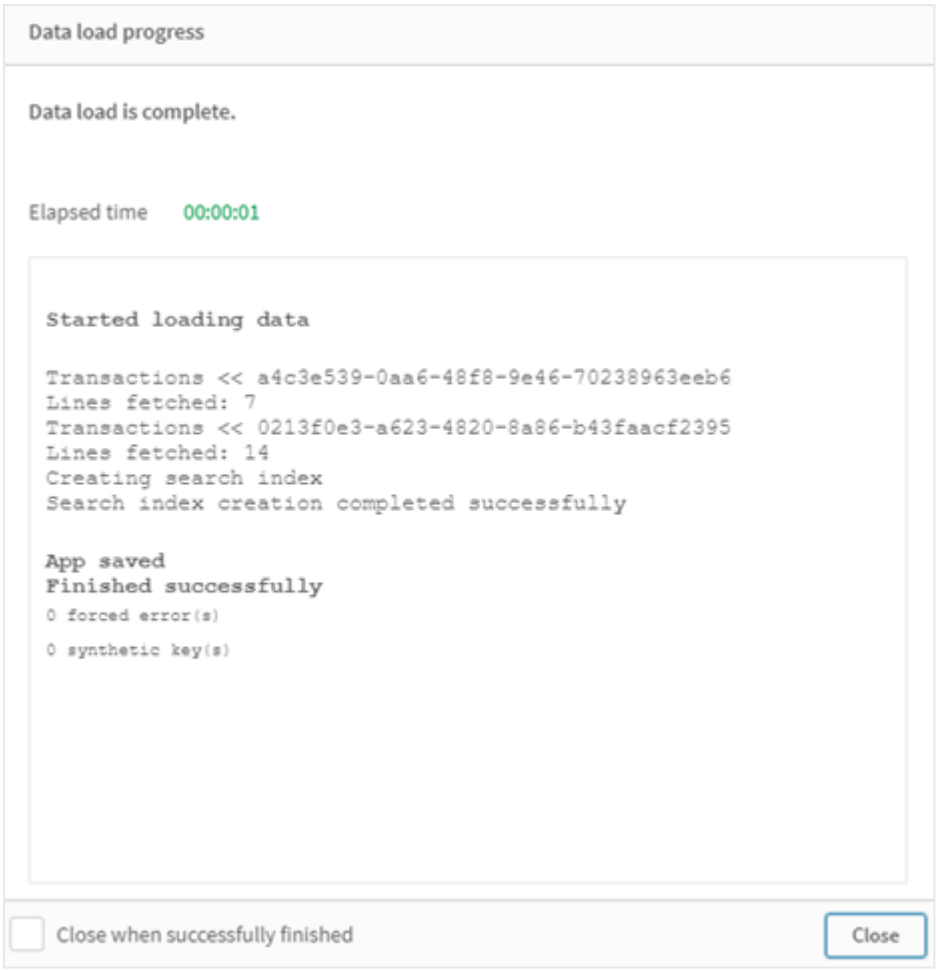
第二个结果表

id	日期	金额
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42
8	10/01/2018	164.27
9	10/03/2018	384.00
10	10/06/2018	25.82
11	10/09/2018	312.00
12	10/15/2018	4.56
13	10/16/2018	90.24
14	10/18/2018	19.32

当脚本运行时，由于两个数据集共享相同数量的字段和相同的字段名，Sales 表隐式串联到现有 Transactions 表上。尽管第二个表名标记试图命名结果集 'Sales'，但仍会发生这种情况。

通过查看**数据加载进度**日志，可以看到销售额数据集是隐式串联的。

显示隐式串联的事务数据的数据加载进度日志。



示例 2 – 用例场景

加载脚本和结果

概述

在这个用例场景中，您有：

- 一个交易数据集，其具有：
  - id
  - 日期
  - 金额 (GBP)
- 一个货币表：
  - USD 到 GBP 的转换率
- 第二个交易数据集，其具有：
  - id

- 日期
- 金额 (USD)

您将按顺序加载五个脚本。

- 第一个加载脚本包含一个初始数据集，其中包含发送到名为 Transactions 的表的日期和金额 (GBP)。
- 第二个加载脚本包含：
  - 第二个初始数据集，其中包含发送到名为 Transactions_in_USD 的表的日期和金额 (USD)。
  - 放置在 Transactions_in_USD 数据集的 Load 语句之前以防止隐式串联的 noconcatenate 前缀。
- 第三个加载脚本包含 join 前缀，用于在 Transactions_in_USD 表中创建 GBP 和 USD 之间的货币汇率。
- 第四个加载脚本包含将 Transactions_in_USD 添加到初始 Transactions 表的 concatenate 前缀。
- 第五个加载脚本包含 drop table 语句，该语句将删除其数据已串联到 Transactions 表的表 Transactions_in_USD。

第一个加载脚本

```
Transactions:

Load * Inline [
id, date, amount
1, 12/30/2018, 23.56
2, 12/07/2018, 556.31
3, 12/16/2018, 5.75
4, 12/22/2018, 125.00
5, 12/22/2018, 484.21
6, 12/22/2018, 59.18
7, 12/23/2018, 177.42
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount

第一个加载脚本结果

id	日期	金额
1	12/30/2018	23.56
2	12/07/2018	556.31

id	日期	金额
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42

该表显示了初始数据集，带有按 GBP 计的金额。

第二个加载脚本

```
Transactions_in_USD:
NoConcatenate
Load * Inline [
id, date, amount
8, 01/01/2019, 164.27
9, 01/03/2019, 384.00
10, 01/06/2019, 25.82
11, 01/09/2019, 312.00
12, 01/15/2019, 4.56
13, 01/16/2019, 90.24
14, 01/18/2019, 19.32
];
```

结果

加载数据并转到表。

第二个加载脚本结果

id	日期	金额
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42
8	01/01/2019	164.27
9	01/03/2019	384.00
10	01/06/2019	25.82
11	01/09/2019	312.00

id	日期	金额
12	01/15/2019	4.56
13	01/16/2019	90.24
14	01/18/2019	19.32

您将看到 Transactions_in_USD 表中的第二个数据集已添加。

第三个加载脚本

此加载脚本将从 USD 到 GBP 的货币汇率加入到 Transactions_in_USD 表中。

```
Join (Transactions_in_USD)
Load * Inline [
rate
0.7
];
```

结果

加载数据并转到数据模型查看器。选择 Transactions_in_USD 表，您将看到每个现有记录的“汇率”字段值为 0.7。

第四个加载脚本

使用常驻加载，此加载脚本将在将金额转换为 USD 后将 Transactions_in_USD 表格串联到 Transactions 表格。

```
Concatenate (Transactions)
LOAD
id,
date,
amount * rate as amount
Resident Transactions_in_USD;
```

结果

加载数据并转到表。您将看到第八行至第十四行以 GBP 为单位的新条目。

第四个加载脚本结果

id	日期	金额
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21

id	日期	金额
6	12/22/2018	59.18
7	12/23/2018	177.42
8	01/01/2019	114.989
8	01/01/2019	164.27
9	01/03/2019	268.80
9	01/03/2019	384.00
10	01/06/2019	18.074
10	01/06/2019	25.82
11	01/09/2019	218.40
11	01/09/2019	312.00
12	01/15/2019	3.192
12	01/15/2019	4.56
13	01/16/2019	63.168
13	01/16/2019	90.24
14	01/18/2019	13.524
14	01/18/2019	19.32

第五个加载脚本

此加载脚本将删除第四个加载脚本结果表中的重复条目，只留下金额按 GBP 计的条目。

```
drop tables Transactions_in_USD;
```

结果

加载数据并转到表。

第五个加载脚本结果

id	日期	金额
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18

id	日期	金额
7	12/23/2018	177.42
8	01/01/2019	114.989
9	01/03/2019	268.80
10	01/06/2019	18.074
11	01/09/2019	218.40
12	01/15/2019	3.192
13	01/16/2019	63.168
14	01/18/2019	13.524

加载第五个加载脚本后，结果表显示了两个事务数据集中存在的所有十四个事务；然而，交易 8-14 的金额已转换为 GBP。

如果我们在第二个加载脚本中删除在 Transactions_in_USD之前使用的 NoConcatenate 前缀，脚本将失败，并显示错误："找不到表 'Transactions_in_USD' "。这是因为 Transactions_in_USD 表将自动串联到原始 Transactions 表上。

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

列 1	列 2	列 3
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

语句	完全重新加载	部分加载
添加/替换/合并加载 ...	语句将会运行	语句将会运行
添加/替换/仅合并加载 ...	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。

限制

如果在完全重新加载期间存在引用表的命令，但在部分重新加载期间没有，则部分重新加载将失败。

示例

示例命令

```
LEFT JOIN(<Table_removed_after_full_reload>)
CONCATENATE(<Table_removed_after_full_reload>)
```

其中，<Table_removed_after_full_reLoad> 是一个存在于完全重新加载中但不存在于部分重新加载中的表。

解决方法

作为解决方法，可以使用以下 if 语句围绕命令：

```
IF NOT IsPartialReload() THEN ... ENDIF.
```

部分重新加载可以从数据中删除值。但是，这不会反映在不同值的列表中，这是一个内部维护的表。因此，在部分重新加载后，列表将包含自上次完全重新加载以来字段中存在的所有不同值，这些值可能会超过部分重新加载后当前存在的值。这会影响 FieldValueCount() 和 FieldValue() 函数的输出。FieldValueCount() 可能返回一个大于当前字段值数量的数字。

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
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	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，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。

示例	结果
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 1411)*

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

列 1	列 2	列 3
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 )
```

所计算的表达式不定义将加载到 Qlik Sense 应用程序中的数据集中记录的百分比，而是定义将读取的每个记录加载到应用中的概率。换句话说，指定值 `p = 0.5`并不意味着将加载总记录数的 50%，而是意味着每个记录将有 50% 的机会加载到 Qlik Sense 应用程序中。

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。

适用场景

当您希望对来自大表的数据进行采样，以了解数据、分布或字段内容的性质时，示例非常有用。由于它带来了数据子集，数据加载速度更快，从而可以更快地测试脚本。与 `First` 不同，`Sample` 函数从整个表中获取数据，而不是仅限于前几行。在某些情况下，这可以提供更准确的数据表示。

以下示例显示了 `Sample` 脚本前缀的两种可能用法：

```
Sample 0.15 SQL SELECT * from Longtable;

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

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 内联表示例

加载脚本和结果

概述

在本例中，脚本将包含七条记录的数据集中的数据样本集加载到内联表中名为 Transactions 的表中。

加载脚本

```
Transactions:
SAMPLE 0.3
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- amount

添加以下度量：

```
=sum(amount)8
```

结果表

id	日期	=Sum(amount)
2	09/07/2018	556.31
4	09/22/2018	125
1	08/30/2018	23.56
3	09/16/2018	5.75

在本例中使用的加载迭代中，读取了所有七条记录，但只有四条记录被加载到数据表中。任何重新运行的加载都可能导致不同的数量，以及不同的记录集被加载到应用程序中。

示例 2 – 来自自动生成表格的示例

加载脚本和结果

概述

在本例中，使用 Autogenerate 创建了包含字段 date、id 和 amount 的 100 条记录的数据集。但是，使用了 Sample 前缀，其值为 0.1。

加载脚本

```
SampleData:
Sample 0.1
LOAD
RecNo() AS id,
MakeDate(2013, Ceil(Rand() * 12), Ceil(Rand() * 29)) as date,
Rand() * 1000 AS amount

Autogenerate(100);
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- amount

添加以下度量：

结果表

id	日期	=Sum(amount)
48	9/28/2013	763
20	5/15/2013	752
19	11/8/2013	657

id	日期	=Sum(amount)
25	3/24/2013	522
27	8/23/2013	389
81	6/1/2013	53
100	8/15/2013	17

在本例中使用的加载迭代中，从创建的数据集中加载了七条记录。同样，任何重新运行的加载都可能导致不同的数量，以及不同的记录集被加载到应用程序中。

Semantic

semantic 加载前缀创建了一种特殊类型的字段，可用于 Qlik Sense 连接和管理关系数据，如树结构、自引用父子结构化数据和/或可描述为图形的数据。

注意，semantic 加载的功能与 *Hierarchy (page 61)* 和 *HierarchyBelongsTo (page 62)* 前缀类似。所有三个前缀都可以用作有效的前端解决方案中的构建块，用于遍历关系数据。

语法：

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

语义加载要求输入正好有三到四个字段宽，并严格定义每个有序字段表示的内容，如下表所示：

语义加载字段

字段名称	字段描述
第 1 字段：	此标记表示两个对象之间存在关系的第一个对象。
第 2 字段：	该标签将用于描述第一和第二对象之间的“正向”关系。如果第一个对象是子对象，而第二个对象是父对象，则可以创建一个关系选项卡，该选项卡状态为“父对象”或“父对象”，就像您遵循从子对象到父对象的关系一样。
第 3 字段：	此标记表示两个对象之间存在关系的第二个对象。
第 4 字段：	该字段为选填。该标签描述了第一个和第二个对象之间的“反向”或“反向”关系。如果第一个对象是子对象，而第二个对象是父对象，则关系选项卡可以显示“子对象”或“子对象”，就像您从父对象跟踪到子对象一样。如果不添加第四个字段，则第二个字段标记将用于描述两个方向的关系。在这种情况下，箭头符号将自动添加为标记的一部分。

以下代码是 semantic 前缀的示例。

```
Semantic
Load
Object,
‘Parent’ AS Relationship,
NeighbouringObject AS Object,
```

```
'Child' AS Relationship
from graphdata.csv;
```

> *允许和典型的做法是将第三个字段标记为与第一个字段相同。这将创建一个自引用查找，以便您可以一步一步地跟踪对象到相关对象。如果第三个字段不具有相同的名称，则最终结果将是从一个对象到其直接关系邻居的简单查找，仅一步之遥，这是几乎没有实际用途的输出。*

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

相关函数

函数	交互
Hierarchy (page 61)	Hierarchy 加载前缀用于划分和组织父子和其他类似图的数据结构中的节点，并将它们转换为表。
HierarchyBelongsTo (page 62)	HierarchyBelongsTo 加载前缀用于定位和组织父子和其他类似图的数据结构的祖先，并将它们转换为表。

示例 - 使用语义前缀创建连接关系的特殊字段

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 表示地理位置关系记录的数据集，该数据集加载到名为 GeographyTree 的表中。
    - 每个条目在行首有一个 ID，在行尾有一个 ParentID。
- 将添加一个标记的特殊行为字段的 semantic 前缀 Relation。

加载脚本

```
GeographyTree:
LOAD
    ID,
    Geography,
```

```
    if(ParentID='',null(),ParentID) AS ParentID

INLINE [
ID,Geography,ParentID
1,World
2,Europe,1
3,Asia,1
4,North America,1
5,South America,1
6,UK,2
7,Germany,2
8,Sweden,2
9,South Korea,3
10,North Korea,3
11,China,3
12,London,6
13,Birmingham,6
];

SemanticTable:
Semantic Load
       ID as ID,
       'Parent' as Relation,
       ParentID as ID,
       'Child' as Relation
resident GeographyTree;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度。

- Id
- Geography

然后，创建一个以 Relation 作为维度的筛选窗格。单击**完成编辑**。

结果表

ID	Geography
1	World
2	Europe
3	Asia
4	North America
5	South America
6	UK
7	Germany
8	Sweden

ID	Geography
9	South Korea
10	North Korea
11	中国
12	London
13	Birmingham

筛选器窗格

关系

子

父

在表中的 Geography 维度中单击**欧洲**，然后在筛选器窗格中的 Relation 维度上单击**儿童**。注意表中的预期结果：

显示欧洲“儿童”的结果表

ID	Geography
6	UK
7	Germany
8	Sweden

再次单击**儿童**将显示英国的“儿童”位置，再向下一步。

显示英国“儿童”的结果表

ID	Geography
12	London
13	Birmingham

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	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。

适用场景

Unless 语句返回布尔值结果。通常，当用户希望有条件地加载或排除脚本的部分时，这种类型的函数将用作条件。

以下几行显示了如何使用 Unless 函数的三个示例：

```
exit script unless A=1;

unless A=1 LOAD * from myfile.csv;

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

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – Unless 前缀

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 创建变量 A，其值为 1。
- 加载到名为 Transactions 的表中的数据集，除非变量 A=2。

加载脚本

```
LET A = 1;
```

```
UNLESS A = 2

Transactions:
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount

结果表

id	日期	金额
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

因为变量 A 在脚本开始时被赋值 1，所以会计算前缀 Unless 后面的条件，并返回结果 FALSE。因此，脚本将继续运行 Load 语句。在结果表中，可以看到 Transactions 表中的所有记录。

如果此变量值设置为等于 2，则不会将任何数据加载到数据模型中。

示例 2 – Unless 后缀

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本首先将初始数据集加载到名为 Transactions 的表中。除非 Transactions 表中的记录少于 10 条，否则脚本将被终止。

如果此条件未导致脚本终止，则会将另一组事务串联到 Transactions 表中，并重复此过程。

加载脚本

```
Transactions:
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];


exit script unless NoOfRows('Transactions') < 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
8, 10/01/2018, 164.27
9, 10/03/2018, 384.00
10, 10/06/2018, 25.82
11, 10/09/2018, 312.00
12, 10/15/2018, 4.56
13, 10/16/2018, 90.24
14, 10/18/2018, 19.32
];

exit script unless NoOfRows('Transactions') < 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
15, 10/01/2018, 164.27
16, 10/03/2018, 384.00
17, 10/06/2018, 25.82
18, 10/09/2018, 312.00
19, 10/15/2018, 4.56
20, 10/16/2018, 90.24
21, 10/18/2018, 19.32
];
```

```
exit script unless NoOfRows('Transactions') < 10 ;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount

结果表

id	日期	金额
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42
8	10/01/2018	164.27
9	10/03/2018	384.00
10	10/06/2018	25.82
11	10/09/2018	312.00
12	10/15/2018	4.56
13	10/16/2018	90.24
14	10/18/2018	19.32

加载脚本的三个数据集中的每个数据集中都有七条记录。

第一个数据集(具有交易 id 1 到 7)被加载到应用程序中。Unless 条件评估 Transactions 表中的行是否少于 10 行。这计算为 TRUE，因此第二个数据集(具有交易 id 8 到 14)被加载到应用程序中。第二个 Unless 条件评估 Transactions 表中的记录是否少于 10 条。此计算的结果为 FALSE，因此脚本终止。

示例 3 – 多个 Unless 非前缀

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

在此示例中，包含一个交易的数据集被创建为名为 Transactions 的表。然后触发 'for' 循环，其中两个嵌套的 Unless 语句进行计算：

1. 除非 Transactions 表中有 100 多条记录
2. 除非 Transactions 表中的记录数是 6 的倍数

如果这些条件为 FALSE，将生成另外七条记录并将其串联到现有 Transactions 表上。重复此过程，直到两个交易之一返回值 TRUE。

加载脚本

```
Transactions:
Load
        0 as id
Autogenerate 1;

For i = 1 to 100
        unless NoOfRows('Transactions') > 100 unless mod(NoOfRows('Transactions'),6) = 0
                Concatenate
      Load
if(isnull(Peek(id)),1,peek(id)+1) as id
               Autogenerate 7;
next i
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：id。

结果表

id
0
1
2
3
4
5
+30 多行

出现在 'for' 循环中的嵌套 Unless 语句的计算结果如下:。

1. Transactions 表中有 100 行以上吗?
2. Transactions 表中的记录总数是 6 的倍数吗?

每当两个 Unless 语句都返回值 FALSE 时,将生成另外七条记录并将其串联到现有表 Transactions 上。

这些语句返回值 FALSE 五次,此时 Transactions 表中总共有 36 行数据。

在此之后,第二条 Unless 语句返回值 TRUE,因此此后的 Load 语句将不再执行。

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 )
```

返回数据类型: 布尔值

在 Qlik Sense 中,布尔 true 值由 -1 表示,false 值由 0 表示。

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。

适用场景

When 语句返回布尔值结果。通常,当用户希望加载或排除脚本的部分时,这种类型的函数将用作条件。

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。

区域设置

除非另有规定,本主题中的示例使用以下日期格式:MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素,系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者,您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
`exit script when A=1;`	当 `A=1` 语句被计算为 TRUE 时，脚本将停止。
`when A=1 LOAD * from myfile.csv;`	当 `A=1` 语句被计算为 TRUE 时，将加载 `myfile.csv`。
`when A=1 unless B=2 drop table Tab1;`	当语句 `A=1`的计算结果为 TRUE 时，如果 `B=2` 的计算结果为 FALSE，则 `Tab1` 表将被删除。

示例 1 – When 前缀

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 一个初始数据集，其中包含发送到名为 'Transactions' 的表的日期和金额。
- 声明变量 `A` 已创建且值为 `1` 的 `Let` 语句。
- 提供条件的 `When` 条件，所提供的条件为如果 `A` 等于 `1`，则将继续加载脚本。

加载脚本

```
LET A = 1;

WHEN A = 1

Transactions:
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount

结果表

id	date	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

因为变量 A 在脚本开始时被赋值 1，所以会计算前缀 When 后面的条件，并返回结果 TRUE。因为它返回 TRUE 结果，所以脚本继续运行 Load 语句。可以看到结果表中的所有记录。

如果此变量值设置为任何不等于 1 的值，则不会将任何数据加载到数据模型中。

示例 2 – When 后缀

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 三个数据集，其中包含发送到名为 'Transactions' 的表的日期和金额。
  - 第一个数据集包含交易 1-7。
  - 第而个数据集包含交易 8-14。
  - 第三个数据集包含交易 15-21。
- 确定 'Transactions' 表是否包含十行以上的 When 条件。如果任何 When 语句的计算结果为 TRUE，则加载脚本将停止。此条件放置在三个数据集的每一个的末尾。

加载脚本

```
Transactions:
LOAD
*
```

```
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];

exit script when NoOfRows('Transactions') > 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
8, 10/01/2018, 164.27
9, 10/03/2018, 384.00
10, 10/06/2018, 25.82
11, 10/09/2018, 312.00
12, 10/15/2018, 4.56
13, 10/16/2018, 90.24
14, 10/18/2018, 19.32
];

exit script when NoOfRows('Transactions') > 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
15, 10/01/2018, 164.27
16, 10/03/2018, 384.00
17, 10/06/2018, 25.82
18, 10/09/2018, 312.00
19, 10/15/2018, 4.56
20, 10/16/2018, 90.24
21, 10/18/2018, 19.32
];

exit script when NoOfRows('Transactions') > 10 ;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- amount

结果表

id	date	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42
8	10/01/2018	164.27
9	10/03/2018	384.00
10	10/06/2018	25.82
11	10/09/2018	312.00
12	10/15/2018	4.56
13	10/16/2018	90.24
14	10/18/2018	19.32

三个数据集中的每一个都有七个交易。第一个数据集包含交易 1-7 并加载到应用程序中。此 Load 语句后面的 When 条件计算为 FALSE，因为 'Transactions' 表中的行少于 10 行。加载脚本继续到下一个数据集。

第二个数据集包含交易 8-14 并加载到应用程序中。第二个 When 条件的计算结果为 TRUE，因为 'Transactions' 表中有 10 行以上。因此，脚本终止。

示例 3 – 多个 When 非前缀

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含单个交易的数据集被创建为名为 'Transactions' 的表。
- 触发的 For 循环包含两个嵌套 When 条件，用于评估：
  1. 'Transactions' 表中的记录是否少于 100 条。
  2. 'Transactions' 表中的记录数是否不是 6 的倍数。

加载脚本

```
RowsCheck = NoOfRows('Transactions') < 100 or mod(NoOfRows('Transactions'),6) <> 0;
Transactions:
Load
	0 as id
Autogenerate 1;
For i = 1 to 100
	when(RowsCheck)
		Concatenate
	Load
		if(isnull(Peek(id)),1,peek(id)+1) as id
	Autogenerate 7;
next i
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- id

结果表仅显示前五个交易 ID，但加载脚本创建 36 行，然后在满足 When 条件后终止。

结果表

id
0
1
2
3
4
5
+30 多行

For 循环中嵌套的 When 条件评估以下问题：

- Transactions 表中的行数是否不到 100?
- Transactions 表中的记录总数是否不是 6 的倍数?

每当两个 When 条件都返回值 TRUE 时，将生成另外七条记录并将其串联到现有的 'Transactions' 表上。

When 条件返回 TRUE 值五次。此时，'Transactions' 表中总共有 36 行数据。

在 'Transactions' 表中创建 36 行数据时，第二条 When 语句返回值 FALSE，因此将不再执行此后的 Load 语句。

3.3 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。表的 "distinct" 属性在 **drop field** 语句之后被删除。

> ***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、Parquet、CSV 或 TXT 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 547)* 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	**Using** 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
South, 2014,   June,   645
South, 2013,   May,    367
South, 2013,   May,    221
];

Budget:
LOAD Budget,
Region &'|'& 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

KeyToOtherTable	Region	Year	Month	Sales
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})
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:`

参数	说明
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 [fields]] 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。表的 "distinct" 属性在 **drop field** 语句之后被删除。

> ***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。

force 语句还可以在加载期间更改字段名称，或使用以下数据源进行选择：

- QVD
- CSV(文本文件)
- XLS
- QVX(文件和 ODBC 连接)

如果数据以紧凑模式加载(使用 * 加载)，则 **force** 语句仅更改字段名。

以下数据源的字段名不受 **force** 语句的影响：

- JSON
- Parquet
- 数据截断，
- XLSX

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。

示例	结果

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	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 Load，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*** • 相对于 Qlik Sense 应用程序工作目录。 **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***

参数	说明
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)。有关更多信息，请参阅*格式规格项目 (page 159)*。 ***from_field** 在分隔表中的字段时，仅支持将逗号作为列表分隔符。*
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** 子句输入的数据必须用特定字符括起来 - 方括号、引号或反引号。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：键入脚本时按压输入键。 在简单的内联加载中，列的数量由第一行定义。 *format-spec ::= (fspec-item {, fspec-item })* 您可以使用许多可用于其他已加载表的相同格式规范项来自定义内联加载。括号中列出了这些项目。有关更多信息，请参阅*格式规格项目 (page 159)*。 有关使用内联加载的详细信息，请参阅使用内联加载来加载数据。
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**扩展**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 Load)
从随后的 **SELECT** 语句中加载的 Table1 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 | parquet | 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 148)

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。
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。 **Table is** 说明符可用于定义要加载为表的工作表名称。 *Table is (page 164)*
html	如果表格是 html 页面或文件的一部分，则应使用 html。

类型	说明
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。 **Table is** 说明符可用于定义要加载为表的 XML 的路径。 *Table is (page 164)*
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。
parquet	Apache Parquet 是一种列式存储格式,可高效存储和查询大型数据集。 对于包含嵌套数据的 Parquet 文件,可以使用 **Table is** 说明符从要加载的 Parquet 文件中指定表。例如: `LOAD * FROM [lib://DataFiles/company.parquet] (parquet, table is [company:salesrep.salesrep]);`。 *Table is (page 164)*

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 148)

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 148)

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 148)

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 148)

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 148)

Table is

对于 Excel、XML 或 Parquet 文件，可以在表格式说明符中指定要从中加载数据的表。

语法：

```
Table is table name
```

参数：

参数

参数	描述

table name	指定表的名称。该值取决于表格式：
	• Excel：工作表名称。
	• XML：指定要加载的 XML 部分的路径。
	• Parquet：指定表的路径，格式为 *<node>.<node>.<node>*。
	在嵌套结构中指定表时使用 **Table is**。
	例如，您有以下模式中的 Parquet 数据：
	`Schema:`
	`Field(name: "Name", datatype: String),`
	`Field(name: "Age", datatype: Float),`
	`Field(name: "Phone", datatype: List(`
	`Field(name: "Item", datatype: Struct[`
	`Field(name: "Number", datatype: String)`
	您可以将 Phone 及其嵌套字段加载为带有参数 `Table is [Schema:Phone.Item]` 的表。这将与表一起生成关键字段 *%Key_Phone*。

示例：Excel

```
LOAD
       "Item Number",
       "Product Group",
       "Product Line",
       "Product Sub Group",
       "Product Type"
FROM [lib://AttachedFiles/Item master.xlsx]
(ooxml, embedded labels, table is [Item master]);
```

示例：数据截断，

```
LOAD
       city%Table,
       %Key_row_7FAC1F878EC01ECB
FROM [lib://AttachedFiles/cities.xml]
(XmlSimple, table is [root/row/country/city]);
```

示例：Parquet

文件 company.parquet 包含以下模式：

```
company (String)
contact (String)
company:salesrep (List)
       salesrep (Group)
              salesrep (String)
company:headquarter (List)
       headquarter (Group)
              country (String)
              city (String)
              city:region (List)
              region (Group)
                     region (String)
```

以下操作将把文件中的内容加载到表中。第一个 LOAD 语句加载根组。第二个 LOAD 语句将 *salesrep* 组的内容加载为一个表。第三个 LOAD 语句将*总部*组加载为一个表。第四个 LOAD 语句将*地区*组加载为表。

```
LOAD * FROM […] (parquet);
LOAD * FROM […] (parquet, table is [company:salesrep.salesrep]);
LOAD * FROM […] (parquet, table is [company:headquarter.headquarter]
LOAD * FROM […] (parquet, table is [company:headquarter.headquarter.city:region.region]
```

限制：

Table is 说明符仅对 Excel、XML 或 Parquet 文件相关。

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 148)

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 148)

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 148)

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。

参数	说明
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、Parquet、CSV 或 TXT 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、Parquet 或文本文件。

该语句仅会从一个数据表格中导出字段，除非您正存储到 Parquet。如果要从多个表格中导出字段为 QVD、CSV 或 TXT 文件，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。通过在 Parquet 文件中嵌套数据，可以将多个表存储在一个 Parquet 中。

文本值将在 UTF-8 中以 BOM 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。

在某些数据格式不正确的情况下，字段将被双引号包围，以确保数据得到正确解释。例如，当字段包含引号、逗号、空格或换行符等字符时，就会发生这种情况。

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*** • 相对于 Qlik Sense 应用程序工作目录。 **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** | **parquet**), *压缩为***编码解码器**)	您可以将格式规范设置为这些文件格式之一。如果省略格式规范，则假定为 **qvd**。 • CSV 和 TXT 文件的 **txt**。 • **qvd** 用于 QVD 文件。 • **parquet** 用于 Parquet 文件。 如果使用 **parquet**，还可以设置用于**压缩为**的压缩编解码器。如果不是用**压缩为**指定压缩编解码器，则使用 snappy。有以下压缩设置可用： • uncompressed • snappy • gzip • lz4 • brotli • zstd • lz4_hadoop 示例： `Store mytable into [lib://AttachedFiles/myfile.parquet] (parquet, compression is lz4);`

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 mytable into [lib://FolderConnection/myfile.csv] (txt);

Store mytable into myfile.parquet (parquet);

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

存储在 Parquet 文件中

Parquet 是一种强类型文件格式，其中每个字段都包含一种特定类型的数据（如 32、double、timestamp 或 text）。Qlik Sense 将内部数据存储为松散类型的 dual 数据，其中来自不同源的数据可以混合到相同的字段中。由于 Parquet 中的每个字段中只能存储 dual 数据的一部分，因此了解每个字段包含的内容很重要。默认情况下，Qlik Sense 使用字段类型来确定字段的存储方式。以特定格式将数据存储在 Parquet 文件中时，必须指定加载字段时的数据类型。如果您试图将数据存储到 Parquet 文件中不兼容的字段中，例如文本字段中的数字或时间戳字段中的文本，则最终会得到 null 值。

加载要存储在 Parquet 中的数据时，可以更改默认行为。您可以格式化它以更改数据类型，也可以标记它以强制执行 Parquet 中的特定列类型。

格式化数据以存储在 Parquet 中

您可以使用 Qlik Sense 格式化功能对数据进行分类。例如，**Text()**、**Num()**、**Interval()** 或 **Timestamp()** 可以在 Parquet 中存储数据时强制执行数据格式。Qlik Sense 可以根据字段属性和自动字段标签将数据存储为近 20 种数据类型。有关更多信息，请参阅*解释函数 (page 1211)*。

示例：使用 Num() 和 Text() 进行格式设定

以下示例演示了准备数据以存储在 Parquet 中。**Num()** 应用于 num 字段。**Text()** 同时应用于文本和混合字段。在混合字段的情况下，**Text()** 防止将其视为 Parquet 中的数字字段，并将文本值更改为 null 值。

```
Data:
LOAD * INLINE [
num, text, mixed
123.321, abc, 123
456.654, def, xyz
789.987, ghi, 321
];

Format:
NoConcatenate
LOAD num, text, Text(mixed) as mixed RESIDENT Data;
STORE Format INTO [lib://AttachedFiles/Tmp.parquet] (parquet);
```

标记数据以存储在 Parquet 中

在以 Parquet 中存储数据时，可以使用 $parquet 标记对数据进行标记，以强制使用特定的列类型。每种数据类型都可以通过添加相应的控制标签来强制执行。例如，要在 Parquet 中将字段存储为 INT32，请在加载脚本中将其标记为 $parquet-int32。根据数据类型，将存储字符串或 dual 数据的数字表示。

以下 Parqout 控制标记可用于标记存储在 Parquet 文件中的字段。

Parquet 控制标记

控制标记	双	物理类型	逻辑类型	转换的类型
$parquet-boolean	数字	BOOLEAN	NONE	NONE
$parquet-int32	数字	INT32	NONE	NONE
$parquet-int64	数字	INT64	NONE	NONE
$parquet-float	数字	FLOAT	NONE	NONE
$parquet-double	数字	DOUBLE	NONE	NONE
$parquet-bytearray	字符串	BYTE_ARRAY	NONE	UTF8
$parquet-bytearrayfix	数字	FIXED_LEN_BYTE_ARRAY	NONE	DECIMAL
$parquet-decimal	数字	INT64	DECIMAL	DECIMAL
$parquet-date	数字	INT32	DATE	DATE
$parquet-time	数字	INT64	TIME	TIME_MICROS
$parquet-timestamp	数字	INT64	TIMESTAMP	TIMESTAMP_MICROS
$parquet-string	字符串	BYTE_ARRAY	STRING	UTF8
$parquet-enum	字符串	BYTE_ARRAY	ENUM	ENUM
$parquet-interval	数字	FIXED_LEN_BYTE_ARRAY	INTERVAL	INTERVAL
$parquet-json	字符串	BYTE_ARRAY	JSON	JSON
$parquet-bson	字符串	BYTE_ARRAY	BSON	BSON
$parquet-uuid	字符串	FIXED_LEN_BYTE_ARRAY	UUID	NONE

示例：标记数据以存储在 Parquet 中

在本例中，使用了两个标记来定义 Parquet 的数据。该字段 *num* 被标记为 `$parquet-int32`，以将其定义为一个数字字段，该字段将在 Parquet 中设置为 INT32。

```
Data:
LOAD * INLINE [
num, text,
123.321, abc
456.654, def
789.987, ghi
];
TAG num WITH '$parquet-int32';
STORE Format INTO [lib://AttachedFiles/Tmp.parquet] (parquet);
```

将嵌套数据存储在 Parquet 文件中

通过将多个表嵌套到结构化数据中，可以将它们存储在一个 Parquet 文件中。**Store** 支持星形模式中的结构化节点和列表节点。也可以使用**分隔符为**说明符以嵌套模式存储单个表。

存储表时，请指定要包含的表，用逗号分隔。例如：`STORE Table1, Table2, Table3 INTO [lib://<file location>/<file name>.parquet] (parquet);`。您可以使用 **Store** 语句中的字段列表来控制存储哪些字段。例如 `STORE Field1, Field2, FROM Table1, Table2 INTO [lib://<file location>/<file name>.parquet] (parquet);`。字段列表中的所有字段必须位于一个或多个列出的表中。**Store** 语句中的第一个表将用作星形模式中的事实表。

字段名用于控制组的创建和嵌套方式。默认情况下，字段名称被拆分为带有句点 (.) 的节点。可以通过设置系统变量 *FieldNameDelimiter* 或使用说明符**分隔符为**来更改分隔符。说明符将覆盖系统变量。。

字段名由分隔符分隔，这些部分用于创建具有嵌套组的架构。例如，`STORE Field1, Field1.Field2, Field1.Field3, Field1.Field4 FROM Table1 INTO [nested.parquet] (parquet, delimiter is '.');` 将创建两个组（*Group1* 和 *Group2*），其中具有 *Fields1*、*Field2* 和 *Field3*、*Field4*。

> *在架构中的节点中，组和字段的名称可能不相同。例如，`STORE Address, Address.Street INTO [nested.parquet] (parquet, delimiter is '.');` 将失败，因为 Address 不明确，并且既是数据字段又是组。*

在 Parquet 中存储嵌套数据时，表之间的键将转换为模式中的链接节点。表被转换为模式中的结构化节点。可以使用字段名称覆盖默认转换。

示例：将嵌套数据存储在 Parquet 文件中

```
company:
LOAD * INLINE [
company, contact
A&G, Amanda Honda
Cabro, Cary Frank
Fenwick, Dennis Fisher
Camros, Molly McKenzie
];
```

```
salesrep:
LOAD * INLINE [
company, salesrep
A&G, Bob Park
Cabro, Cezar Sandu
Fenwick, Ken Roberts
Camros, Max Smith
];

headquarter:
LOAD * INLINE [
company, country, city
A&G, USA, Los Angeles
Cabro, USA, Albuquerque
Fenwick, USA, Baltimore
Camros, USA, Omaha
];

region:
LOAD * INLINE [
region, city
West, Los Angeles
Southwest, Albuquerque
East, Baltimore
Central, Omaha
];

STORE company, salesrep, headquarter, region INTO [lib://AttachedFiles/company.parquet]
(parquet)
DROP TABLES company, salesrep, headquarter, region;
```

生成的 Parquet 文件具有以下架构：

```
company (String)
contact (String)
company:salesrep (List)
       salesrep (Group)
               salesrep (String)
company:headquarter (List)
       headquarter (Group)
               country (String)
               city (String)
               city:region (List)
                       region (Group)
                              region (String)
```

限制

在 Parquet 中存储嵌套数据有以下限制：

- 存储不支持映射节点。
- 存储不包括加载嵌套 parquet 文件而生成的关键字段。
- 不能将未与键字段链接的表中的数据存储在一起。
- 嵌套文件将取消数据模型的规范化。未引用的值将不会被保存，多次引用的值也将被复制。

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。

参数	说明
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;
```

3.4 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。

4 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。

4.1 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。

4.2 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"。
```

为变量命名

作为最佳实践，可以考虑对在应用程序中创建的变量使用标准化的命名约定。例如，您可以确保所有变量名都以 *v* 开头。例如：*vUserText*。这有助于确保变量被快速识别为变量，并与度量、字段和函数区分开来。

4.3 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。

4.4 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
```

4.5 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 263)*
- *数字解释变量 (page 203)*
- *系统变量 (page 195)*
- *值处理变量 (page 201)*

4.6 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。

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
```

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。

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
```

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)
```

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
```

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
```

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。

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。

新应用程序在默认的 SET 语句中包含 set CreateSearchIndexOnReload=1。在 Analytics 活动中心创建脚本时，SET 表达式不包含 CreateSearchIndexOnReload，因为脚本资产未持久化数据，因此索引值并非必需。

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。

在Qlik Sense和Qlik Sense Desktop中，该变量仅支持对文件夹数据连接的引用。不支持引用基于云的存储提供者中的文件。

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。请勿在等于号前后输入空格字符。

*结构 **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;
```

4.7 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'。

4.8 数字解释变量

数字解释变量为系统定义的。变量包含在加载脚本的顶部，并在脚本执行时应用数字格式设置。您可以删除、编辑或复制这些变量。

数字解释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，此项依据计算机操作系统的设置。在 Qlik Sense 中，它依据安装 Qlik Sense 的服务器的操作系统。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

货币格式

MoneyDecimalSep

定义的货币小数分隔符将替换由区域设置设定的小数位符号。

```
MoneyDecimalSep
```

MoneyFormat

定义的符号将替换由地区设置设定的货币符号。

```
MoneyFormat
```

MoneyThousandSep

定义的千位分隔符会替换由地区设置设定的货币数字分组符号。

```
MoneyThousandSep
```

数字格式

DecimalSep

定义的小数分隔符将替换由区域设置设定的小数位符号。

```
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

定义的格式会替换地区设置的月名称惯例。

```
MonthNames
```

LongMonthNames

定义的格式会替换地区设置的长版本月名称惯例。

```
LongMonthNames
```

DayNames

定义的格式会替换由地区设置设定的普通日名称惯例。

```
DayNames
```

LongDayNames

定义的格式会替换地区设置的长版本普通日名称惯例。

```
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
```

在 Qlik Sense 中，创建应用程序时获取区域设置，相应的设置作为环境变量存储在脚本中。

北美的应用程序开发人员经常在脚本中加入 `Set BrokenWeeks=1;`，对应于中断周。欧洲的应用程序开发人员经常在脚本中加入 `Set BrokenWeeks=0;`，对应于非中断周。

非中断周意味着：

- 在某些年份，第 1 周从 12 月开始，而在其他年份，前一年的最后一周持续到一月。
- 根据 ISO 8601，第 1 周在 1 月始终至少有 4 天。在 Qlik Sense 中，可以使用 `ReferenceDay` 变量对此进行配置。

中断周意味着：

- 一年中的最后一周永远不会持续到一月。
- 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)

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例：

如果您想要周和周数的 ISO 设置，请确保脚本中包含以下内容：

```
Set FirstWeekDay=0;
Set BrokenWeeks=0;   //(use unbroken weeks)
Set ReferenceDay=4;
```

如果需要 US 设置，请确保脚本中包含以下内容：

```
Set FirstWeekDay=6;
Set BrokenWeeks=1;   //(use broken weeks)
Set ReferenceDay=1;
```

DateFormat

此环境变量定义了应用程序中默认使用的日期格式以及按日期返回函数，如 date() 和 date#()。该格式用于解释和格式化日期。如果未定义变量，则在脚本运行时获取由区域设置设定的日期格式。

语法：

DateFormat

DateFormat 函数示例

示例	结果
`Set DateFormat='M/D/YY'; //(US format)`	DateFormat 函数的使用将日期定义为美国格式，月/日/年。
`Set DateFormat='DD/MM/YY'; //(UK date format)`	DateFormat 函数的使用将日期定义为英国格式，日/月/年。
`Set DateFormat='YYYY/MM/DD'; //(ISO date format)`	DateFormat 函数的使用将日期定义为 ISO 格式，年/月/日。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

1– 系统变量默认值

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 日期数据集。
- DateFormat 函数，将使用美国日期格式。

在本例中，将数据集加载到名为 'Transactions' 的表中。它包括一个 date 字段。使用美国 DateFormat 定义。加载文本日期时，此模式将用于隐式文本到日期的转换。

加载脚本

```
Set DateFormat='MM/DD/YYYY';

Transactions:
LOAD
date,
month(date) as month,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- month

创建该度量：

```
=sum(amount)
```

结果表

日期	月	=sum(amount)
01/01/2022	一月	1000

日期	月	=sum(amount)
02/01/2022	二月	2123
03/01/2022	三月	4124
04/01/2022	四月	2431

DateFormat 定义 MM/DD/YYYY 用于将文本隐式转换为日期，这就是为什么 date 字段被正确解释为日期的原因。相同的格式用于显示日期，如结果表所示。

示例 2 – 更改系统变量

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 上一 中的相同数据集。
- DateFormat 函数将使用“DD/MM/YYYY”格式。

加载脚本

```
SET DateFormat='DD/MM/YYYY';
Transactions:
LOAD
date,
month(date) as month,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- month

创建该度量：

```
=sum(amount)
```

结果表

日期	月	=sum(amount)
01/01/2022	一月	1000
02/01/2022	一月	2123
03/01/2022	一月	4124
04/01/2022	一月	2431

由于 DateFormat 定义被设置为 “DD/MM/YYYY”, 您可以看到, 第一个“/”符号后的两位数字被解释为月份, 因此所有记录都来自一月。

示例 3 – 日期解释

加载脚本和结果

概述

打开数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 以数字格式显示日期的数据集。
- DateFormat 变量将使用“DD/MM/YYYY”格式。
- date() 变量,

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
date(numerical_date),
month(date(numerical_date)) as month,
id,
amount
Inline
[
numerical_date,id,amount
43254,1,1000
43255,2,2123
43256,3,4124
43258,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- date
- month

创建该度量：

```
=sum(amount)
```

结果表

日期	月	=sum(amount)
06/03/2022	六月	1000
06/04/2022	六月	2123
06/05/2022	六月	4124
06/07/2022	六月	2431

在加载脚本中，使用 date() 函数将数字日期转换为日期格式。因为您没有在函数中提供指定的格式作为第二个参数，所以使用了 DateFormat 格式。这将导致日期字段使用的格式为“MM/DD/YYYY”。

示例 4 – 外部日期格式

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 日期数据集。
- DateFormat 变量，其使用 'DD/MM/YYYY' 格式，但未使用正斜杠注释。

加载脚本

```
// SET DateFormat='DD/MM/YYYY';


Transactions:
Load
date,
month(date) as month,
id,
amount
Inline
[
date,id,amount
22-05-2022,1,1000
23-05-2022,2,2123
24-05-2022,3,4124
25-05-2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- month

创建该度量：

```
=sum(amount)
```

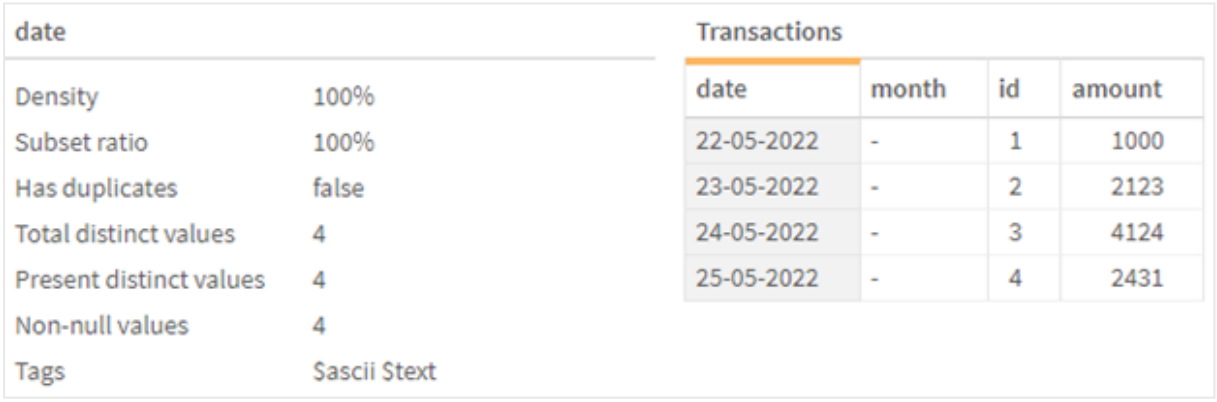
结果表

日期	月	=sum(amount)
22-05-2022	-	1000
23-05-2022	-	2123
24-05-2022	-	4124
25-05-2022	-	2431

在初始加载脚本中，DateFormat 使用的是默认的“MM/DD/YYYY”。由于交易数据集中的 date 字段不是此格式，因此该字段不会解释为日期。这显示在结果表中，其中 month 字段值为空。

通过检查 date 字段的“标记”属性，可以在数据模型查看器中验证已解释的数据类型：

Transactions 表格的预览。请注意，date 字段的“标记”表示文本输入数据尚未隐式转换为日期/时间戳。



这可以通过启用 DateFormat 系统变量来解决：

```
// SET DateFormat='DD/MM/YYYY';
```

删除双正斜杠并重新加载数据。

Transactions 表格的预览。请注意，date 字段的“标记”表示文本输入数据已隐式转换为日期/时间戳。

date	
Density	100%
Subset ratio	100%
Has duplicates	false
Total distinct values	4
Present distinct values	4
Non-null values	4
Tags	$numeric $integer $timestamp $date

Transactions

date	month	id	amount
22-05-2022	May	1	1000
23-05-2022	May	2	2123
24-05-2022	May	3	4124
25-05-2022	May	4	2431

DayNames

定义的格式会替换由地区设置设定的普通日名称惯例。

语法：

DayNames

修改变量时，需要使用分号 ； 分隔各个值。

DayName 函数示例

函数	结果定义
`Set DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';`	使用函数以缩写形式定义 DayNames。
`Set DayNames='M;Tu;W;Th;F;Sa;Su';`	使用 DayNames 函数可以根据日期的首字母定义日期。

DayNames 函数通常与以下功能结合使用：

相关函数

函数	交互
weekday (page 1020)	脚本函数返回 DayNames 作为字段值。
Date (page 1180)	脚本函数返回 DayNames 作为字段值。
LongDayNames (page 222)	长形 DayNames 值。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1– 系统变量默认值

加载脚本和结果

概述

在本例中，数据集中的日期以 MM/DD/YYYY 格式设置。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 带有日期的数据集，将加载到名为 Transactions 的表中。
- date 字段。
- 默认 DayNames 定义。

加载脚本

```
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';

Transactions:
LOAD
date,
WeekDay(date) as dayname,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- dayname

创建该度量：

```
sum(amount)
```

结果表

日期	dayname	sum(amount)
01/01/2022	Sat	1000
02/01/2022	Tue	2123
03/01/2022	Tue	4124
04/01/2022	Fri	2431

在加载脚本中，WeekDay 函数与 date 字段一起用作提供的参数。在结果表中，此 WeekDay 函数的输出以 DayNames 定义的格式显示一周中的天数。

示例 2 – 更改系统变量

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。使用与第一个示例相同的数据集和场景。

但是，在脚本开始时，DayNames 定义被修改为使用南非荷兰语中一周中的缩写天数。

加载脚本

```
SET DayNames='Ma;Di;Wo;Do;Vr;Sa;So';

Transactions:
Load
date,
WeekDay(date) as dayname,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- dayname

创建该度量：

```
sum(amount)
```

结果表

日期	dayname	sum(amount)
01/01/2022	Sa	1000
02/01/2022	Di	2123
03/01/2022	Di	4124
04/01/2022	Vr	2431

在结果表中，此 WeekDay 函数的输出以 DayNames 定义的格式显示一周中的天数。

重要的是要记住，如果对 DayNames 的语言进行了修改，如本例中所示，则 LongDayNames 仍将包含英文的星期几。如果应用程序中使用了两个变量，则也需要修改。

示例 3 – 日期函数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 带有日期的数据集，将加载到名为 Transactions 的表中。
- date 字段。
- 默认 DayNames 定义。

加载脚本

```
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';

Transactions:
Load
date,
Date(date,'WWW') as dayname,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- dayname

创建该度量：

```
sum(amount)
```

结果表

日期	dayname	sum(amount)
01/01/2022	Sat	1000
02/01/2022	Tue	2123
03/01/2022	Tue	4124
04/01/2022	Fri	2431

使用默认 DayNames 定义。在加载脚本中，Date 函数与 date 字段一起用作第一个参数。第二个参数为 WWW。此格式将结果转换为存储在 DayNames 定义中的值。这将显示在结果表的输出中。

DecimalSep

定义的小数分隔符将替换由区域设置设定的小数位符号。

每当遇到可识别的数字模式时，Qlik Sense 自动将文本解释为数字。ThousandSep 和 DecimalSep 系统变量确定将文本解析为数字时应用的模式的组成。ThousandSep 和 DecimalSep 变量设置在前端图表和表格中可视化数字内容时的默认数字格式样式。也就是说，它直接影响任何前端表达式的**数字格式**选项。

假设千位分隔符为逗号‘,’和小数位分隔符为‘.’，以下是将隐式转换为数值等效值的模式示例：

```
0,000.00
```

```
0000.00
```

```
0,000
```

这些是作为文本保持不变的模式示例；即，不转换为数字：

```
0.000,00
```

```
0,00
```

语法：

DecimalSep

函数示例

示例	结果
`Set DecimalSep='.';`	设置“.”作为小数位分隔符。
`Set DecimalSep=',';`	将“,”设置为小数位分隔符。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 – 设置数字分隔符变量对不同输入数据的影响

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 以不同格式样式设置的总和和日期的数据集。
- 名为 Transactions 的表。
- DecimalSep 变量，设置为 '.'。
- ThousandSep 变量，设置为 ','。
- delimiter 变量，设置为"|"字符以分隔行中的不同字段。

加载脚本

```
Set ThousandSep=',';
Set DecimalSep='.';

Transactions:
Load date,
id,
amount as amount
Inline
[
date|id|amount
01/01/2022|1|1.000-45
01/02/2022|2|23.344
01/03/2022|3|4124,35
01/04/2022|4|2431.36
01/05/2022|5|4,787
01/06/2022|6|2431.84
01/07/2022|7|4132.5246
01/08/2022|8|3554.284
01/09/2022|9|3.756,178
```

```
01/10/2022|10|3,454.356
] (delimiter is '|');
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度 amount。

创建该度量：

```
=sum(amount)
```

结果表

金额	=Sum(amount)
总计	20814.7086
1.000-45	
3.756,178	
4124,35	
23.344	23.344
2431.36	2431.36
2431.84	2431.84
3,454.356	3454.356
3554.284	3554.284
4132.5246	4132.5246
4,787	4787

任何未解释为数字的值都将保留为文本，并在默认情况下向左对齐。任何成功转换的值都向右对齐，保留原始输入格式。

表达式列显示等效的数值，默认情况下，其格式仅使用小数位分隔符 '.'。这可以使用表达式配置中的**数字格式**下拉设置覆盖。

FirstWeekDay

整数用于定义将哪一天用作一周的第一天。

语法：

FirstWeekDay

根据国际日期和时间表示标准 ISO 8601，星期一是一周的第一天。在许多国家，星期一也被用作一周的第一天，例如英国、法国、德国和瑞典。

但在其他国家，如美国和加拿大，星期日被认为是一周的开始。

在 Qlik Sense 中，创建应用程序时获取区域设置，相应的设置作为环境变量存储在脚本中。

北美的应用程序开发人员经常在脚本中加入 `Set FirstWeekDay=6;`，对应于星期日。欧洲的应用程序开发人员经常在脚本中加入 `Set FirstWeekDay=0;`，对应于星期一。

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

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例：

如果您想要周和周数的 ISO 设置，请确保脚本中包含以下内容：

```
Set FirstWeekDay=0;  // Monday as first week day
Set BrokenWeeks=0;
Set ReferenceDay=4;
```

如果需要 US 设置，请确保脚本中包含以下内容：

```
Set FirstWeekDay=6;  // Sunday as first week day
Set BrokenWeeks=1;
Set ReferenceDay=1;
```

示例 1 – 使用默认值(脚本)

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

在此示例中，加载脚本使用默认 Qlik Sense 系统变量值 FirstWeekDay=6。该数据包含 2020 年第一个14 天的数据。

加载脚本

```
// Example 1: Load Script using the default value of FirstWeekDay=6, i.e. Sunday

SET FirstWeekDay = 6;

Sales:
LOAD
        date,
        sales,
        week(date) as week,
        weekday(date) as weekday
Inline [
date,sales
01/01/2021,6000
01/02/2021,3000
01/03/2021,6000
01/04/2021,8000
01/05/2021,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
01/12/2020,7000
01/13/2020,7000
01/14/2020,7000
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- week
- weekday

结果表

日期	周	weekday
01/01/2021	1	Wed
01/02/2021	1	Thu
01/03/2021	1	Fri
01/04/2021	1	Sat

日期	周	weekday
01/05/2021	2	Sun
01/06/2020	2	Mon
01/07/2020	2	Tue
01/08/2020	2	Wed
01/09/2020	2	Thu
01/10/2020	2	Fri
01/11/2020	2	Sat
01/12/2020	3	Sun
01/13/2020	3	Mon
01/14/2020	3	Tue

由于使用默认设置，FirstWeekDay 系统变量设置为 6。在结果表中，可以看到每个新的一周从星期日(1月 5 日和12日)开始。

示例 2–更改 FirstWeekDay 变量(脚本)

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

在本例中，数据包含 2020 年第一个 14 天的数据。在脚本开始时，我们将变量 FirstWeekDay 设置为 3。

加载脚本

```
// Example 2: Load Script setting the value of FirstWeekDay=3, i.e. Thursday

SET FirstWeekDay = 3;

Sales:
LOAD
       date,
       sales,
       week(date) as week,
       weekday(date) as weekday
Inline [
date,sales
01/01/2021,6000
01/02/2021,3000
01/03/2021,6000
01/04/2021,8000
01/05/2021,5000
```

```
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
01/12/2020,7000
01/13/2020,7000
01/14/2020,7000
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `week`
- `weekday`

结果表

日期	周	weekday
01/01/2021	52	Wed
01/02/2021	1	Thu
01/03/2021	1	Fri
01/04/2021	1	Sat
01/05/2021	1	Sun
01/06/2020	1	Mon
01/07/2020	1	Tue
01/08/2020	1	Wed
01/09/2020	2	Thu
01/10/2020	2	Fri
01/11/2020	2	Sat
01/12/2020	2	Sun
01/13/2020	2	Mon
01/14/2020	2	Tue

由于 `FirstWeekDay` 系统变量设置为 3，因此每周的第一天将是星期四。在结果表中，可以看到新的一周从星期四(1月 2日和 9日)开始。

LongDayNames

定义的格式会替换地区设置的长版本普通日名称惯例。

语法：

LongDayNames

以下 LongDayNames 函数 完整定义了日名称：

```
Set LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
```

修改变量时，需要使用分号 ; 分隔各个值。

LongDayNames 函数可以与将 DayNames 作为字段值返回的 *Date (page 1180)* 函数结合使用。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 系统变量默认值

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 带有日期的数据集，将加载到名为 Transactions 的表中。
- date 字段。
- 默认 LongDayNames 定义。

加载脚本

```
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';

Transactions:
LOAD
date,
Date(date,'WWWW') as dayname,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
```

```
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- dayname

创建该度量：

```
=sum(amount)
```

结果表

日期	dayname	=sum(amount)
01/01/2022	星期六	1000
02/01/2022	星期二	2123
03/01/2022	星期二	4124
04/01/2022	星期五	2431

在加载脚本中，要创建一个名为 dayname 的字段，Date 函数将与作为第一个参数的 date 字段结合使用。函数中的第二个参数是格式化 WWWW。

使用此格式将第一个参数的值转换为变量 LongDayNames 中设置的相应全天名称。在结果表中，我们创建的字段 dayname 的字段值显示此项。

示例 2 – 更改系统变量

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

使用与第一个示例相同的数据集和场景。但是，在脚本开始时，LongDayNames 定义被修改为使用西班牙语中一周中的天数。

加载脚本

```
SET LongDayNames='Lunes;Martes;Miércoles;Jueves;Viernes;Sábado;Domingo';

Transactions:
LOAD
date,
Date(date,'WWWW') as dayname,
id,
amount
INLINE
[
```

```
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- dayname

创建该度量：

```
=sum(amount)
```

结果表

日期	dayname	=sum(amount)
01/01/2022	Sábado	1000
02/01/2022	Martes	2123
03/01/2022	Martes	4124
04/01/2022	Viernes	2431

在加载脚本中，LongDayNames 变量被修改，从而以西班牙语列出一周中的天数。

然后，创建一个名为 dayname 的字段，其为要结合作为第一个参数的 date 字段使用的 Date 函数。

函数中的第二个参数是格式化 wwww。使用此格式 Qlik Sense 将第一个参数的值转换为变量 LongDayNames 中设置的相应全天名称。

在结果表中，我们创建的字段 dayname 的字段值以西班牙语和完整形式显示一周中的天数。

LongMonthNames

定义的格式会替换地区设置的长版本月名称惯例。

语法：

LongMonthNames

修改变量时，需要使用 ; 分隔各个值。

LongMonthNames 函数的以下示例完整定义了月份名称：

```
Set
LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;D
ecember';
```

LongMonthNames 函数通常与以下功能结合使用：

相关函数

函数	交互
Date (page 1180)	脚本函数返回 DayNames 作为字段值。
LongDayNames (page 222)	长形 DayNames 值。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 系统变量默认值

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的日期数据集。
- date 字段。
- 默认 LongMonthNames 定义。

加载脚本

```
SET
LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;D
ecember';

Transactions:
Load
date,
Date(date,'MMMM') as monthname,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,1000.45
01/02/2022,2,2123.34
01/03/2022,3,4124.35
```

```
01/04/2022,4,2431.36
01/05/2022,5,4787.78
01/06/2022,6,2431.84
01/07/2022,7,2854.83
01/08/2022,8,3554.28
01/09/2022,9,3756.17
01/10/2022,10,3454.35
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度。

- date
- monthname

创建该度量

```
=sum(amount)
```

结果表

日期	monthname	sum(amount)
01/01/2022	一月	1000.45
01/02/2022	一月	2123.34
01/03/2022	一月	4124.35
01/04/2022	一月	2431.36
01/05/2022	一月	4787.78
01/06/2022	一月	2431.84
01/07/2022	一月	2854.83
01/08/2022	一月	3554.28
01/09/2022	一月	3756.17
01/10/2022	一月	3454.35

使用了默认 LongMonthNames 定义。在加载脚本中，要创建一个名为 month 的字段，Date 函数将与作为第一个参数的 date 字段结合使用。函数中的第二个参数是格式化 MMMM。

使用此格式 Qlik Sense 将第一个参数中的值转换为变量 LongMonthNames 中设置的相应整月名称。在结果表中，我们创建的字段 month 的字段值显示此项。

示例 2 – 更改系统变量

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的日期数据集。
- date 字段。
- 修改后的 LongMonthNames 变量，以使用西班牙语中一周中的天数缩写。

加载脚本

```
SET
LongMonthNames='Enero;Febrero;Marzo;Abril;Mayo;Junio;Julio;Agosto;Septiembre;OctubreNoviembre;
Diciembre';

Transactions:
LOAD
date,
Date(date,'MMMM') as monthname,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将 sum(amount) 添加为度量，将这些字段添加为维度：

- date
- monthname

创建该度量：

```
=sum(amount)
```

结果表

日期	monthname	sum(amount)
01/01/2022	Enero	1000.45
01/02/2022	Enero	2123.34
01/03/2022	Enero	4124.35
01/04/2022	Enero	2431.36
01/05/2022	Enero	4787.78
01/06/2022	Enero	2431.84
01/07/2022	Enero	2854.83

日期	monthname	sum(amount)
01/08/2022	Enero	3554.28
01/09/2022	Enero	3756.17
01/10/2022	Enero	3454.35

在加载脚本中，LongMonthNames 变量被修改，从而以西班牙语列出月份。然后，要创建一个名为 monthname 的字段，其为要结合作为第一个参数的 date 字段使用的 Date 函数。函数中的第二个参数是格式化 MMMM。

使用此格式 Qlik Sense 将第一个参数中的值转换为变量 LongMonthNames 中设置的相应整月名称。在结果表中，我们创建的字段 monthname 的字段值显示以西班牙语写下的月份名称。

MoneyDecimalSep

定义的货币小数分隔符将替换由区域设置设定的小数位符号。

默认情况下，Qlik Sense 在表格图表中以不同的方式显示数字和文本。数字右对齐，文本左对齐。这使得查找文本到数字的转换问题变得很容易。显示 Qlik Sense 结果的此页面上，所有表都将使用此格式。

语法：

MoneyDecimalSep

Qlik Sense 应用程序将符合此格式的文本字段解释为货币值。文本字段必须包含 MoneyFormat 系统变量中定义的货币符号。MoneyDecimalSep 在处理从多个不同区域设置接收的数据源时特别有用。

以下示例显示了 MoneyDecimalSep 系统变量的可能用法：

```
Set MoneyDecimalSep='.';
```

此函数通常与以下函数一起使用：

相关函数

函数	交互
MoneyFormat	在文本字段解释的情况下，MoneyFormat 符号将用作解释的一部分。对于数字格式，图表对象中的 Qlik Sense 将使用 MoneyFormat 格式设定。
MoneyThousandSep	在文本字段解释的情况下，还必须遵守 MoneyThousandSep 函数。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 - MoneyDecimalSep dot (.) 表示法

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的数据集。
- 提供了数据，其货币字段采用以点 '.' 用作小数分隔符的文本格式。每个记录都以 '$' 符号作为前缀，但最后一条记录除外，该记录以 '£' 符号为前缀。

请记住，MoneyFormat 系统变量将美元 '$' 定义为默认货币。

加载脚本

```
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='$###0.00;-$###0.00';

Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$14.41'
01/02/2022,2,'$2,814.32'
01/03/2022,3,'$249.36'
01/04/2022,4,'$24.37'
01/05/2022,5,'$7.54'
01/06/2022,6,'$243.63'
01/07/2022,7,'$545.36'
01/08/2022,8,'$3.55'
01/09/2022,9,'$3.436'
01/10/2022,10,'£345.66'
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：amount。

添加以下度量：

- isNum(amount)
- sum(amount)

查看下面的结果，仅证明对所有美元 '$' 值的正确解释。

结果表

金额	=isNum(amount)	=Sum(amount)
总计	0	$3905.98
£345.66	0	$0.00
$3.436	-1	$3.44
$3.55	-1	$3.55
$7.54	-1	$7.54
$14.41	-1	$14.41
$24.37	-1	$24.37
243.63	-1	$243.63
$249.36	-1	$249.36
$545.36	-1	$545.36
$2,814.32	-1	$2814.32

上面的结果表显示了如何正确解释所有美元 ($) 前缀值的 amount 字段，而英镑 (£) 前缀的 amount 尚未转换为货币值。

示例 2 - MoneyDecimalSep 逗号 (,) 表示法

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的数据集。
- 提供了数据，其货币字段采用以逗号 ',' 用作小数分隔符的文本格式。除了最后一条记录错误地使用了小数点分隔符 '.' 之外，每条记录都以 '$' 符号作为前缀。

请记住，MoneyFormat 系统变量将美元 '$' 定义为默认货币。

加载脚本

```
SET MoneyThousandSep='.';
SET MoneyDecimalSep=',';
SET MoneyFormat='$###0.00;-$###0.00';
```

```
Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$14,41'
01/02/2022,2,'$2.814,32'
01/03/2022,3,'$249,36'
01/04/2022,4,'$24,37'
01/05/2022,5,'$7,54'
01/06/2022,6,'$243,63'
01/07/2022,7,'$545,36'
01/08/2022,8,'$3,55'
01/09/2022,9,'$3,436'
01/10/2022,10,'$345.66'
];
```

结果

结果的段落文本。

加载数据并打开工作表。创建新表并将该字段添加为维度：amount。

添加以下度量：

- isNum(amount)
- sum(amount)

查看下面的结果，演示所有值的正确解释，除了使用点'.'符号作为小数点分隔符的金额。在这种情况下，应该使用逗号。

结果表

金额	=isNum(amount)	=Sum(amount)
总计	0	$3905.98
$345.66	0	$0.00
$3,436	-1	$3.44
$3,55	-1	$3.55
$7,54	-1	$7.54
$14,41	-1	$14.41
$24,37	-1	$24.37
$243,63	-1	$243.63
$249,36	-1	$249.36

金额	=isNum(amount)	=Sum(amount)
$545,36	-1	$545.36
$2.814,32	-1	$2814.32

MoneyFormat

此系统变量定义 Qlik 用于将文本自动转换为数字的格式模式，其中数字以货币符号为前缀。它还定义了数字格式属性设置为“货币”的度量将如何显示在图表对象中。

MoneyFormat 系统变量中定义为格式模式一部分的符号将替换由区域设置设定的货币符号。

默认情况下，Qlik Sense 在表格图表中以不同的方式显示数字和文本。数字右对齐，文本左对齐。这使得查找文本到数字的转换问题变得很容易。显示 Qlik Sense 结果的此页面上，所有表都将使用此格式。

语法：

MoneyFormat

```
Set MoneyFormat='$ #,##0.00; ($ #,##0.00)';
```

当数值字段的 Number Formatting 属性设置为 Money 时，此格式设定将显示在图表对象中。此外，当 Qlik Sense 解释数字文本字段时，如果文本字段的货币符号与 MoneyFormat 变量中定义的符号的货币符号匹配，则 Qlik Sense 将此字段解释为货币值。

此函数通常与以下函数一起使用：

相关函数

函数	交互
MoneyDecimalSep (page 229)	对于数字格式设定，MoneyDecimalSep 将用于对象的字段格式设定。
MoneyThousandSep (page 237)	对于数字格式设定，MoneyThousandSep 将用于对象的字段格式设定。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 - MoneyFormat

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含一个数据集，该数据集加载到名为 Transactions 的表中。使用了默认 MoneyFormat 变量定义。

加载脚本

```
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='$###0.00;-$###0.00';

Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,$10000000441
01/02/2022,2,$21237492432
01/03/2022,3,$249475336
01/04/2022,4,$24313369837
01/05/2022,5,$7873578754
01/06/2022,6,$24313884663
01/07/2022,7,$545883436
01/08/2022,8,$35545828255
01/09/2022,9,$37565817436
01/10/2022,10,$3454343566
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- amount

添加该度量：

```
=Sum(amount)
```

在**数字格式设定**下，选择**货币**以配置 Sum(amount) 为货币值。

结果表

日期	金额	=Sum(amount)
总计		$165099674156.00
01/01/2022	$10000000441	$10000000441.00
01/02/2022	$21237492432	$21237492432.00
01/03/2022	$249475336	$249475336.00
01/04/2022	$24313369837	$24313369837.00
01/05/2022	$7873578754	$7873578754.00
01/06/2022	$24313884663	$24313884663.00
01/07/2022	$545883436	$545883436.00
01/08/2022	$35545828255	$35545828255.00
01/09/2022	$37565817436	$37565817436.00
01/10/2022	$3454343566	$3454343566.00

使用了默认 MoneyFormat 定义。此项的格式为：$###0.00;-$###0.00. 在结果表中，amount 字段的格式显示货币符号，小数点和小数位已包含在内。

示例 2 - 具有千位分隔符和混合输入格式的 MoneyFormat

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 一个混合输入格式的数据集，它被加载到一个名为 Transactions 的表中，其中散布着数千个分隔符和十进制分隔符。
- MoneyFormat 定义的修改被修改为包含逗号作为千位分隔符。
- 其中一行数据错误地在错误的位置用数千个逗号分隔。请注意，该金额是如何保留为文本而不能解释为数字的。

加载脚本

```
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat = '$#,##0.00;-$#,##0.00';

Transactions:
Load
date,
id,
```

```
amount
Inline
[
date,id,amount
01/01/2022,1,'$10,000,000,441.45'
01/02/2022,2,'$212,3749,24,32.23'
01/03/2022,3,$249475336.45
01/04/2022,4,$24,313,369,837
01/05/2022,5,$7873578754
01/06/2022,6,$24313884663
01/07/2022,7,$545883436
01/08/2022,8,$35545828255
01/09/2022,9,$37565817436
01/10/2022,10,$3454343566
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- amount

添加该度量：

```
=Sum(amount)
```

在**数字格式设定**下，选择**货币**以配置 Sum(amount) 为货币值。

结果表

日期	金额	=Sum(amount)
总计		$119,548,811,911.90
01/01/2022	$10,000,000,441.45	$10,000,000,441.45
01/02/2022	$212,3749,24,32.23	$0.00
01/03/2022	$249475336.45	$249,475,336.45
01/04/2022	$24	$24.00
01/05/2022	$7873578754	$7,873,578,754.00
01/06/2022	$24313884663	$24,313,884,663.00
01/07/2022	$545883436	$545,883,436.00
01/08/2022	$35545828255	$35,545,828,255.00
01/09/2022	$37565817436	$37,565,817,436.00
01/10/2022	$3454343566	$3,454,343,566.00

在脚本的开头，MoneyFormat 系统变量被修改为包含一个逗号作为千位分隔符。在 Qlik Sense 表格中，可以看到格式设定包括此分隔符。此外，带有错误分隔符的行未被正确解释，并保留为文本。这就是为什么它对总额没有贡献的原因。

MoneyThousandSep

定义的千位分隔符会替换由地区设置设定的货币数字分组符号。

默认情况下，Qlik Sense 在表格图表中以不同的方式显示数字和文本。数字右对齐，文本左对齐。这使得查找文本到数字的转换问题变得很容易。显示 Qlik Sense 结果的此页面上，所有表都将使用此格式。

语法：

MoneyThousandSep

Qlik Sense 应用程序将符合此格式的文本字段解释为货币值。文本字段必须包含 MoneyFormat 系统变量中定义的货币符号。MoneyThousandSep 在处理从多个不同区域设置接收的数据源时特别有用。

以下示例显示了 MoneyThousandSep 系统变量的可能用法：

```
Set MoneyDecimalSep=',';
```

此函数通常与以下函数一起使用：

相关函数

函数	交互
MoneyFormat	在文本字段解释的情况下，MoneyFormat 符号将用作解释的一部分。对于数字格式，图表对象中的 Qlik Sense 将使用 MoneyFormat 格式设定。
MoneyDecimalSep	在文本字段解释的情况下，还必须遵守 MoneyDecimalSep 函数。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 - MoneyThousandSep 逗号 (,) 表示法

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 加载到名为 Transactions 的表中的数据集。
- 提供了数据,其货币字段采用以逗号用作千位分隔符的文本格式。每条记录还以 '$' 符号为前缀。

请记住,MoneyFormat 系统变量将美元 '$' 定义为默认货币。

加载脚本

```
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='$###0.00;-$###0.00';

Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$10,000,000,441'
01/02/2022,2,'$21,237,492,432'
01/03/2022,3,'$249,475,336'
01/04/2022,4,'$24,313,369,837'
01/05/2022,5,'$7,873,578,754'
01/06/2022,6,'$24,313,884,663'
01/07/2022,7,'$545,883,436'
01/08/2022,8,'$35,545,828,255'
01/09/2022,9,'$37,565,817,436'
01/10/2022,10,'$3.454.343.566'
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度:amount。

添加以下度量:

- isNum(amount)
- sum(amount)

查看下面的结果。该表演示了使用逗号 ',' 符号作为千位分隔符的对所有值的正确解释。

除使用点 '.' 作为千位分隔符的一个值外,amount 字段已针对所有值得到正确解释。

结果表

金额	=isNum(amount)	=Sum(amount)
总计	0	$161645330590.00

金额	=isNum(amount)	=Sum(amount)
$3.454.343.566	0	$0.00
$249,475,336	-1	$249475336.00
$545,883,436	-1	$545883436.00
$7,873,578,754	-1	$7873578754.00
$10,000,000,441	-1	$10000000441.00
$21,237,492,432	-1	$21237492432.00
$24,313,369,837	-1	$24313369837.00
$24,33,884,663	-1	$24313884663.00
$35,545,828,255	-1	$35545828255.00
$37,565,817,436	-1	$37565817436.00

示例 2 - MoneyThousandSep 点 (.) 表示法

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的数据集。
- 提供了数据，其货币字段采用以点 '.' 用作千位分隔符的文本格式。每条记录还以 '$' 符号为前缀。

请记住，MoneyFormat 系统变量将美元 '$' 定义为默认货币。

加载脚本

```
SET MoneyThousandSep='.';
SET MoneyDecimalSep=',';
SET MoneyFormat='$###0.00;-$###0.00';

Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$10.000.000.441'
01/02/2022,2,'$21.237.492.432'
01/03/2022,3,'$249.475.336'
```

```
01/04/2022,4,'$24.313.369.837'
01/05/2022,5,'$7.873.578.754'
01/06/2022,6,'$24.313.884.663'
01/07/2022,7,'$545.883.436'
01/08/2022,8,'$35.545.828.255'
01/09/2022,9,'$37.565.817.436'
01/10/2022,10,'$3,454,343,566'
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：amount。

添加以下度量：

- isNum(amount)
- sum(amount)

查看下面的结果，用点 '.' 标记法作为千位分隔符来演示所有值的正确解释。

除使用逗号 ',' 作为千位分隔符的一个值外，amount 字段已针对所有值得到正确解释。

结果表

金额	=isNum(amount)	=Sum(amount)
总计	0	$161645330590.00
$3,545,343,566	0	$0.00
$249.475.336	-1	$249475336.00
$545.883.436	-1	545883436.00
$7.873.578.754	-1	$7873578754.00
$10.000.000.441	-1	$10000000441.00
$21.237.492.432	-1	$21237492432.00
$24.313.884.663	-1	$24313884663.00
$24.313.884.663	-1	$24313884663.00
$35.545.828.255	-1	$35545828255.00
$37.565.817.436	-1	$37565817436.00

MonthNames

定义的格式会替换地区设置的月名称惯例。

语法：

```
MonthNames
```

修改变量时，需要使用 ; 分隔各个值。

函数示例

示例	结果
`Set MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';`	此 MonthNames 函数的使用定义了英文月份名称及其缩写形式。
`Set MonthNames='Enero;Feb;Marzo;Abr;Mayo;Jun;Jul;Agosto;Set;Oct;Nov;Dic';`	此 MonthNames 函数的使用定义了西班牙语的月份名称及其缩写形式。

MonthNames 函数可与以下函数结合使用：

相关函数

函数	交互
month (page 868)	脚本函数返回 MonthNames 中定义的值作为字段值
Date (page 1180)	脚本函数，用于根据提供的格式化参数返回 MonthNames 中定义为字段值的值
LongMonthNames (page 225)	MonthNames 的长形值

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

1 – 系统变量默认值

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的日期数据集。
- date 字段。
- 默认 MonthNames 定义。

加载脚本

```
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
LOAD
date,
Month(date) as monthname,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000.45
01/02/2022,2,2123.34
01/03/2022,3,4124.35
01/04/2022,4,2431.36
01/05/2022,5,4787.78
01/06/2022,6,2431.84
01/07/2022,7,2854.83
01/08/2022,8,3554.28
01/09/2022,9,3756.17
01/10/2022,10,3454.35
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- monthname

创建该度量：

```
=sum(amount)
```

结果表

日期	monthname	sum(amount)
01/01/2022	一月	1000.45
01/02/2022	一月	2123.34
01/03/2022	一月	4124.35
01/04/2022	一月	2431.36
01/05/2022	一月	4787.78

日期	monthname	sum(amount)
01/06/2022	一月	2431.84
01/07/2022	一月	2854.83
01/08/2022	一月	3554.28
01/09/2022	一月	3756.17
01/10/2022	一月	3454.35

使用了默认 MonthNames 定义。在加载脚本中， Month 函数与 date 字段一起用作提供的参数。

在结果表中，此 Month 函数的输出以 MonthNames 定义的格式显示一年中的月份。

示例 2 – 更改系统变量

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的日期数据集。
- date 字段。
- 修改后的 MonthNames 变量，以使用西班牙语的缩写月份。

加载脚本

```
                              Set
MonthNames='Enero;Feb;Marzo;Abr;Mayo;Jun;Jul;Agosto;Set;Oct;Nov;Dic';

Transactions:
LOAD
date,
month(date) as month,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- monthname

创建该度量：

```
=sum(amount)
```

结果表

日期	monthname	sum(amount)
01/01/2022	Enero	1000.45
01/02/2022	Enero	2123.34
01/03/2022	Enero	4124.35
01/04/2022	Enero	2431.36
01/05/2022	Enero	4787.78
01/06/2022	Enero	2431.84
01/07/2022	Enero	2854.83
01/08/2022	Enero	3554.28
01/09/2022	Enero	3756.17
01/10/2022	Enero	3454.35

在加载脚本中，首先 MonthNames 变量被修改，从而以西班牙语缩写列出年中月份。Month 函数与 date 字段一起用作提供的参数。

在结果表中，此 Month 函数的输出以 MonthNames 定义的格式显示一年中的月份。

重要的是要记住，如果对 MonthNames 变量的语言进行了修改，如本例中所示，则 LongMonthNames 变量仍将包含英文的一年的月份。如果应用程序中同时使用两个变量，则必须修改 LongMonthNames 变量。

示例 3 – 日期函数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的日期数据集。
- date 字段。
- 默认 MonthNames 定义。

加载脚本

```
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
LOAD
date,
Month(date, 'MMM') as monthname,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000.45
01/02/2022,2,2123.34
01/03/2022,3,4124.35
01/04/2022,4,2431.36
01/05/2022,5,4787.78
01/06/2022,6,2431.84
01/07/2022,7,2854.83
01/08/2022,8,3554.28
01/09/2022,9,3756.17
01/10/2022,10,3454.35
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- monthname

创建该度量：

```
=sum(amount)
```

结果表

日期	monthname	sum(amount)
01/01/2022	一月	1000.45
01/02/2022	一月	2123.34
01/03/2022	一月	4124.35
01/04/2022	一月	2431.36
01/05/2022	一月	4787.78
01/06/2022	一月	2431.84
01/07/2022	一月	2854.83
01/08/2022	一月	3554.28

日期	monthname	sum(amount)
01/09/2022	一月	3756.17
01/10/2022	一月	3454.35

使用了默认 MonthNames 定义。在加载脚本中，Date 函数与 date 字段一起用作第一个参数。第二个参数为 MMM。

使用此格式 Qlik Sense 将第一个参数中的值转换为变量 MonthNames 中设置的相应月名称。在结果表中，我们创建的字段 month 的字段值显示此项。

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。

默认值设置为 '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

该设置定义了将1月的哪一天设置为定义第 1 周的参考日。换句话说，该设置规定了第 1 周中有多少天必须是 1 月内的日期。

语法：

ReferenceDay

ReferenceDay 设置一年中第一周包含的天数。ReferenceDay 可以设置为 1 和 7 之间的任何值。1-7 范围之外的任何值都被解释为周的中点 (4)，这相当于 ReferenceDay 设置为 4。

如果未为 ReferenceDay 设置选择值，则默认值将显示 ReferenceDay=0，该值将被解释为周的中点 (4)，如 ReferenceDay 表值所示。

ReferenceDay 函数通常与以下功能结合使用：

相关函数

变量	交互
BrokenWeeks (page 205)	如果Qlik Sense 应用程序在连续几周内运行，则将强制执行 ReferenceDay 变量设置。但是，如果正在使用中断周，则第 1 周将从 1 月 1 日开始，并与 FirstWeekDay 变量设置一起终止，并忽略该 ReferenceDay 标志。
FirstWeekDay (page 218)	整数用于定义将哪一天用作一周的第一天。

Qlik Sense 允许为 ReferenceDay 设置以下值：

ReferenceDay 值

值	参考日
0(默认)	1 月 4 日
1	1 月 1 日
2	1 月 2 日
3	1 月 3 日
4	1 月 4 日
5	1 月 5 日
6	1 月 6 日
7	1 月 7 日

在以下示例中，ReferenceDay = 3 将 1 月 3 日定义为参考日：

```
SET ReferenceDay=3; //(set January 3 as the reference day)
```

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例：

如果您想要周和周数的 ISO 设置，请确保脚本中包含以下内容：

```
Set FirstWeekDay=0;
Set BrokenWeeks=0;
Set ReferenceDay=4;    // Jan 4th is always in week 1
```

如果需要 US 设置，请确保脚本中包含以下内容：

```
Set FirstWeekDay=6;
Set BrokenWeeks=1;
Set ReferenceDay=1;     // Jan 1st is always in week 1
```

示例 1 – 使用默认值加载脚本;ReferenceDay=0

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- ReferenceDay 变量,设置为 0。
- BrokenWeeks 变量,设置为 0,这迫使应用程序使用连续数周。
- 从 2019 年底到 2020 年初的日期的数据集。

加载脚本

```
SET BrokenWeeks = 0;
SET ReferenceDay = 0;

Sales:
LOAD
date,
sales,
week(date) as week,
weekday(date) as weekday
Inline [
date,sales
12/27/2019,5000
12/28/2019,6000
12/29/2019,7000
12/30/2019,4000
12/31/2019,3000
01/01/2020,6000
01/02/2020,3000
01/03/2020,6000
01/04/2020,8000
01/05/2020,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- date
- week
- weekday

结果表

日期	周	weekday
2019 年 12 月 27 日	52	周五
2019 年 12 月 28 日	52	周六
2019 年 12 月 29 日	1	周日
2019 年 12 月 30 日	1	周一
2019 年 12 月 31 日	1	周二
2020 年 1 月 1 日	1	周三
2020 年 1 月 2 日	1	周四
2020 年 1 月 3 日	1	周五
2020 年 1 月 4 日	1	周六
2020 年 1 月 5 日	2	周日
2020 年 1 月 6 日	2	周一
2020 年 1 月 7 日	2	周二
2020 年 1 月 8 日	2	周三
2020 年 1 月 9 日	2	周四
2020 年 1 月 10 日	2	周五
2020 年 1 月 11 日	2	周六

第 52 周将于 12 月 28 日星期六结束。因为 ReferenceDay 需要将 1 月 4 日包括在第 1 周中，因此第 1 周从 12 月 29 日开始，到 1 月 4 号星期六结束。

示例 - ReferenceDay 变量设置为 5

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- ReferenceDay 变量，设置为 5。
- BrokenWeeks 变量，设置为 0，这迫使应用程序使用连续数周。
- 从 2019 年底到 2020 年初的日期的数据集。

加载脚本

```
SET BrokenWeeks = 0;
SET ReferenceDay = 5;

Sales:
LOAD
date,
sales,
week(date) as week,
weekday(date) as weekday
Inline [
date,sales
12/27/2019,5000
12/28/2019,6000
12/29/2019,7000
12/30/2019,4000
12/31/2019,3000
01/01/2020,6000
01/02/2020,3000
01/03/2020,6000
01/04/2020,8000
01/05/2020,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- week
- weekday

结果表

日期	周	weekday
2019 年 12 月 27 日	52	周五
2019 年 12 月 28 日	52	周六
2019 年 12 月 29 日	53	周日
2019 年 12 月 30 日	53	周一
2019 年 12 月 31 日	53	周二
2020 年 1 月 1 日	53	周三

日期	周	weekday
2020 年 1 月 2 日	53	周四
2020 年 1 月 3 日	53	周五
2020 年 1 月 4 日	53	周六
2020 年 1 月 5 日	1	周日
2020 年 1 月 6 日	1	周一
2020 年 1 月 7 日	1	周二
2020 年 1 月 8 日	1	周三
2020 年 1 月 9 日	1	周四
2020 年 1 月 10 日	1	周五
2020 年 1 月 11 日	1	周六

第 52 周将于 12 月 28 日星期六结束。BrokenWeeks 变量，这迫使应用程序使用连续数周。5 的参考日值要求将 1 月 5 日包括在第 1 周中。

然而，这是去年第 52 周结束后的 8 天。因此，第 53 周从 12 月 29 日开始，1 月 4 日结束。第 1 周从 1 月 5 日星期日开始。

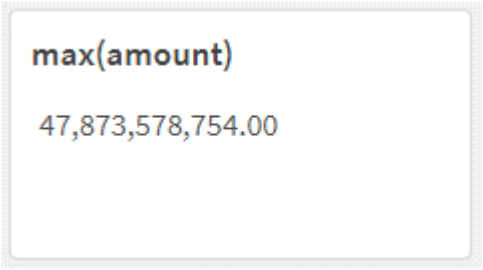
ThousandSep

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
```

使用 ThousandSep 变量的对象 Qlik Sense(带千位分隔符)



Qlik Sense 应用程序将符合此格式的文本字段解释为数字。当数字字段的**数字格式设定**属性设置为**数字**时，此格式设定将显示在图表对象中。

ThousandSep 在处理从多个区域设置接收的数据源时非常有用。

> *如果在应用程序中创建和格式化对象后修改了 ThousandSep 变量，则用户需要通过取消选择然后重新选择**数字格式设定**属性**数字**来重新格式化每个相关字段。*

以下示例显示了 ThousandSep 系统变量的可能用途：

```
Set ThousandSep=','; //(for example, seven billion will be displayed as: 7,000,000,000)

Set ThousandSep=' '; //(for example, seven billion will be displayed as: 7 000 000 000)
```

这些主题可以帮助您使用此函数:

相关主题

主题	说明
DecimalSep (page 216)	在文本字段解释的情况下,还必须遵守此函数提供的小数位分隔符设置。对于数字格式设定,将在必要时由 Qlik Sense 使用 **DecimalSep**。

区域设置

除非另有规定,本主题中的示例使用以下日期格式:MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素,系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者,您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典,则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 默认系统变量

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 加载到名为 Transactions 的表中的数据集。
- 使用默认 ThousandSep 变量定义。

加载脚本

```
Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,10000000441
01/02/2022,2,21237492432
01/03/2022,3,41249475336
```

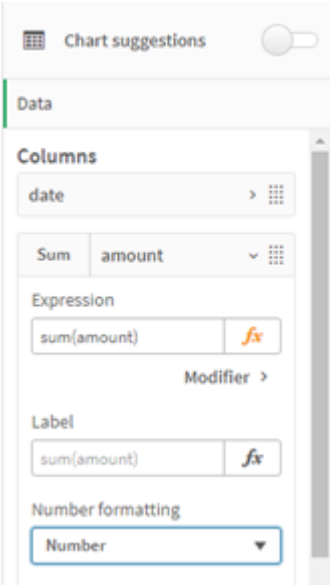
```
01/04/2022,4,24313369837
01/05/2022,5,47873578754
01/06/2022,6,24313884663
01/07/2022,7,28545883436
01/08/2022,8,35545828255
01/09/2022,9,37565817436
01/10/2022,10,3454343566
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将该字段添加为维度：date。
2. 添加以下度量：
   `=sum(amount)`
3. 在属性面板的**数据**下，选择度量。
4. 在**数字格式设定**下，选择**数字**。

调整图表度量的数字格式设定



结果表

日期	=sum(amount)
01/01/2022	10,000,000,441.00
01/02/2022	21,237,492,432.00
01/03/2022	41,249,475,336.00
01/04/2022	24,313,369,837.00
01/05/2022	47,873,578,754.00

日期	=sum(amount)
01/06/2022	24,313,884,663.00
01/07/2022	28,545,883,436.00
01/08/2022	35,545,828,255.00
01/09/2022	37,565,817,436.00
01/10/2022	3,454,343,566.00

在本例中，使用默认 ThousandSep 定义，该定义设置为逗号格式 (',')。在结果表中，金额字段的格式在千分组之间显示逗号。

示例 2 – 更改系统变量

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个示例中的相同数据集，加载到名为 Transactions 的表中。
- 修改脚本开头的 ThousandSep 定义，将 '*' 字符显示为千位分隔符。这是一个极端的例子，仅用于演示变量的功能。

本例中使用的修改是极端的，并不常用，但在这里显示是为了演示变量的功能。

加载脚本

```
SET ThousandSep='*';

Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,10000000441
01/02/2022,2,21237492432
01/03/2022,3,41249475336
01/04/2022,4,24313369837
01/05/2022,5,47873578754
01/06/2022,6,24313884663
01/07/2022,7,28545883436
01/08/2022,8,35545828255
01/09/2022,9,37565817436
```

```
01/10/2022,10,3454343566
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将该字段添加为维度：date。
2. 添加以下度量：
   `=sum(amount)`
3. 在“属性”面板的**数据**下，选择度量。
4. 在**数字格式设定**下，选择**自定义**。

结果表

日期	=sum(amount)
01/01/2022	10*000*000*441.00
01/02/2022	21*237*492*432.00
01/03/2022	41*249*475*336.00
01/04/2022	24*313*369*837.00
01/05/2022	47*873*578*754.00
01/06/2022	24*313*884*663.00
01/07/2022	28*545*883*436.00
01/08/2022	35*545*828*255.00
01/09/2022	37*565*817*436.00
01/10/2022	3*454*343*566.00

在脚本开头时，ThousandSep 系统变量被修改为 '*'。在结果表中，可以看到金额字段的格式显示千分组之间的 '*'。

示例 3 – 文本解释

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Transactions 的表中的数据集。
- 具有文本格式的数字字段的数据，使用逗号作为千位分隔符。
- 使用默认 ThousandSep 系统变量 。

加载脚本

```
Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'10,000,000,441'
01/02/2022,2,'21,492,432'
01/03/2022,3,'4,249,475,336'
01/04/2022,4,'24,313,369,837'
01/05/2022,5,'4,873,578,754'
01/06/2022,6,'313,884,663'
01/07/2022,7,'2,545,883,436'
01/08/2022,8,'545,828,255'
01/09/2022,9,'37,565,817,436'
01/10/2022,10,'3,454,343,566'
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将该字段添加为维度：date。
2. 添加以下度量：
   `=sum(amount)`
3. 在“属性”面板的**数据**下，选择度量。
4. 在**数字格式设定**下，选择**数字**。
5. 添加以下度量以评估金额字段是否为数值：
   `=isnum(amount)`

结果表

日期	=sum(amount)	=isnum(amount)
01/01/2022	10,000,000,441.00	-1
01/02/2022	21,492,432.00	-1
01/03/2022	4,249,475,336.00	-1
01/04/2022	24,313,369,837.00	-1
01/05/2022	4,873,578,754.00	-1
01/06/2022	313,884,663.00	-1
01/07/2022	2,545,883,436.00	-1
01/08/2022	545,828,255.00	-1

日期	=sum(amount)	=isnum(amount)
01/09/2022	37,565,817,436.00	-1
01/10/2022	3*454*343*566.00	-1

加载数据后，我们可以看到，由于数据符合 ThousandSep 变量，因此 Qlik Sense 已将金额字段解释为数值。这可以通过 isnum() 函数来证明，该函数将每个条目计算为 -1 或 TRUE。

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

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
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

4.9 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]';
```

4.10 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。
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
```

5 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。

6 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。

6.1 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 ' ');
```

6.2 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

集合表达式可以在聚合函数内外使用,并用花括号括起来。

示例:内部集合表达式

```
Sum( {$<Year={2021}>} Sales )
```

示例:外部集合表达式

```
{<Year={2021}>} Sum(Sales) / Count(distinct Customer)
```

集合表达式包含以下元件的组合:

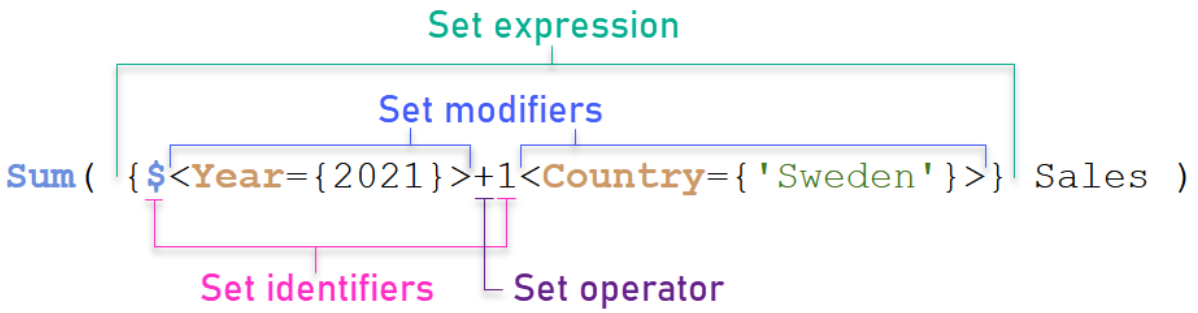
- **标识符**。集合标识符表示在别处定义的选择。它还表示数据中的一组特定记录。它可以是当前选择、书签选择或备用状态选择。简单的集合表达式包含一个单一的标识符(如美元符号{$}),这意味着当前选择项中的所有记录。

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。

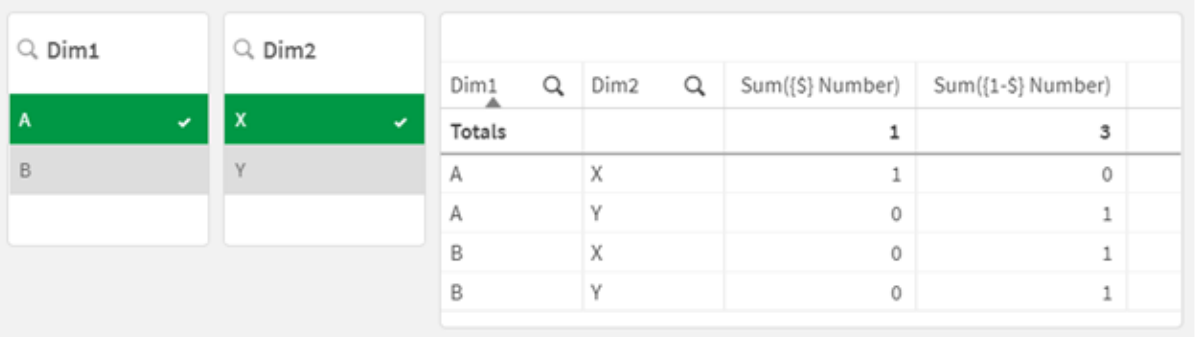
例如，{1-$} 给定的集合不能总是由选择定义。因此，它不是一个自然的集合。这可以通过加载以下数据、将其添加到表中，然后使用筛选窗格进行选择来显示。

```
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



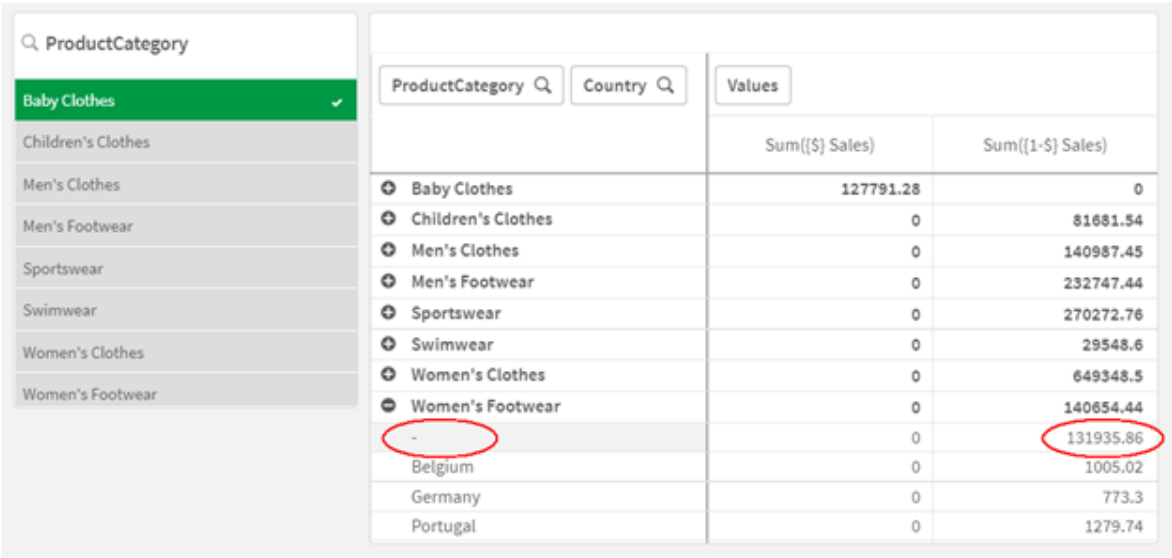
第一个度量中的集合表达式使用自然集合：它对应于所做的 {$} 选择 。

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。

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。
$	表示当前选择项的记录。因此，集合表达式 **{$}** 与不陈述集合表达式的意义等同。
$1	表示上一个选择项。$2 表示上一个选择项，但只表示一个，以此类推。
$_1	表示下一个(前进)选择项。$_2 表示下一个选择项，但只表示一个，以此类推。
BM01	您可以使用任何书签 ID 或书签名称。
MyAltState	您可以使用状态名称引用处于备用状态的这些选择项。

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。

示例	结果
sum({$<OrderDate = DeliveryDate>} Sales)	返回当前选择项的销售额，其中 OrderDate = DeliveryDate。

示例	结果
sum({1<Region = {US}>} Sales)	返回美国地区的销售额，忽略当前选择项
sum({$<Region = >} Sales)	返回选择项的销售额，但移除 *Region* 中的选择项。
sum({<Region = >} Sales)	返回与上述示例相同的销售额。当省略要修改的集合时，则假定 $。
sum({$<Year={2000}, Region={"U*"}>} Sales)	返回当前选择项的销售额，但 *Year* 和 *Region* 中均有新选择项。

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`。

$ 集合标识符与不声明集合标识符相同。例如，在上面的例子中，表达式 `Sum( {$<Year = {2009}>} Sales )` 等价于 `Sum( {<Year = {2009}>} Sales )`。

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。
`$`（或无集合标识符）	表示默认状态下的当前选择项的记录。因此，集合表达式 `{$}` 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。

示例	结果
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。
*	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

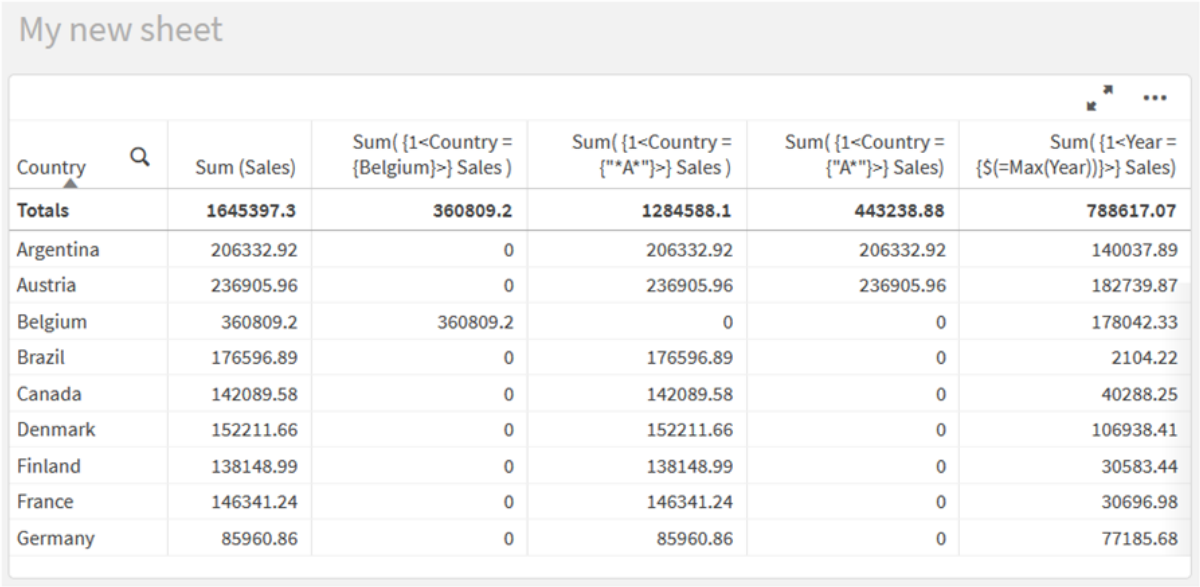
国家	Sum(Sales)	Sum ({1<Country= {Belgium}>} Sales)	Sum ({1<Country= {"*A*"}>} Sales)	Sum ({1<Country= {"A*"}>} Sales)	Sum({1<Year= {$(=Max (Year))}>} Sales)
总计	1645397.3	360809.2	1284588.1	443238.88	788617.07
阿根廷	206332.92	0	206332.92	206332.92	140037.89
奥地利	236905.96	0	236905.96	236905.96	182739.87
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
德国	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。

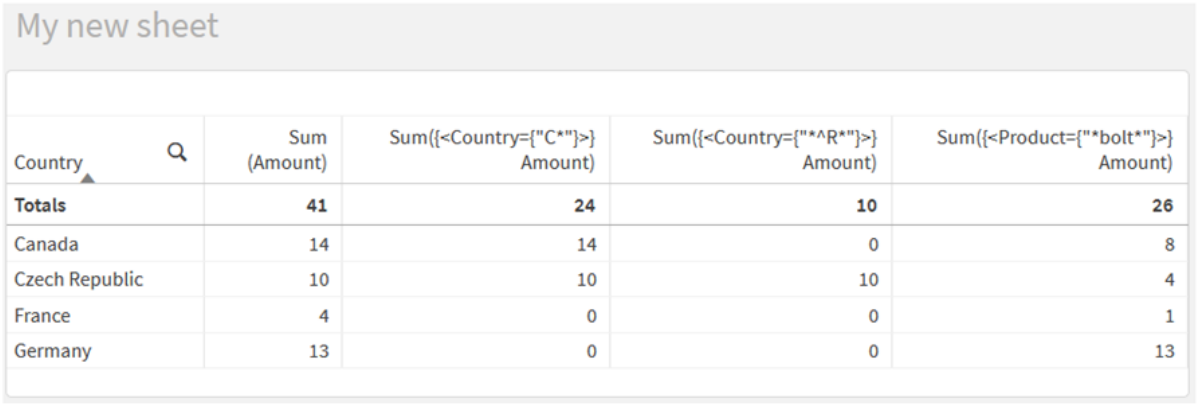
表格 - 带文本搜索的集合修饰符

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
捷克共和国	10	10	10	4
法国	4	0	0	1
德国	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。

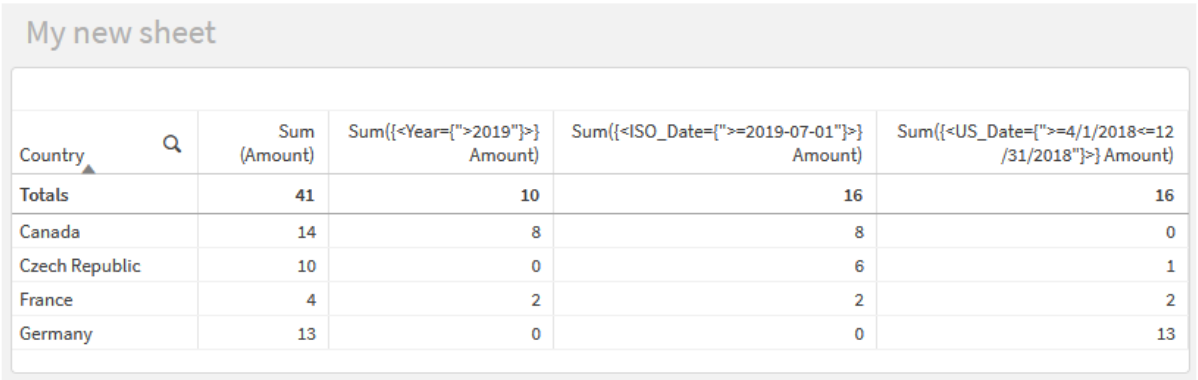
表 - 带有数字搜索的集合表达式

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
德国	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。

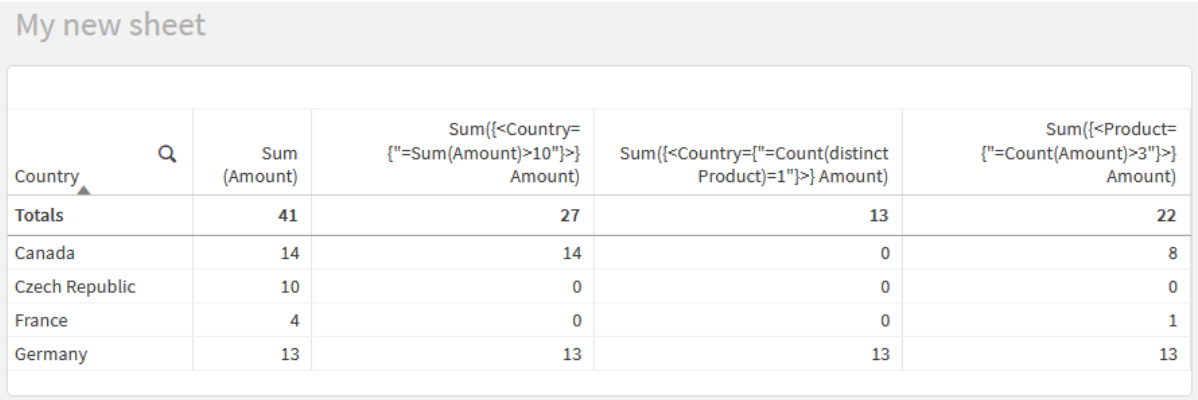
Table - Set modifiers with expression searches

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



示例	结果
sum({$–1<Product = {“*Internal*”, “*Domestic*”}>} Sales)	返回当前选择项的销售额，排除产品名中包含字符串 'Internal' 或 'Domestic' 的产品的相关交易。
sum({$<Customer = {“=Sum({1<Year = {2007}>} Sales) > 1000000”}>} Sales)	返回当前选择项的销售额，但“Customer”字段中有新选择项：仅限在 2007 年总销售额超过 1000000 的客户。

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。

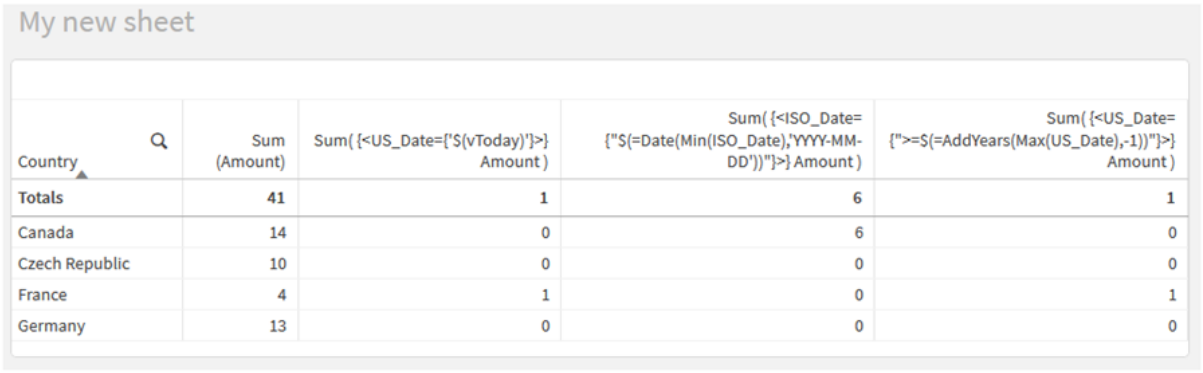
表 - 集合修饰符和美元符号扩展

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
德国	13	0	0	0

解释

- 维度：
  - Country
- 度量：
  - Sum(Amount)
    没有集合表达式的总和 Amount。
  - Sum({<US_Date={'$(vToday)'}>}Amount)
    总和 Amount，针对和变量 vToday 中 US_Date 一样的所有记录。
  - Sum({<ISO_Date={"$(=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


My new sheet

Country	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)
Totals	**41**	**1**	**6**	**1**
Canada	14	0	6	0
Czech Republic	10	0	0	0
France	4	1	0	1
Germany	13	0	0	0

示例	结果
sum({$<Year = {$(#vLastYear)}>} Sales)	返回与当前选择项相关的上一年的销售额。此处，将包含相关年份的变量 vLastYear 用于美元符号扩展。
sum({$<Year = {$(#=Only (Year)-1)}>} Sales)	返回与当前选择项相关的上一年的销售额。在这里，美元符号扩展被用于计算上一年份。

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。

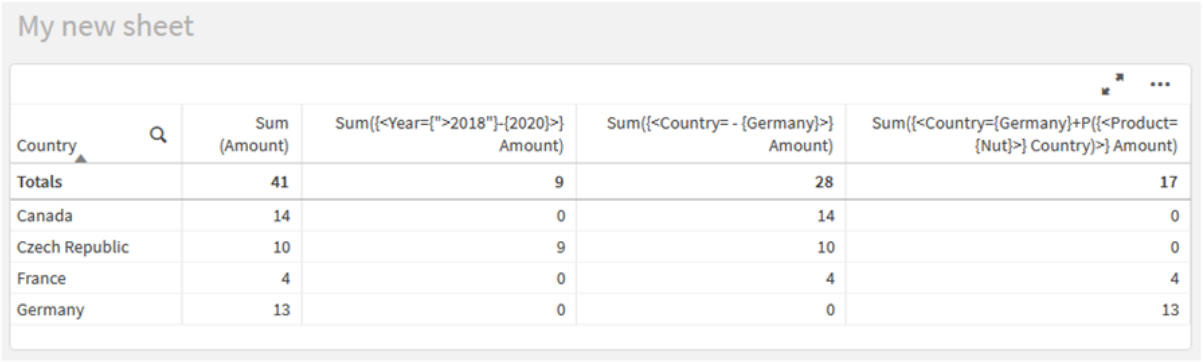
表格 - 集合修饰符和集合运算符

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
德国	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


My new sheet

Country	Sum (Amount)	Sum({<Year={">2018"}-{2020}>} Amount)	Sum({<Country= - {Germany}>} Amount)	Sum({<Country={Germany}+P({<Product={Nut}>} Country)>} Amount)
Totals	**41**	**9**	**28**	**17**
Canada	14	0	14	0
Czech Republic	10	9	10	0
France	4	0	4	4
Germany	13	0	0	13

示例	结果
sum({$<Product = Product + {OurProduct1} – {OurProduct2} >} Sales)	返回当前选择项的销售额，但将产品“OurProduct1”添加到所选产品列表中，并从所选产品列表中移除“OurProduct2”。
sum({$<Year = Year + ({“20*”,1997} – {2000}) >} Sales)	返回当前选择项的销售额，但字段“Year”中包含附加的选择项：1997 年和以“20”开头的所有年份，但不包括 2000 年。 注意：如果当前选择项中包含 2000，它也将在修改后被包括进来。
sum({$<Year = (Year + {“20*”,1997}) – {2000} >} Sales)	返回的结果几乎与上例相同，但同时如果 2000 年最初包含在当前选择项中，此时将排除 2000 年。该例显示的是使用括号定义优先顺序的重要性。
sum({$<Year = {“*”} – {2000}, Product = {“*bearing*”} >} Sales)	返回当前选择项的销售额，但在“Year”中包含新选择项：除 2000 年以外的所有年份；并且仅针对包含字符串 'bearing' 的产品。

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

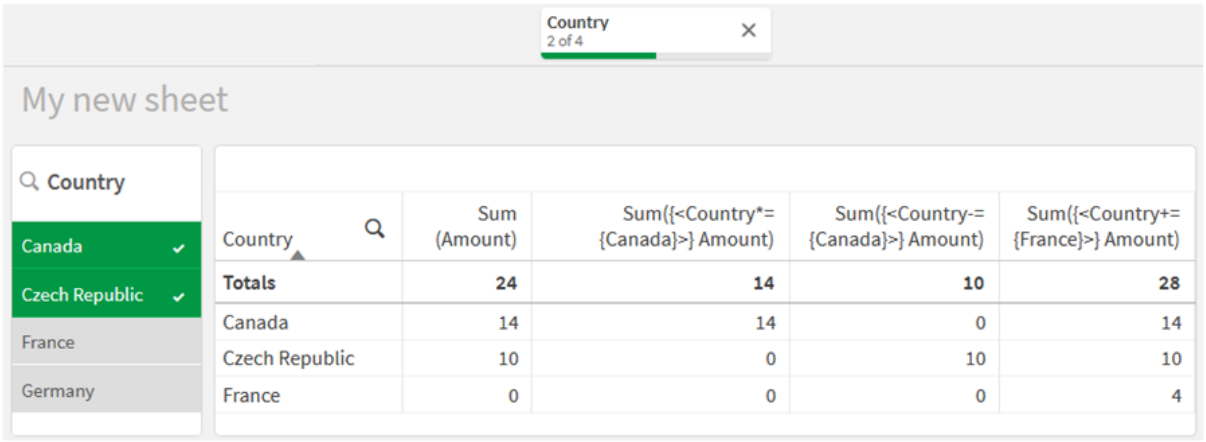
国家	Sum (Amount)	Sum({<Country*= {Canada}>} Amount)	Sum({<Country-= {Canada}>} Amount)	Sum({<Country+= {France}>} Amount)
总计	**24**	**14**	**10**	**28**
加拿大	14	14	0	14
捷克	10	0	10	10

国家	Sum (Amount)	Sum({<Country*= {Canada}>} Amount)	Sum({<Country-= {Canada}>} Amount)	Sum({<Country+= {France}>} Amount)
总计	**24**	**14**	**10**	**28**
共和国				
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



示例	结果
sum({$<Product += {OurProduct1, OurProduct2} >} Sales)	返回当前选择项的销售额，但使用默认并集将产品“OurProduct1”和“OurProduct2”添加到所选产品列表。

示例	结果
sum({$<Year += {"20*",1997} -{2000} >} Sales)	返回当前选择项的销售额，但使用默认并集在选择项中添加许多年份：1997 年和所有以"20"开头的年份，但不包括 2000。 注意：如果当前选择项中包含 2000，它也将在修改后被包括进来。如同 <Year=Year + ({"20*",1997}-{2000})>。
sum({$<Product *= {OurProduct1} >} Sales)	返回当前选择项的销售额，但仅针对当前所选产品和 OurProduct1 产品的交集。

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

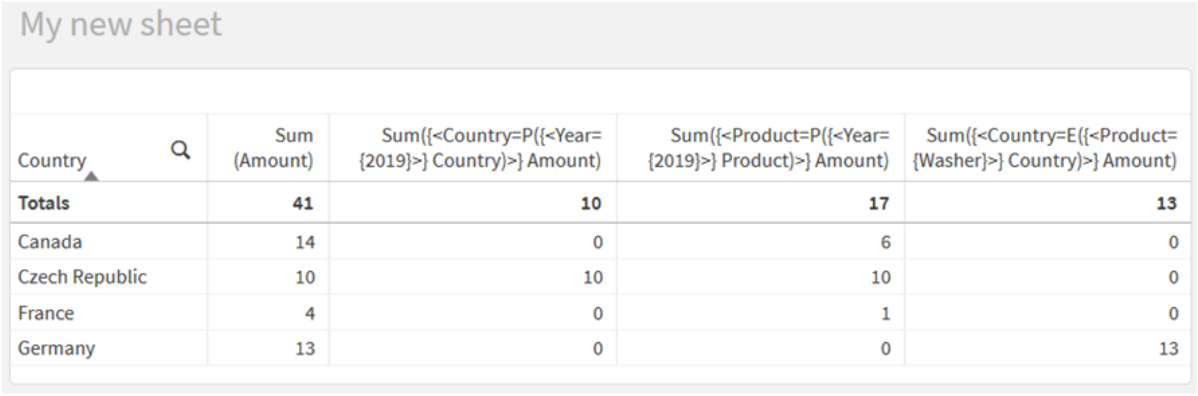
国家	Sum (Amount)	Sum({<Country=P ({<Year= {2019}>}Country)>} Amount)	Sum({<Product=P ({<Year= {2019}>}Product)>} Amount)	Sum({<Country=E ({<Product= {Washer}>}Country)>} Amount)
总计	**41**	**10**	**17**	**13**
加拿大	14	0	6	0
捷克共和国	10	10	10	0
法国	4	0	1	0
德国	13	0	0	13

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


My new sheet

Country	Sum (Amount)	Sum({<Country=P({<Year={2019}>} Country)>} Amount)	Sum({<Product=P({<Year={2019}>} Product)>} Amount)	Sum({<Country=E({<Product={Washer}>} Country)>} Amount)
Totals	**41**	**10**	**17**	**13**
Canada	14	0	6	0
Czech Republic	10	10	10	0
France	4	0	1	0
Germany	13	0	0	13

示例	结果
sum({$<Customer = P({1<Product={'Shoe'}>} Customer)>} Sales)	返回当前选择项的销售额，但仅限购买过产品“Shoe”的客户。元素函数 P( ) 在此返回可能的客户列表；即字段 Product 中的选择项“Shoe”暗指的那些客户。
sum({$<Customer = P({1<Product={'Shoe'}>})>} Sales)	同上。如果省略 Element 函数中的字段，该函数将返回外部任务中指定字段的正值。
sum({$<Customer = P({1<Product={'Shoe'}>} Supplier)>} Sales)	返回当前选择项的销售额，但仅限提供过产品“Shoe”的客户，即客户也是供应商。元素函数 P( ) 在此返回可能的供应商列表；即字段 Product 中的选择项“Shoe”暗指的那些供应商。供应商列表随后用作字段 Customer 中的选择项。
sum({$<Customer = E({1<Product={'Shoe'}>})>} Sales)	返回当前选择项的销售额，但仅限从未购买过产品“Shoe”的那些客户。元素函数 E( ) 在此返回排除的客户列表；即根据字段 Product 中的选择项“Shoe”排除的那些客户。

内部和外部集合表达式

集合表达式可以在聚合函数内外使用，并用花括号括起来。

在聚合函数中使用集合表达式时，它可能如下所示：

示例：内部集合表达式

```
Sum( {$<Year={2021}>} Sales )
```

如果表达式具有多个聚合，并且希望避免在每个聚合函数中写入相同的集合表达式，请在聚合函数外使用集合表达式。

如果使用外部集合表达式，则必须将其放置在范围的开头。

示例：外部集合表达式

```
{<Year={2021}>} Sum(Sales) / Count(distinct Customer)
```

如果在聚合函数之外使用集合表达式，也可以将其应用于现有的主度量值。

示例：应用于主度量值的外部集表达式

```
{<Year={2021}>} [Master Measure]
```

在聚合函数外部使用的集合表达式会影响整个表达式，除非它用括号括起来，否则括号定义范围。在下面的词法范围示例中，集合表达式仅应用于括号内的聚合。

示例：词法定界

```
( {<Year={2021}>} Sum(Amount) / Count(distinct Customer) ) - Avg(CustomerSales)
```

规则

词法定界

除非用括号括起来，否则集合表达式会影响整个表达式。如果是，括号定义词法范围。

位置

集合表达式必须放在词法范围的开头。

上下文

上下文是与表达式相关的选择。传统上，上下文始终是当前选择的默认状态。但如果对象设置为备用状态，则上下文是当前选择的备用状态。

还可以以外部集表达式的形式定义上下文。

继承

内部集合表达式优先于外部集合表达式。如果内部集合表达式包含集合标识符，它将替换上下文。否则，将合并上下文和集合表达式。

- `{$<SetExpression>}` - 覆盖外部集合表达式
- `{<SetExpression>}` - 与外部集合表达式合并

元素集分配

元素集指定决定了如何合并两个选择。如果使用法线等号，则内部集合表达式中的选择具有优先权。否则，将使用隐式集合运算符。

- `{<Field={value}>}` - 此内部选择将替换 “`Field`” 中的任何外部选择。
- `{<Field+={value}>}` - 使用并集运算符，此内部选择与 “`Field`” 中的外部选择合并。
- `{<Field*={value}>}` - 使用交叉运算符，此内部选择与 “`Field`” 中的外部选择合并。

多步骤继承

继承可以分多个步骤进行。示例：

- 当前选择项 → `Sum(Amount)`
  聚合函数将使用上下文，这是当前选择。

- 当前选择项 → {<Set1>} Sum(Amount)
  Set1 将从当前选择继承，结果将是聚合函数的上下文。
- 当前选择项 → {<Set1>} ({<Set2>} Sum(Amount))
  Set2 将从 Set1 继承，其将从当前选择继承，结果将是聚合函数的上下文。

Aggr() 函数

Aggr() 函数创建一个具有两个独立聚合的嵌套聚合。在下面的示例中，为 Dim 的每个值计算 Count()，并使用 Sum() 函数聚合得到的数组。

示例：

Sum(Aggr(Count(X),Dim))
Count() 是内部聚合，Sum() 是外部聚合。

- 内部聚合未从外部聚合继承任何上下文。
- 内部聚合从 Aggr() 函数继承上下文，该函数可能包含集合表达式。
- Aggr() 函数和外部聚合函数都从外部集表达式继承上下文。

教程 - 创建集合表达式

可以构建 Qlik Sense 中的集表达式 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，该范围不同于应用程序中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

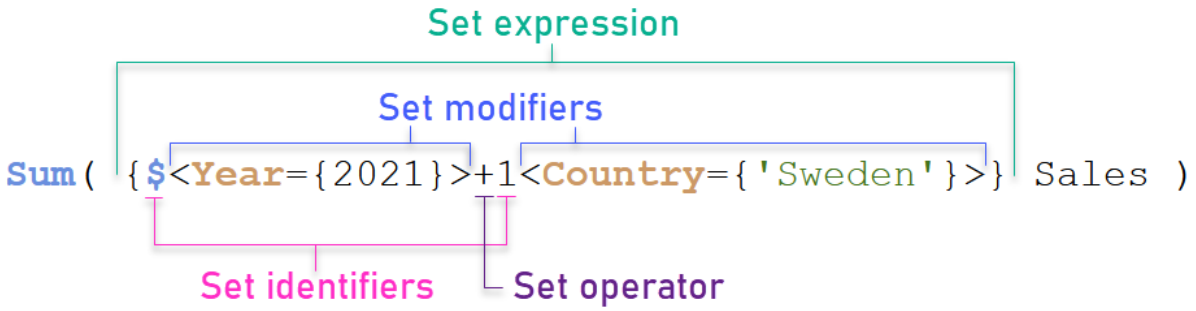
Qlik Sense Enterprise Professional 访问权限分配，允许您加载数据和创建应用程序。

- 集合分析第一部分：初学者简介
- 集合分析第 2 部分

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 306)*
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 )
```

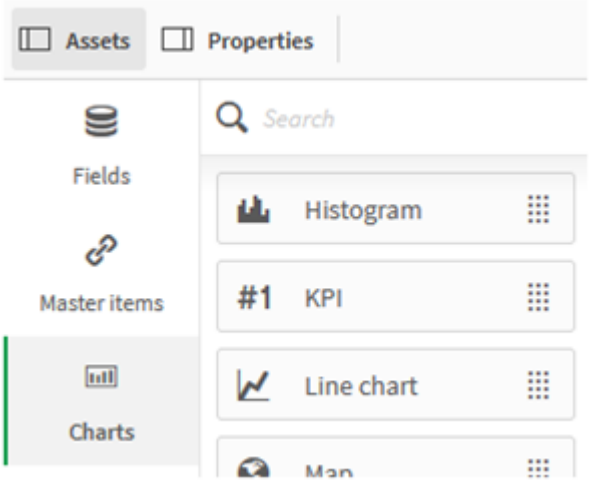
修饰符为：

```
<Year = {2015}>
```

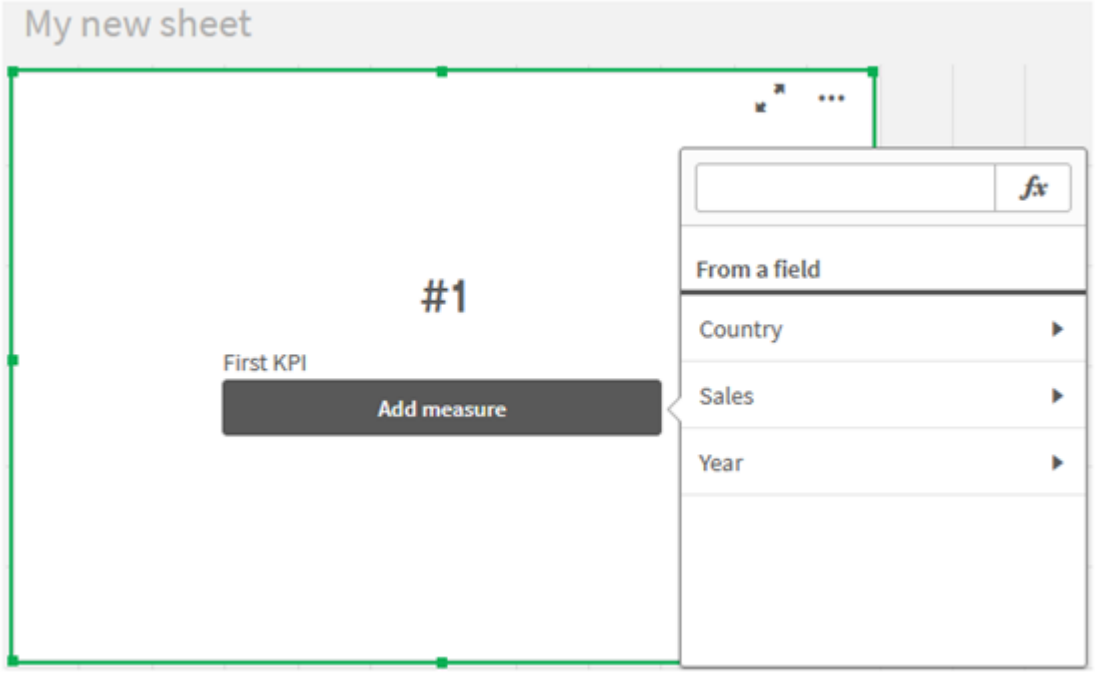
此修饰符指定仅选择 2015 年的数据。包含修饰符的花括号表示集合表达式。

执行以下操作：

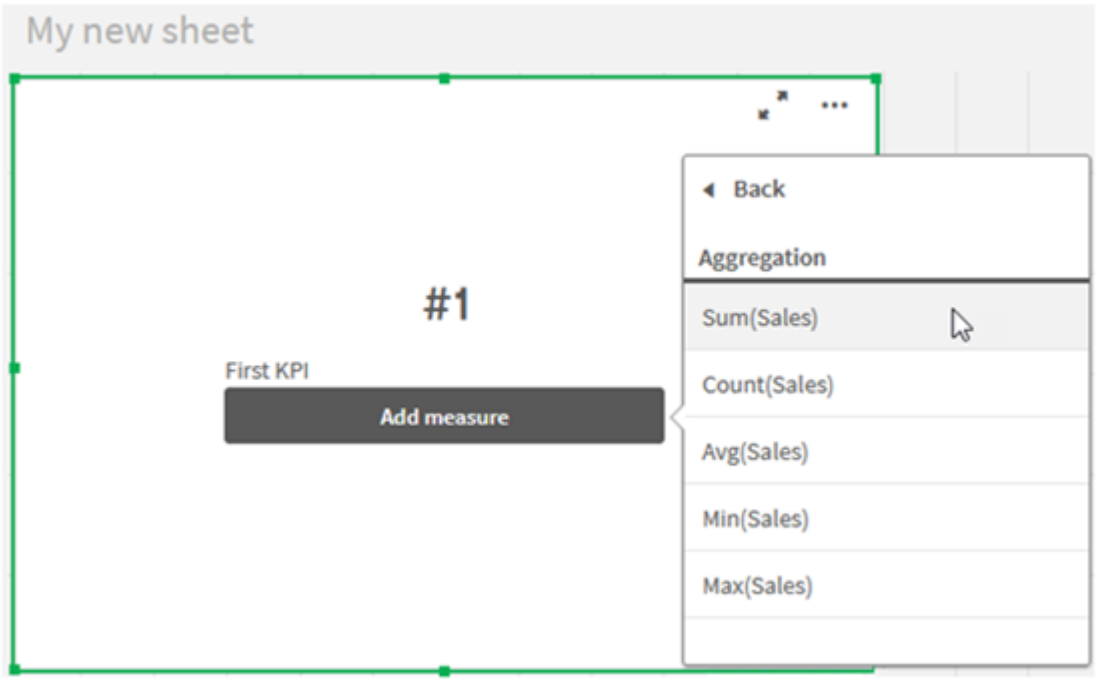
1. 在工作表中，从导航栏打开**资产**面板，然后单击**图表**。



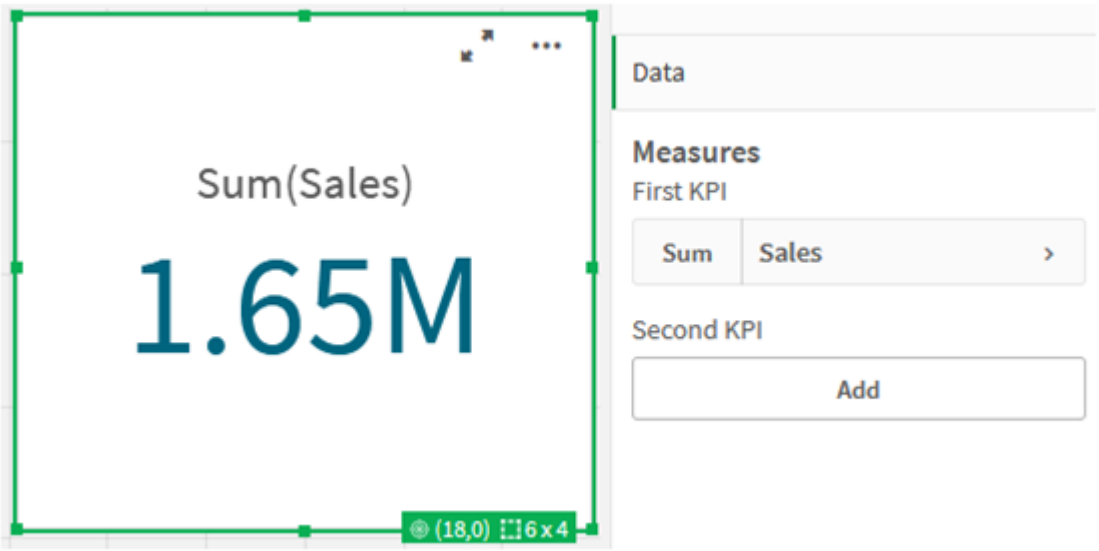
2. 将 **KPI** 拖到工作表上，然后单击**添加度量**。



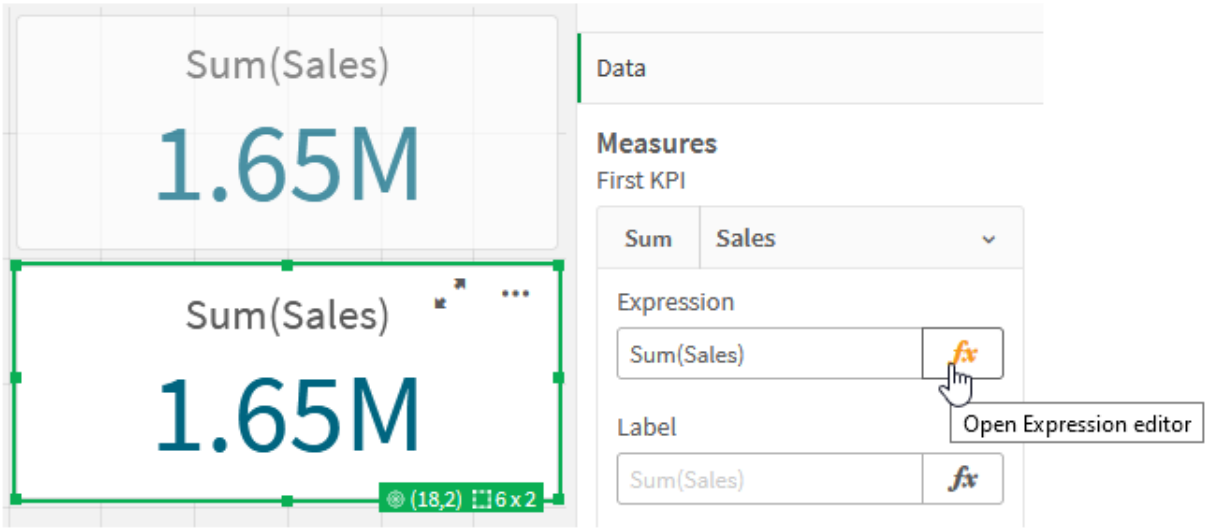
3. 单击 `Sales`，然后为聚合选择 `Sum(Sales)`。



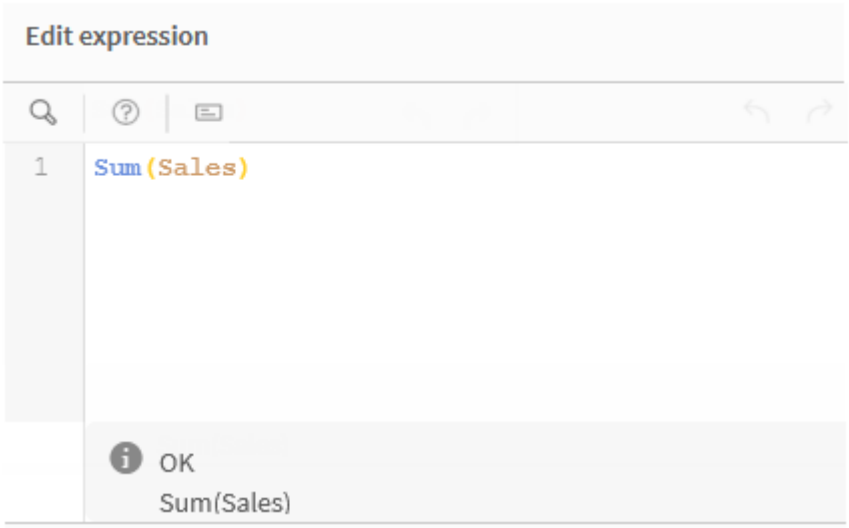
KPI 显示所有年份的销售额总和。



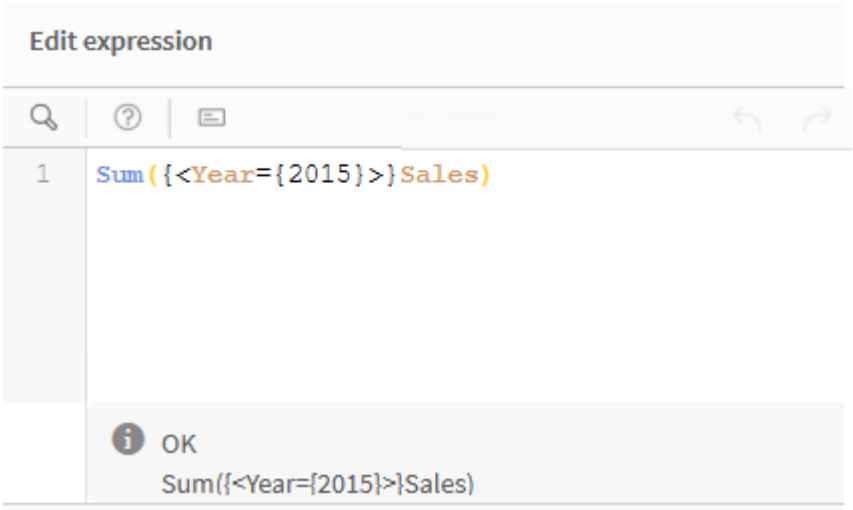
4. 复制并粘贴 KPI 以新建 KPI。
5. 单击新的 KPI，单击**度量**下方的**销售**，然后单击**打开表达式编辑器**。



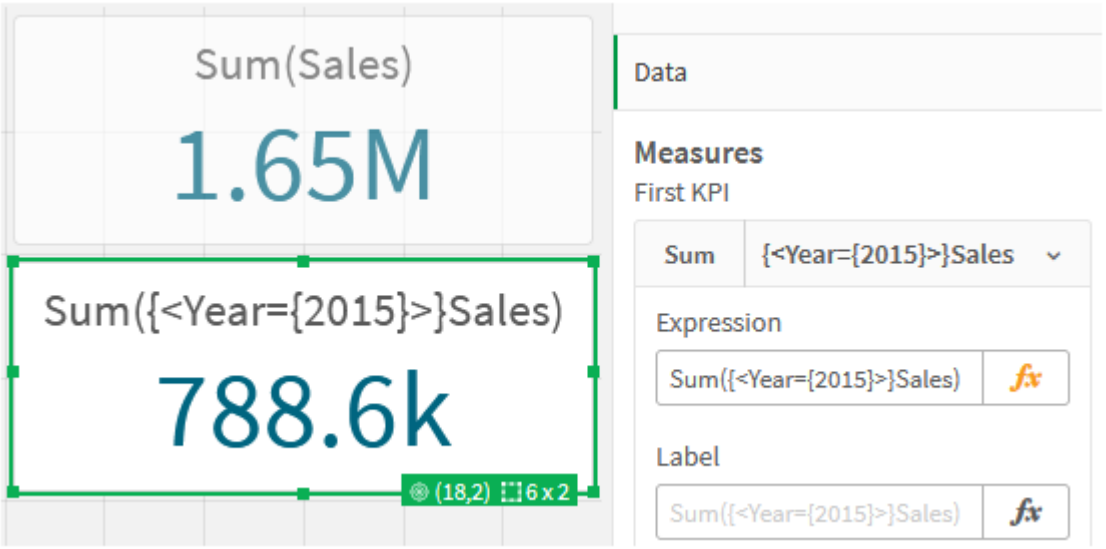
表达式编辑器连同聚合 `Sum(Sales)` 打开。



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)`

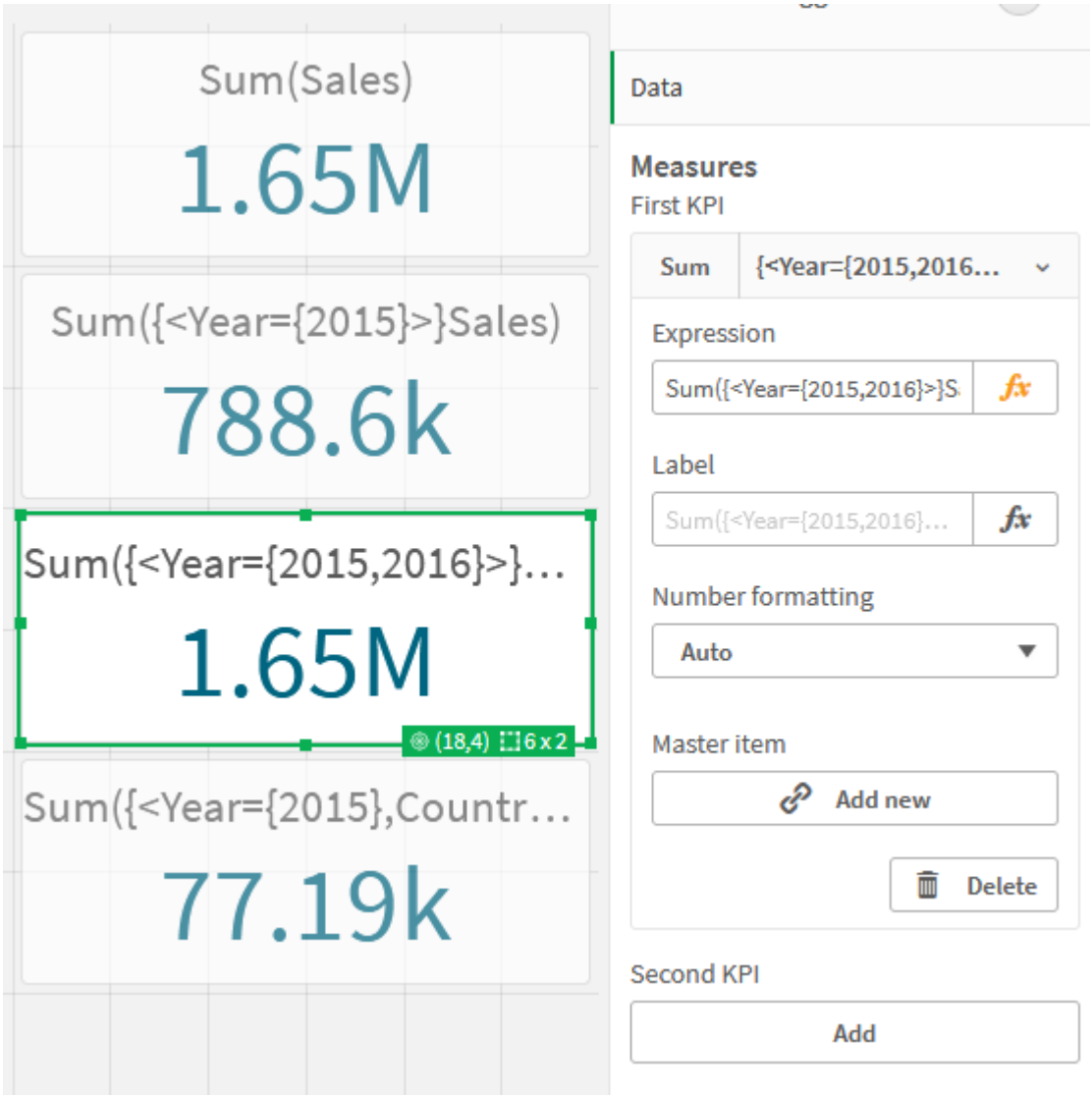


iii. 单击**应用**保存表达式并关闭表达式编辑器。2015 年的 Sales总和如 KPI 中所示。



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：

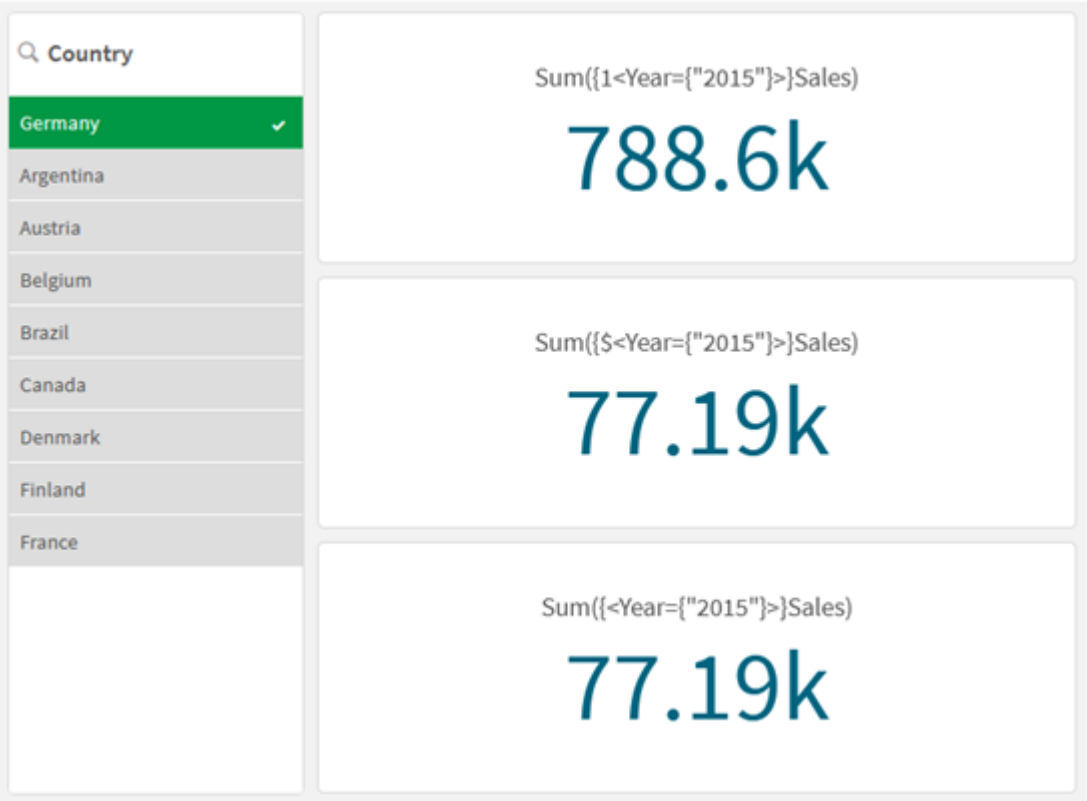
```
Sum({$<Year={"2015"}>}Sales)
```

$ 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。

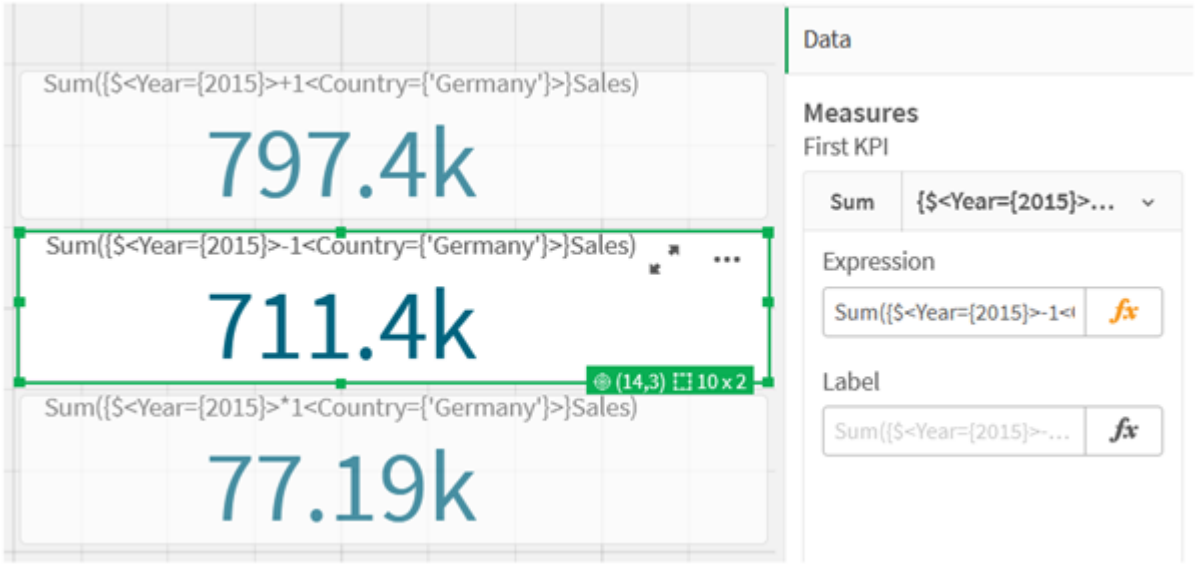
使用加号 (+) 运算符的 KPI

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 "
```

6.3 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。

6.4 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。

7 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

7.1 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

7.2 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。

7.3 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。

7.4 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

7.5 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)

8 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。

8.1 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”。

8.2 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。

命名实体时，避免将同一名称指定给多个字段、变量或度量。解决同名实体之间的冲突有严格的优先顺序。这种顺序反映在使用这些实体的任何对象或上下文中。该优先顺序如下：

- 在聚合中，字段优先于变量。度量标签在聚合中不相关，并且没有排定优先级。
- 在聚合外部，度量标签优先于变量，而变量又优先于字段名称。
- 此外，在聚合外部，度量可以通过引用其标签来重用，除非标签实际上是计算标签。在这种情况下，度量的重要性降低，以减少自引用的风险，并且在这种情况下，名称将始终首先解释为度量标签，其次解释为字段名称，再次解释为变量名称。

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

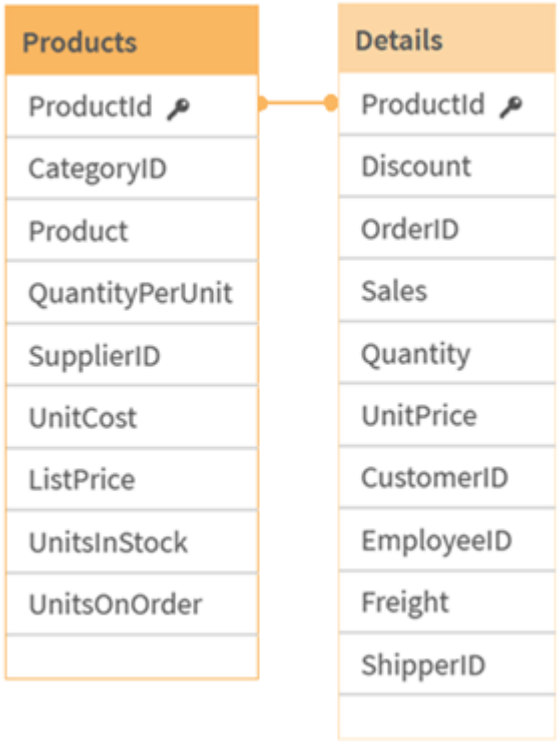
聚合的计算方式意味着您不能聚合关键字段，因为不清楚应该使用哪个表进行聚合。例如，如果字段 <Key> 链接两个表，则不清楚 **Count**(<Key>) 是否应返回第一个表或第二个表中的记录数。

但是，如果使用 **distinct** 子句，则聚合定义良好，可以为跨两个表链接的关键字段计算。

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。

Products 和 Details 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。

包含在三个或多个表中的关键字段的聚合

distinct 前缀仅适用于最多链接两个表的关键字段。当对存在于三个或多个表中的关键字段进行聚合分组时,任何需要字段的频率信息的操作都将返回 NULL。如果关键字段链接三个或多个表,则必须改为使用该字段的非关键副本。

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() 用于查找聚合数据的最高值。通过指定 **rank** n，可以查找第 n 个最高值。

```
Max - 图表函数([{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。

参数	说明
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。

示例	结果

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。

参数	说明
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
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`。

示例	结果
`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() 用于查找聚合数据的最高值。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
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。

示例	结果
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 319)*
- *RangeMax (page 1288)*

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
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

Customer	Product	UnitSales	UnitPrice
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 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 319)*
- *RangeMin (page 1292)*

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 383)*
- *Median - 图表函数 (page 418)*

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。

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。

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。*
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

Customer	Product	UnitSales	UnitPrice
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。

参数	说明
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	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。
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。

参数	说明
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(每年)，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时间表(不必为周期性的)。所有付款按 365 天一年年折扣。

```
XIRR (valueexpression, dateexpression )
```

NPV

NPV() 脚本函数接受折扣率和按期间排序的多个值。在这些计算中，流入(收入)为正值，流出(未来付款)为负值。这些发生在每个周期结束时。

```
NPV (rate, expression)
```

XNPV

XNPV() 函数用于返回聚合净现值，以揭示 **pmt** 和 **date** 表达式的成对数值表示的现金流时间表(不必为周期性的)。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(不必为周期性的)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([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。

该函数使用牛顿法的简化版本来计算内部回报率 (IRR)。

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。

该函数使用牛顿法的简化版本来计算内部回报率 (IRR)。

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 366)*
- *Aggr - 图表函数 (page 517)*

NPV

NPV() 脚本函数接受折扣率和按期间排序的多个值。在这些计算中，流入(收入)为正值，流出(未来付款)为负值。这些发生在每个周期结束时。

净现值 (NPV) 用于计算未来现金流的当前总价值。为了计算净现值，我们需要估计每个期间的未来现金流，并确定正确的贴现率。**NPV()** 脚本函数接受贴现率和按期间排序的多个值。在这些计算中，流入(收入)为正值，流出(未来付款)为负值。这些发生在每个周期结束时。

语法：

```
NPV(discount_rate, value)
```

返回数据类型： 数字。默认情况下，结果将被格式化为货币。

净现值的计算公式为：

$$NPV = \sum_{t=1}^{n} \frac{R_t}{(1+i)^t}$$

其中：

- R_t = 单期净现金流入流出 t
- i = 替代投资可能获得的贴现率或回报
- t = 定时器周期数

参数

参数	说明
discount_rate	**discount_rate** 是应用的折扣百分比。 值 0.1 表示 10% 的贴现率。
value	此字段保存按期间排序的多个期间的值。假设第一个值是期间 1 结束时的现金流，依此类推。

限制：

NPV() 函数具有以下限制：

- 文本值，NULL 值和缺失值都忽略不计。
- 现金流量值必须按期间升序排列。

适用场景

NPV() 是一种财务函数，用于检查项目盈利能力并得出其他度量。当现金流可用作原始数据时，此函数非常有用。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 一次性付款(脚本)

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 一个项目及其一个期间的现金流的数据集,该数据集加载到名为 CashFlow 的表中。
- CashFlow 表中的常驻荷载,用于计算名为 NPV 的表中项目的 NPV 字段。
- 硬编码贴现率为 10%,用于净现值计算。
- 用于对项目的所有付款进行分组的 Group By 报表。

加载脚本

```
CashFlow:
Load
*
Inline
[
PrjId,PeriodId,Values
1,1,1000
];

NPV:
Load
       PrjId,
       NPV(0.1,Values) as NPV //Discount Rate of 10%
Resident CashFlow
Group By PrjId;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- PrjId
- NPV

结果表

PrjId	NPV
1	$909.09

对于在一个周期结束时收到的 1000 美元的单笔付款,在每个周期 10% 的贴现率下,净现值等于 1000 美元除以(1 + 贴现率)。有效净现值等于 909.09 美元

示例 2 – 多次付款(脚本)

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 一个项目及其多个期间的现金流的数据集,该数据集加载到名为 CashFlow 的表中。
- CashFlow 表中的常驻荷载,用于计算名为 NPV 的表中项目的 NPV 字段。
- 硬编码贴现率为 10% (0.1),用于净现值计算。
- 用于对项目的所有付款进行分组的 Group By 报表。

加载脚本

```
CashFlow:
Load
*
Inline
[
PrjId,PeriodId,Values
1,1,1000
1,2,1000
];

NPV:
Load
       PrjId,
       NPV(0.1,Values) as NPV //Discount Rate of 10%
Resident CashFlow
Group By PrjId;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- PrjId
- NPV

结果表

PrjId	NPV
1	$1735.54

对于在两个周期结束时收到的 1000 美元付款,按每个周期 10% 的贴现率计算,有效净现值等于 1735.54 美元。

示例 3 – 多次付款(脚本)

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 两个项目的贴现率,它被加载到名为 Project 的表中。
- 按项目 ID 和期间 ID 列出每个项目的多个期间的现金流。如果数据未排序,则可以使用此期间 ID 对记录进行排序。
- NoConcatenate、常驻加载和 Left Join 函数的组合用于创建临时表 tmpNPV。该表将 Project 和 CashFlow 表的记录合并为一个平面表。此表将重复每个期间的贴现率。
- tmpNPV 表中的常驻荷载,用于计算名为 NPV 的表中每个项目的 NPV 字段。
- 与每个项目关联的单个值贴现率。这是使用 only() 函数检索的,并用于每个项目的净现值计算。
- 一个 Group By 语句,用于按项目 ID 对每个项目的所有付款进行分组。

为了避免将任何合成或冗余数据加载到数据模型中,将在脚本末尾删除 tmpNPV 表。

加载脚本

```
Project:
Load * inline [
PrjId,Discount_Rate
1,0.1
2,0.15
];

CashFlow:
Load
*
Inline
[
PrjId,PeriodId,Values
1,1,1000
1,2,1000
1,3,1000
2,1,500
2,2,500
2,3,1000
2,4,1000
];

tmpNPV:
NoConcatenate Load *
Resident Project;
Left Join
Load *
```

```
Resident CashFlow;

NPV:
Load
       PrjId,
       NPV(Only(Discount_Rate),Values) as NPV //Discount Rate will be 10% for Project 1 and 15% for
Project 2
Resident tmpNPV
Group By PrjId;

Drop table tmpNPV;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- PrjId
- NPV

结果表

PrjId	NPV
1	$2486.85
2	$2042.12

项目 ID 1 预计在三个周期结束时收到 1000 美元的付款，每个周期的贴现率为 10%。因此，有效净现值等于 2486.85 美元。

项目 ID 2 预计在四个时期内以 15% 的贴现率支付两次 500 美元的付款和另外两次 1000 美元的付款。因此，有效净现值等于 2042.12 美元。

示例 4 – 项目盈利能力示例(脚本)

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 两个项目的贴现率和初始投资(期间 0)，加载到名为 Project 的表中。
- 按项目 ID 和期间 ID 列出每个项目的多个期间的现金流。如果数据未排序，则可以使用此期间 ID 对记录进行排序。
- NoConcatenate、常驻加载和 Left Join 函数的组合用于创建临时表 tmpNPV。该表将 Project 和 CashFlow 表的记录合并为一个平面表。此表将重复每个期间的贴现率。
- 与每个项目相关的单值贴现率，使用 only() 函数检索，并用于每个项目的净现值计算。
- tmpNPV 表中的常驻负载用于计算名为 NPV 的表中每个项目的 NPV 字段。

- 创建一个额外字段，将净现值除以每个项目的初始投资，以计算项目盈利能力指数。
- 按项目 ID 分组的 group-by 语句用于对每个项目的所有付款进行分组。

为了避免将任何合成或冗余数据加载到数据模型中，将在脚本末尾删除 tmpNPV 表。

加载脚本

```
Project:
Load * inline [
PrjId,Discount_Rate, Initial_Investment
1,0.1,100000
2,0.15,100000
];

CashFlow:
Load
*
Inline
[
PrjId,PeriodId,Values,
1,1,35000
1,2,35000
1,3,35000
2,1,30000
2,2,40000
2,3,50000
2,4,60000
];

tmpNPV:
NoConcatenate Load *
Resident Project;
Left Join
Load *
Resident CashFlow;

NPV:
Load
        PrjId,
        NPV(Only(Discount_Rate),Values) as NPV, //Discount Rate will be 10% for Project 1 and
15% for Project 2
    NPV(Only(Discount_Rate),Values)/ Only(Initial_Investment) as Profitability_Index
Resident tmpNPV
Group By PrjId;

Drop table tmpNPV;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- PrjId
- NPV

创建以下度量：

```
=only(Profitability_Index)
```

结果表

PrjId	NPV	=only(Profitability_Index)
1	$87039.82	0.87
2	$123513.71	1.24

项目 ID 1 的有效净现值为 87039.82 美元，初始投资为 100000 美元。因此，盈利能力指数等于 0.87。因为它小于 1，所以该项目不盈利。

项目 ID 2 的有效净现值为 123513.71 美元，初始投资为 100000 美元。因此，盈利能力指数等于 1.24。因为它大于 1，所以项目是有利可图的。

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 默认情况下，结果将被格式化为货币。

参数：

参数

参数	说明
discount_rate	**discount_rate** 是应用的折扣百分比。
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 375)*
- *Aggr - 图表函数 (page 517)*

XIRR

XIRR() 函数用于返回聚合内部回报率(每年)，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时间表(不必为周期性的)。所有付款按 365 天一年年折扣。

Qlik 的 XIRR 函数(**XIRR()** 和 **RangeXIRR()** 函数)使用以下方程来求解 `Rate` 值，以确定正确的 XIRR 值：

```
XNPV(Rate, pmt, date) = 0
```

这个方程是用简化版的牛顿法求解的。

语法：

```
XIRR(pmt, date )
```

返回数据类型：数字

参数

参数	描述
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。

使用此功能时，以下限制适用：

- 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。
- 此函数需要至少一个有效的负付款和至少一个无效的正付款(具有相应的有效日期)。如果没有提供这些付款，则会返回 NULL 值。

这些主题可以帮助您使用此函数：

- *XNPV (page 369)*：使用此函数可以计算现金流表的聚合净现值。
- *RangeXIRR (page 1310)*：**RangeXIRR()** 是 **XIRR()** 函数的等效范围函数。

在 Qlik Sense 客户端托管 的不同版本中，此函数使用的底层算法存在差异。有关算法最近更新的更多信息，请参阅支持文章，请参阅支持文章 XIRR 函数修复和更新。

示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 一系列现金流的交易数据。
- 使用 **XIRR()** 函数来计算这些现金流的内部年回报率。

加载脚本

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Payments
2013-01-01|-10000
2013-03-01|3000
2013-10-30|4200
2014-02-01|6800
] (delimiter is '|');
```

```
Cashflow1:
LOAD Year,XIRR(Payments, Date) as XIRR2013  Resident Cashflow Group By Year;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- Year
- XIRR2013

结果表

年	XIRR2013
2013	0.5385

解释 XIRR 返回值

XIRR 功能通常用于分析一项投资，其中一开始有一笔向外(负)付款，然后是一系列较小的收入(正)付款。以下是一个简化的例子，只有一个负付款和一个正付款：

```
Cashflow:
LOAD * inline [
Date|Payments
2023-01-01|-100
2024-01-01|110
] (delimiter is '|');
```

我们首付 100，一年后收回 110。这意味着每年 10% 的回报率。`XIRR(Payments, Date)` 返回值 0.1。

XIRR 功能的返回值可以为正，也可以为负。对于投资而言，为负的结果表明投资是亏损的。收益或损失的金额可以直接通过在支付字段上进行求和聚合来计算。

在上面的例子中，我们将贷款一年。回报率可以被认为是利息。当你是交易的另一方时(例如，如果你是借款人而不是贷款人)，也可以使用 XIRR 的功能。

考虑以下示例：

```
Cashflow:
LOAD * inline [
Date|Payments
2023-01-01|100
2024-01-01|-110
] (delimiter is '|');
```

这与第一个示例相同，但进行了反向处理。在这里，我们借入 100，为期一年，并以 10% 的利息偿还。在这个例子中，XIRR 计算返回 0.1 (10%)，与第一个例子的值相同。

请注意，在第一个示例中，我们获得了 10 的利润，在第二个示例中我们遇到了 10 的损失，但对于这两个示例，XIRR 功能的返回值都为正。这是因为 XIRR 功能计算交易中隐藏的利息，而不考虑您在交易中处于哪一方。

对于多个解的限制

Qlik 的 XIRR 函数由以下等式定义，在其中求解 `Rate` 值：

```
XNPV(Rate, pmt, date) = 0
```

这个方程有时可能有不止一个解。这被称为“多个 IRR 问题”，由非正常现金流（也称为非常规现金流）引起。以下加载脚本显示了与此相关的一个示例：

```
Cashflow:
LOAD * inline [
Date|Payments
2021-01-01|-200
2022-01-01|500
2023-01-01|-250
] (delimiter is '|');
```

在这个例子中，有一个负解和一个正解（Rate = -0.3 和 Rate = 0.8）。**XIRR()** 将返回 0.8。

当的 Qlik 的 XIRR 功能搜索解时，它从 Rate = 0 开始，并逐步增加速率，直至找到解。如果有一个以上的正解，它将返回遇到的第一个解。如果它找不到正解，它会将 Rate 重置为零，并开始在负方向上寻找解。

请注意保证“正常”现金流，以便只有一个解。“正常”现金流意味着所有具有相同符号（正或负）的付款都在一个连续组中。

另请参见：

- *XNPV (page 369)*
- *RangeXIRR (page 1310)*
- XIRR 函数修复和更新

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（不必为周期性的）时间表（即不一定是周期性的）。所有付款按 365 天一年年折扣。

Qlik 的 XIRR 函数（**XIRR()** 和 **RangeXIRR()** 函数）使用以下方程来求解 Rate 值，以确定正确的 XIRR 值：

```
XNPV(Rate, pmt, date) = 0
```

这个方程是用简化版的牛顿法求解的。

语法：

```
XIRR([TOTAL [<fld {,fld}>]] pmt, date)
```

返回数据类型：数字

参数

参数	描述
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。

参数	描述
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。

使用此功能时，以下限制适用：

- **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。
- 此函数需要至少一个有效的负付款和至少一个无效的正付款（具有相应的有效日期）。如果没有提供这些付款，则会返回 NULL 值。

这些主题可以帮助您使用此函数：

- *XNPV - 图表函数 (page 375)*：使用此函数可以计算现金流表的聚合净现值。
- *RangeXIRR (page 1310)*：**RangeXIRR()** 是 **XIRR()** 函数的等效范围函数。

在 Qlik Sense 客户端托管 的不同版本中，此函数使用的底层算法存在差异。有关算法最近更新的更多信息，请参阅支持文章，请参阅支持文章 XIRR 函数修复和更新。

示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含现金流交易的数据集。
- 存储在名为 `Cashflow` 的表中的信息。

加载脚本

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Payments
2013-01-01|-10000
2013-03-01|3000
2013-10-30|4200
2014-02-01|6800
] (delimiter is '|');
```

结果

执行以下操作：

加载数据并打开工作表。创建新表并添加以下计算作为度量：

```
=XIRR(Payments, Date)
```

结果表

=XIRR(Payments, Date)
0.5385

解释 XIRR 返回值

XIRR 功能通常用于分析一项投资，其中一开始有一笔向外（负）付款，然后是一系列较小的收入（正）付款。以下是一个简化的例子，只有一个负付款和一个正付款：

```
Cashflow:
LOAD * inline [
Date|Payments
2023-01-01|-100
2024-01-01|110
] (delimiter is '|');
```

我们首付 100，一年后收回 110。这意味着每年 10% 的回报率。`XIRR(Payments, Date)` 返回值 0.1。

XIRR 功能的返回值可以为正，也可以为负。对于投资而言，为负的结果表明投资是亏损的。收益或损失的金额可以直接通过在支付字段上进行求和聚合来计算。

在上面的例子中，我们将贷款一年。回报率可以被认为是利息。当你是交易的另一方时（例如，如果你是借款人而不是贷款人），也可以使用 XIRR 的功能。

考虑以下示例：

```
Cashflow:
LOAD * inline [
Date|Payments
2023-01-01|100
2024-01-01|-110
] (delimiter is '|');
```

这与第一个示例相同，但进行了反向处理。在这里，我们借入 100，为期一年，并以 10% 的利息偿还。在这个例子中，XIRR 计算返回 0.1 (10%)，与第一个例子的值相同。

请注意，在第一个示例中，我们获得了 10 的利润，在第二个示例中我们遇到了 10 的损失，但对于这两个示例，XIRR 功能的返回值都为正。这是因为 XIRR 功能计算交易中隐藏的利息，而不考虑您在交易中处于哪一方。

对于多个解的限制

Qlik 的 XIRR 函数由以下等式定义，在其中求解 `Rate` 值：

```
XNPV(Rate, pmt, date) = 0
```

这个方程有时可能有不止一个解。这被称为“多个 IRR 问题”，由非正常现金流（也称为非常规现金流）引起。以下加载脚本显示了与此相关的一个示例：

```
Cashflow:
LOAD * inline [
Date|Payments
2021-01-01|-200
2022-01-01|500
2023-01-01|-250
] (delimiter is '|');
```

在这个例子中，有一个负解和一个正解（Rate = -0.3 和 Rate = 0.8）。**XIRR()** 将返回 0.8。

当的 Qlik 的 XIRR 功能搜索解时，它从 Rate = 0 开始，并逐步增加速率，直至找到解。如果有一个以上的正解，它将返回遇到的第一个解。如果它找不到正解，它会将 Rate 重置为零，并开始在负方向上寻找解。

请注意保证“正常”现金流，以便只有一个解。“正常”现金流意味着所有具有相同符号（正或负）的付款都在一个连续组中。

另请参见：

- *IRR - 图表函数 (page 354)*
- *Aggr - 图表函数 (page 517)*
- XIRR 函数修复和更新

XNPV

XNPV() 函数用于返回聚合净现值，以揭示 **pmt** 和 **date** 表达式的成对数值表示的现金流时间表（不必为周期性的）。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

> *默认情况下，结果将被格式化为货币。*

计算 XNPV 的公式如下所示：

XNPV 聚合公式

$$XNPV = \sum_{i=1}^{n} \frac{P_i}{(1+rate)^{(di-d1)/365}}$$

其中：

- P_i = 单期净现金流入流出 i
- d_1 = 第一次付款日期
- d_i = 第 i 次付款日期
- rate = 贴现率

净现值 (NPV) 用于在给定折扣率的情况下计算未来现金流的当前总价值。要计算 XNPV，我们需要估计具有相应日期的未来现金流。在此之后，对于每次付款，我们将根据付款日期采用复合折扣率。

对一系列付款执行 XNPV 聚合类似于对这些付款执行求和聚合。不同之处在于，每个金额都会根据所选的折扣率（类似于利率）和未来付款的时间进行修改（或“打折扣”）。在 **discount_rate** 参数设置为零的情况下执行 XNPV 将使 XNPV 等效于求和操作（在求和之前不会修改付款）。一般而言，**discount_rate** 设置得越接近零，XNPV 结果就越类似于求和聚合的结果。

参数

参数	说明
discount_rate	**discount_rate** 是付款折扣时依据的年费率。 值 0.1 表示 10% 的贴现率。
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。正值被假定为流入，负值被假定为流出。 ***XNPV()*** *不会对初始现金流进行折扣，因为它总是在开始日期发生。后续付款按 365 天的年折扣。这与第一笔付款也打折的* ***NPV()*** *不同。*
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。第一个值用作计算未来现金流抵销的开始日期。

使用此功能时，以下限制适用：

- 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。

适用场景

- XNPV() 用于计算投资机会的净现值 (NPV)。
- 由于其较高的精度，XNPV 比 NPV 更适合所有类型的金融模型。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 一次性付款(脚本)

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 一个项目及其一年现金流的数据集，位于名为 CashFlow 的表中。计算的初始日期为 2022 年 7 月 1 日，净现金流为 0。一年后，现金流为 1000 美元。
- CashFlow 表中的常驻荷载，用于计算名为 XNPV 的表中项目的 XNPV 字段。
- 硬编码贴现率为 10% (0.1)，用于 XNPV 计算。
- 用于对项目的所有付款进行分组的 Group By 报告。

加载脚本

```
CashFlow:
Load
*
Inline
[
PrjId,Dates,Values
1,'07/01/2022',0
1,'07/01/2023',1000
];

XNPV:
Load
       PrjId,
       XNPV(0.1,Values,Dates) as XNPV //Discount Rate of 10%
Resident CashFlow
Group By PrjId;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- PrjId
- XNPV

结果表

PrjId	XNPV
1	$909.09

根据公式，第一条记录的 XNPV 值为 0，第二条记录的 XNPV 值是 909.09 美元。因此，总 XNPV 是 909.09 美元。

示例 2 – 多次付款(脚本)

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 一个项目及其一年现金流的数据集，位于名为 CashFlow 的表中。
- CashFlow 表中的常驻荷载，用于计算名为 XNPV 的表中项目的 XNPV 字段。
- 硬编码贴现率为 10% (0.1)，用于 XNPV 计算。
- 用于对项目的所有付款进行分组的 Group By 报告。

加载脚本

```
CashFlow:
Load
*
Inline
[
PrjId,Dates,Values
1,'07/01/2022',0
1,'07/01/2024',500
1,'07/01/2023',1000
];

XNPV:
Load
       PrjId,
       XNPV(0.1,Values,Dates) as XNPV //Discount Rate of 10%
Resident CashFlow
Group By PrjId;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- PrjId
- XNPV

结果表

PrjId	XNPV
1	$1322.21

在本例中，第一年年底收到 1000 美元的付款，第二年年底收到 500 美元的付款。如果每期贴现率为 10%，有效 XNPV 等于 1322.21 美元。

请注意，只有第一行数据应引用计算基准日期。对于其余的行，顺序并不重要，因为日期参数用于计算经过的期间。

示例 3 – 多次付款和不规则现金流(脚本)

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名为 Project 的表中两个项目的折扣率。
- 按项目 ID 和日期列出每个项目的多个期间的现金流。Dates 字段用于计算贴现率应用于现金流的持续时间。除了第一条记录(初始现金流和日期)外，记录的顺序并不重要，更改它不应影响计算。
- 使用 NoConcatenate、Resident Load 和 Left Join 函数的组合，将创建一个临时表 tmpNPV，将 Project 和 CashFlow 表的记录组合在一个平面表中。此表将重复每个现金流的贴现率。
- tmpNPV 表中的常驻荷载，用于计算名为 XNPV 的表中每个项目的 XNPV 字段。
- 使用 only() 函数获取与每个项目相关的单值贴现率，并用于每个项目的 XNPV 计算。
- 按项目 ID 分组的 Group By 语句用于对每个项目的所有付款和相应日期进行分组。
- 为了避免将任何合成或冗余数据加载到数据模型中，将在脚本末尾删除 tmpXNPV 表。

加载脚本

```
Project:
Load * inline [
PrjId,Discount_Rate
1,0.1
2,0.15
];

CashFlow:
Load
*
Inline
[
PrjId,Dates,Values
1,'07/01/2021',0
1,'07/01/2022',1000
1,'07/01/2023',1000
```

```
2,'07/01/2020',0
2,'07/01/2023',500
2,'07/01/2024',1000
2,'07/01/2022',500
];

tmpXNPV:
NoConcatenate Load *
Resident Project;
Left Join
Load *
Resident CashFlow;

XNPV:
Load
       PrjId,
       XNPV(Only(Discount_Rate),Values,Dates) as XNPV //Discount Rate will be 10% for Project 1 and
15% for Project 2
Resident tmpXNPV
Group By PrjId;

Drop table tmpXNPV;
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- PrjId
- XNPV

结果表

PrjId	XNPV
1	$1735.54
2	$278.36

2021 年 7 月 1 日，项目 ID 1 的初始现金流为 0 美元。在随后的两年结束时，将收到两笔 1000美 元的付款，每个期间的贴现率为 10%。因此，有效 XNPV 等于 1735.54 美元。

项目 ID 2 在 2020 年 7 月 1 日的初始流出量为 1000 美元（因此为负号）。两年后，预计将支付 500 美元。3 年后，预计将再支付 500 美元。最后，2024 年 7 月 1 日，预计将支付 1000 美元。贴现率为 15% 时，有效 XNPV 等于 278.36 美元。

另请参见：

- *Drop table (page 144)*
- *group by (page 153)*
- *Join (page 69)*
- *Max (page 321)*
- *NoConcatenate (page 86)*
- *NPV - 图表函数 (page 362)*

Only (page 330)

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

> *默认情况下,结果将被格式化为货币。*

计算 XNPV 的公式如下所示:

XNPV 聚合公式

$$XNPV = \sum_{i=1}^{n} \frac{P_i}{(1+rate)^{(di-d1)/365}}$$

其中:

- P_i = 单期净现金流入流出 i
- d_1 = 第一次付款日期
- d_i = 第 i 次付款日期
- rate = 贴现率

净现值 (NPV) 用于在给定折扣率的情况下计算未来现金流的当前总价值。要计算 XNPV,我们需要估计具有相应日期的未来现金流。在此之后,对于每次付款,我们将根据付款日期采用复合折扣率。

对一系列付款执行 XNPV 聚合类似于对这些付款执行求和聚合。不同之处在于,每个金额都会根据所选的折扣率(类似于利率)和未来付款的时间进行修改(或"打折扣")。在 **discount_rate** 参数设置为零的情况下执行 XNPV 将使 XNPV 等效于求和操作(在求和之前不会修改付款)。一般而言,**discount_rate** 设置得越接近零,XNPV 结果就越类似于求和聚合的结果。

参数

参数	说明
discount_rate	**discount_rate** 是付款折扣时依据的年费率。 值 0.1 表示 10% 的贴现率。

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。正值被假定为流入，负值被假定为流出。 ***XNPV()*** *不会对初始现金流进行折扣，因为它总是在开始日期发生。后续付款按 365 天的年折扣。这与第一笔付款也打折的* ***NPV()*** *不同。*
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。第一个值用作计算未来现金流抵销的开始日期。
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。

使用此功能时，以下限制适用：

- **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。

适用场景

- XNPV() 用于计算投资机会的净现值 (NPV)。
- 由于其较高的精度，XNPV 比 NPV 更适合所有类型的金融模型。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含现金流交易的数据集。
- 存储在名为 Cashflow 的表中的信息。

加载脚本

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Payments
2013-01-01|-10000
2013-03-01|3000
2013-10-30|4200
2014-02-01|6800
] (delimiter is '|');
```

结果

执行以下操作：

加载数据并打开工作表。创建新表并添加以下计算作为度量：

```
=XNPV(0.09, Payments, Date)
```

结果表

=XNPV(0.09, Payments, Date)
$3062.49

另请参见：

NPV - 图表函数 (page 362)
Aggr - 图表函数 (page 517)

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({[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{[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。

示例	结果
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 517)

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 Correl() 计算的结果将显示：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` 和度量 `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 517)*
- *Avg - 图表函数 (page 383)*
- *RangeCorrel (page 1279)*

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 392) 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 rank 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 (Type, Value) 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 Fractile() 计算的结果为： 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 394) 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 rank 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 517)

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 389) 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 rank 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 (Type, Value) 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 FractileExc() 计算的结果为： 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 390) 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 rank 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 517)

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 (Type, Value) 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 Kurtosis() 计算的结果为： 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 (Type, Value)
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 383)

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。

参数	说明
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 437)

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。
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。

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 437)
Avg - 图表函数 (page 383)

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 437)

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。
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 437)*
- *Avg - 图表函数 (page 383)*

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 437)*

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。
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 437)
Avg - 图表函数 (page 383)

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 437)

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 437)*
- *Avg - 图表函数 (page 383)*

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 437)

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 437)*
- *Avg - 图表函数 (page 383)*

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 437)

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 437)*
- *Avg - 图表函数 (page 383)*

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 437)

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 437)
Avg - 图表函数 (page 383)

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 437)

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 437)*
- *Avg - 图表函数 (page 383)*

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 437)*

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 437)
Avg - 图表函数 (page 383)

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 437)

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 437)*
- *Avg - 图表函数 (page 383)*

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 Group By Letter;
```

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

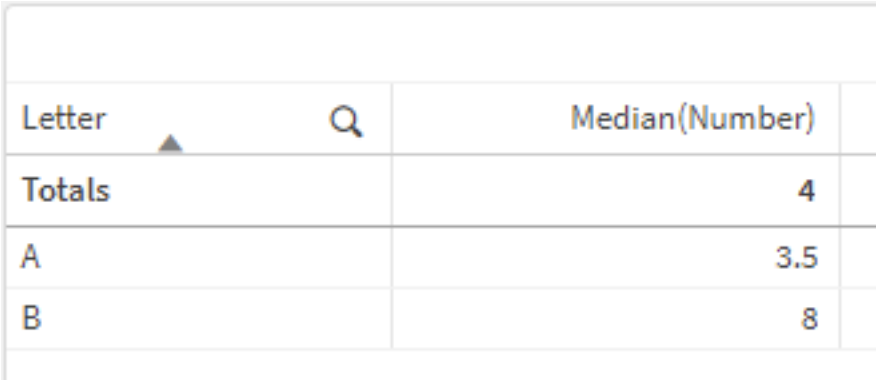
在 Qlik Sense 工作表中创建表可视化，以 **Letter** 为维度。

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 383)

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。
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 (Type, Value) 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。
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 (Type, Value) 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 383)

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 (Type, Value) 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。

参数	说明
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 (Type, Value)`
`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 383)*
- *STEYX - 图表函数 (page 435)*

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 (Type, Value) 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 Sterr() 计算的结果为： 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。

参数	说明
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 (Type, Value)`
`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 383)*
- *STEYX - 图表函数 (page 435)*

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 STEYX() 计算的结果为 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	描述
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 383)*
- *Sterr - 图表函数 (page 432)*

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 Cloud中可用的 linest 函数值来创建可视化内容。Qlik Senselinest 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

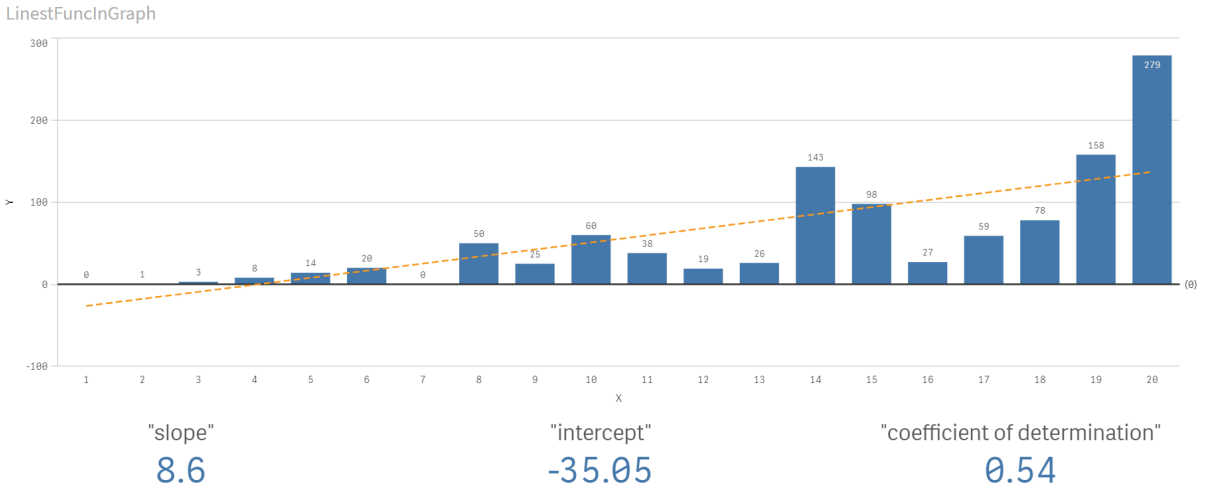
Linest functions	Linest function results
Linest_b	-35.047
Linest_df	18
Linest_f	20.788
Linest_m	8.605
Linest_r2	0.536
Linest_SEB	22.607
Linest_SEM	1.887
Linest_SEY	48.666
Linest_SSREG	49235.014
Linest_SSRESID	42631.186

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() 用于返回一个或两个值系列的聚合卡方检验值。

```
Chi2Test_chi2(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(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 445)*
- *Z 检验函数 (page 476)*

Chi2Test_chi2

Chi2Test_chi2() 用于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 489)*
- *如何在数据加载脚本中使用 chi2-test 函数的示例 (page 493)*

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 489)*
- *如何在数据加载脚本中使用 chi2-test 函数的示例 (page 493)*

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 489)*
- *如何在数据加载脚本中使用 chi2-test 函数的示例 (page 493)*

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)*

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 495)*

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 495)

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 495)

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 495)

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 495)

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 495)*

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 495)

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 495)

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 495)

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 495)

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 495)*

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 495)

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 495)

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 495)

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 495)

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 495)

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 495)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 499)

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 以转到应用视图，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**以编辑表格。
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。

   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.
   ```

2.

   ```
   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

独立的样本测试

Type	conf	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval (Lower)	95% Confidence Interval (Upper)
Equal Variance not Assumed	0	3.534	37.116717335823	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	8.706939	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. 创建具有新工作表的新应用。
2. 在数据加载编辑器中输入以下内容:

```
Table1:
Crosstable (Type, Value)
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 以转到应用视图, 然后单击您以前创建的表格。
   此项将打开工作表视图。
2. 单击 **编辑工作表**以编辑表格。
3. 在**图表**中添加表格, 在**字段**中向表格添加 Type 作为维度。

4. 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)

5. 单击**排序**，并确保 Type 在排序列表顶部。

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 Independent sample test 表格

执行以下操作：

1. 单击 **编辑工作表**以编辑表格。
2. 从**图表**中添加一个具有以下表达式的表作为表的维度。`=ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1))` 并给它提供“类型”标签。

3. 添加以下表达式作为度量：

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 (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 (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	conf	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval (Lower)	95% Confidence Interval (Upper)
Equal Variance not Assumed	0	3.534	37.116717335823	0.001	15.2	4.30101	6.48625	23.9137

Type	conf	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval (Lower)	95% Confidence Interval (Upper)
Equal Variance Assumed	8.706939	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 Cloud中可用的 z-test 函数值来创建可视化内容。Qlik Sense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. 创建具有新工作表的新应用。

   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 (Type, Value)
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 表格

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，然后单击您上面创建的表格。
   随即打开工作表视图。
2. 单击 **编辑工作表**以编辑表格。
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：

z-test 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
Observation	5.48	27.15	0.000000	2.80	9.71

创建 z-testw 表格

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。

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，然后单击您上面创建的表格。随即打开工作表视图。
2. 单击 **编辑工作表**以编辑表格。
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
ZTestw Conf	ZTestw_conf(2,Value)
ZTestw Dif	ZTestw_dif(2,Value)
ZTestw Sig	ZTestw_sig(2,Value)
ZTestw Sterr	ZTestw_sterr(2,Value)
ZTestw Z	ZTestw_z(2,Value)

使用与 z-test 函数示例中相同的数字格式。

结果：

z-testw 函数结果表格中将包含以下值：

z-testw 结果表

Type	ZTestw Conf	ZTestw Dif	ZTestw Sig	ZTestw Sterr	ZTestw Z
Comparison	4.47	11.95	8.037185e-08	2.28	5.24
Observation	3.83	27.15	0	1.95	13.91

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

示例	结果	结果一次添加至工作表
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。

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：

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

示例	结果	
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 517)

8.3 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 Aggr 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

Year	Month	Sum(Sales)	Structured Aggr()
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**，排序为 (1) **Year**（数字）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 315)

8.4 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。

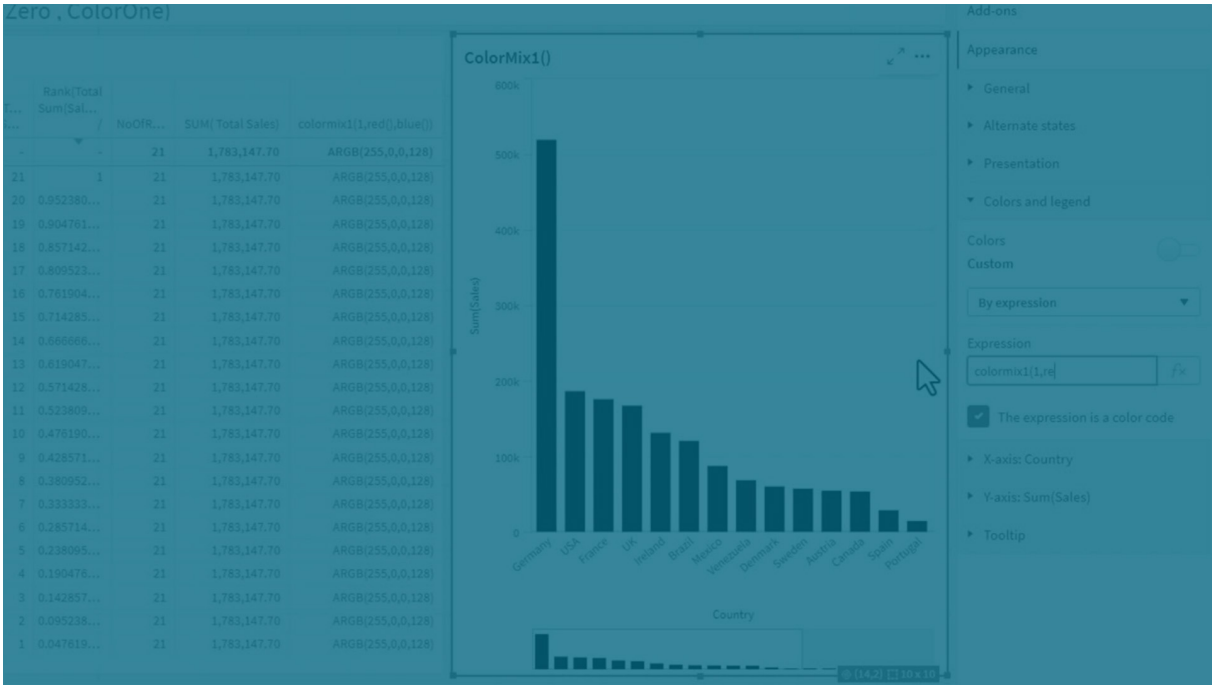
ColorOne 是指与时间间隔高端相关联的颜色的有效 RGB 颜色呈现形式。

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	描述
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：

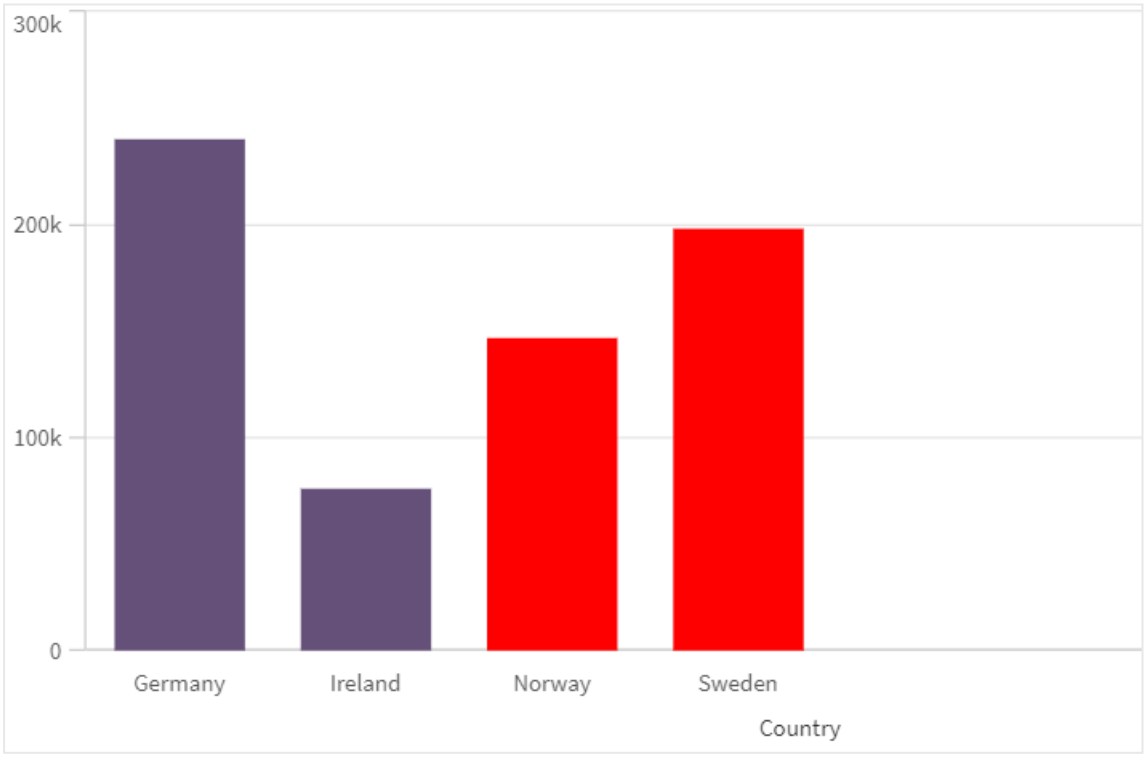
示例中所使用的数据：

```
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。

8.5 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 函数将第一个参数与以下所有参数进行比较，并返回匹配表达式的数字位置。该比较不区分大小写，对日语平假名和片假名字符系统不会进行区分。

```
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符(***** 和 **?**)。***** 匹配任何字符序列。**?** 匹配任何单个字符。该比较不区分大小写，对日语平假名和片假名字符系统不会进行区分。

```
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。

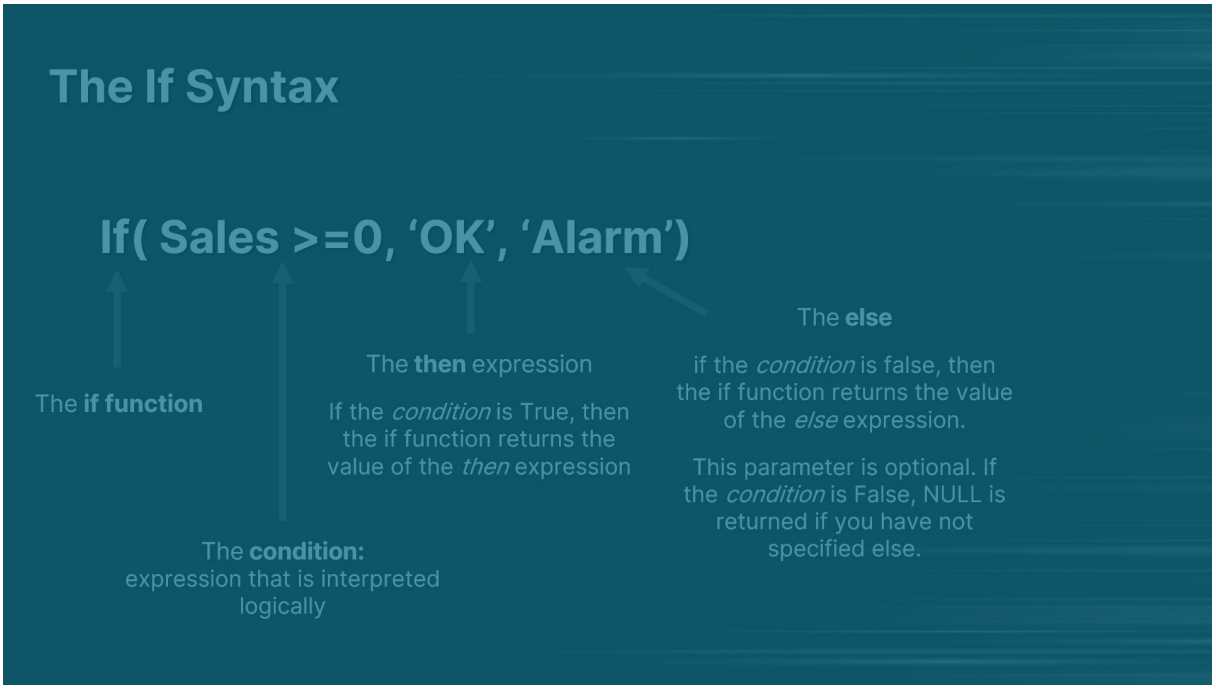
示例：

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”。
`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])
```

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
Compare transaction_amount field from Transaction table to threshold of 100.
Output results into a new field called Compared to Threshold
 */

Transaction_Buckets:
Load
   transaction_id,
   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

transaction_id	与阈值比较
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	Warning

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

Cities	match(Cities,'Toronto','Boston','Beijing','Zurich')	match(Cities,'Toronto','Boston','Beijing','Stockholm','zurich')
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
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 函数将第一个参数与以下所有参数进行比较，并返回匹配表达式的数字位置。该比较不区分大小写，对日语平假名和片假名字符系统不会进行区分。

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符(***** 和 **?**)。***** 匹配任何字符序列。**?** 匹配任何单个字符。该比较不区分大小写，对日语平假名和片假名字符系统不会进行区分。

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

8.6 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钥，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
North	2014	June	127	199

Region	Year	Month	Sum([Sales])	Sum([Costs])
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钥，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
North	2014	June	127	199

Region	Year	Month	Sum([Sales])	Sum([Costs])
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钥，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 **LOAD** 语句中使用 **RowNo( )**。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

当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 1222)

8.7 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环境变量周数所属的年份。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
此函数用于返回当前时间的时间戳。函数以 **TimeStamp** 系统变量格式返回值。默认 **timer_mode** 值为 1。

```
now ([ timer_mode])
```

today
此函数用于返回当前日期。函数以 DateFormat 系统变量格式返回值。

```
today ([timer_mode])
```

LocalTime
此函数用于返回指定时区的当前时间戳。

```
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
此函数返回根据年、周数和星期几计算的日期。

```
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

此函数返回在 Qlik 引擎运行的计算机上定义的时区。

```
timezone ( )
```

GMT

此函数用于返回来自地区设置的当前 Greenwich Mean Time。

```
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

此函数用于查找时间戳是否在作为基准日期的同一个月、两个月、季度、四个月期间或半年内。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。

```
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、四个月期间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 中的农历周定义为将 1 月 1 日计算为一周的第一天。除了一年中的最后一周外，每周都会有七天。

```
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 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

```
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、四个月期间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。默认输出格式是在脚本中设置的 **DateFormat**。

```
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、四个月期间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、四个月期间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

此函数返回一个值，该值对应于包含 **date** 的日历周的最后一天的最后一毫秒的时间戳。默认的输出格式为在脚本中所设置的 **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** 的阴历周第一天的第一毫秒的时间戳对应的值。Qlik Sense 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

```
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 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

```
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 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

```
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。创建主日历时使用此功能。

```
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：

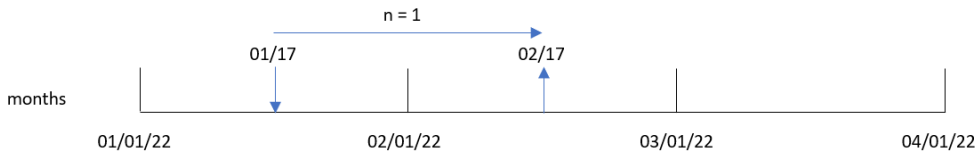
```
AddMonths(startdate, n , [ , mode])
```

返回数据类型：双

addmonths() 函数从 startdate 中加上或减去定义的月数 n，并返回结果日期。

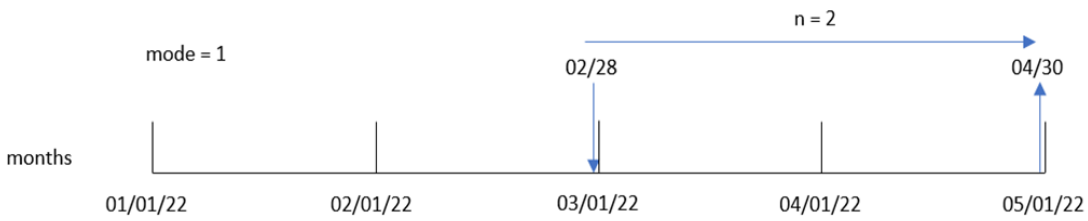
mode 参数将影响当月 28 日当天或之后的 startdate 值。通过将 mode 参数设置为 1，addmonths() 函数将返回一个与月末的相对距离相等的日期作为 startdate。

addmonths() 函数的示例图表



例如，2 月 28 日是一个月的最后一天。如果 addmonths() 函数（其中 mode 为 1）用于在两个月后返回日期，则函数将返回最后一个日期 4 月 30 日。

addmonths() 函数的示例图表，具有 mode=1



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。

适用场景

addmonths() 函数通常用于表达式中，以查找一段时间之前或之后的给定月数的日期。

例如，addmonths() 函数可用于识别手机合同的结束日期。

函数示例

示例	结果
addmonths ('01/29/2003' ,3)	返回 '04/29/2003'。
addmonths ('01/29/2003',3,0)	返回 '04/29/2003'。
addmonths ('01/29/2003',3,1)	返回 '04/28/2003'。
addmonths ('01/29/2003',1,0)	返回 '02/28/2003'。
addmonths ('01/29/2003',1,1)	返回 '02/26/2003'。
addmonths ('02/28/2003',1,0)	返回 '03/28/2003'。
addmonths ('02/28/2003',1,1)	返回 '03/31/2003'。
addmonths ('01/29/2003',-3)	返回 '10/29/2002'。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 two_months_later，返回交易发生后两个月的日期。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        addmonths(date,2) as two_months_later
        ;
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

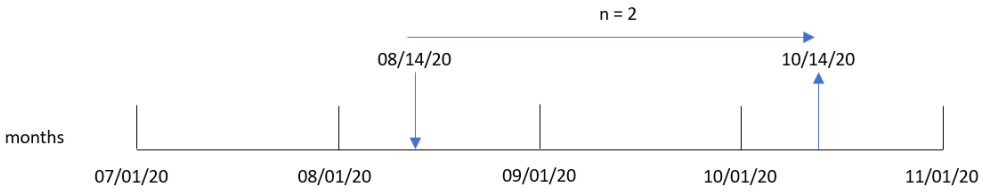
- date
- two_months_later

结果表

日期	two_months_later
01/10/2020	03/10/2020
02/28/2020	04/28/2020
04/09/2020	06/09/2020
04/16/2020	06/16/2020
05/21/2020	07/21/2020
08/14/2020	10/14/2020
10/07/2020	12/07/2020
12/05/2020	02/05/2021
01/22/2021	03/22/2021
02/03/2021	04/03/2021
03/17/2021	05/17/2021
04/23/2021	06/23/2021
05/04/2021	07/04/2021
06/30/2021	08/30/2021
07/26/2021	09/26/2021
12/27/2021	02/27/2022
02/02/2022	04/02/2022
02/26/2022	04/26/2022
03/07/2022	05/07/2022
03/11/2022	05/11/2022

two_months_later 字段是在前置 Load 语句中使用 addmonths() 函数创建的。提供的第一个参数标识正在评估的日期。第二个参数是要从 startdate 中加减的月数。在本例中，提供的值为 2。

addmonths() 函数图表，示例没有额外参数



交易 8193 发生在 8 月 14 日。因此，对于 two_months_later 字段，addmonths() 函数返回 October 14, 2020。

示例 2 – 相对月末

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年月末交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 relative_two_months_prior，返回交易发生前两个月的相对月末日期。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        addmonths(date,-2,1) as relative_two_months_prior
        ;
Load
*
Inline
[
id,date,amount
8188,'01/28/2022',37.23
8189,'01/31/2022',57.54
8190,'02/28/2022',17.17
8191,'04/29/2022',88.27
8192,'04/30/2022',57.42
8193,'05/31/2022',53.80
8194,'08/14/2022',82.06
8195,'10/07/2022',40.39
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

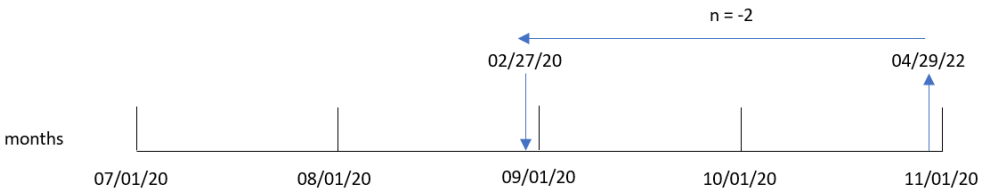
- date
- relative_two_months_prior

结果表

日期	relative_two_months_prior
01/28/2022	11/27/2021
01/31/2022	11/30/2021
02/28/2022	12/31/2021
04/29/2022	02/27/2022
04/30/2022	02/28/2022
05/31/2022	03/31/2022
08/14/2022	06/14/2022
10/07/2022	08/07/2022

relative_two_months_prior 字段是在前置 Load 语句中使用 addmonths() 函数创建的。提供的第一个参数标识正在评估的日期。第二个参数是要从 startdate 中加减的月数。在本例中，提供的值为 2。最后一个参数是 mode，值为 1，它强制函数计算所有大于或等于 28 的日期的相对月末日期。

addmonths() 函数的图表，示例具有 n=-2



交易 8191 发生在 2022 年 4 月 29 日。最初，两个月前会将该月设置为 2 月。然后，由于函数的第三个参数将 mode 设置为 1，并且日值晚于 27 号，因此函数计算相对月末值。该函数确定 29 号是 4 月的第二个最后一天，因此返回 2 月的第 2 个最后一日，即 27 号。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生后两个月的日期的计算在图表对象中创建为度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：`date`。

创建以下度量：

```
=addmonths(date,2)
```

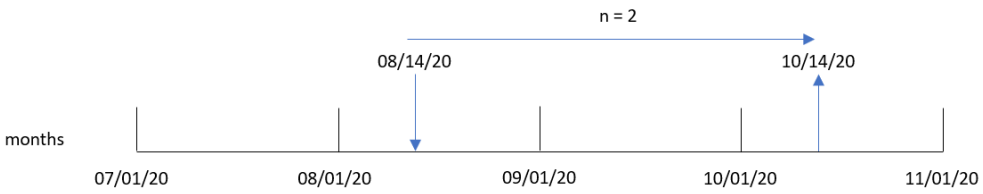
结果表

日期	=addmonths(date,2)
01/10/2020	03/10/2020
02/28/2020	04/28/2020
04/09/2020	06/09/2020

日期	=addmonths(date,2)
04/16/2020	06/16/2020
05/21/2020	07/21/2020
08/14/2020	10/14/2020
10/07/2020	12/07/2020
12/05/2020	02/05/2021
01/22/2021	03/22/2021
02/03/2021	04/03/2021
03/17/2021	05/17/2021
04/23/2021	06/23/2021
05/04/2021	07/04/2021
06/30/2021	08/30/2021
07/26/2021	09/26/2021
12/27/2021	02/27/2022
02/02/2022	04/02/2022
02/26/2022	04/26/2022
03/07/2022	05/07/2022
03/11/2022	05/11/2022

通过使用 addmonths() 函数在图表对象中创建 two_months_later 度量。提供的第一个参数标识正在评估的日期。第二个参数是要从 startdate 中加减的月数。在本例中，提供的值为 2。

addmonths() 函数的图表，图表对象示例



交易 8193 发生在 8 月 14 日。因此，对于 two_months_later 字段，addmonths() 函数返回 October 14, 2020。

示例 4 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Mobile_Plans 的表中的数据集。
- 包含合同 ID、开始日期、合同长度和月费的信息。

最终用户想要一个图表对象，该对象按合同 ID 显示每个电话合同的终止日期。

加载脚本

```
Mobile_Plans:
Load
*
Inline
[
contract_id,start_date,contract_length,monthly_fee
8188,'01/13/2020',18,37.23
8189,'02/26/2020',24,17.17
8190,'03/27/2020',36,88.27
8191,'04/16/2020',24,57.42
8192,'05/21/2020',24,53.80
8193,'08/14/2020',12,82.06
8194,'10/07/2020',18,40.39
8195,'12/05/2020',12,87.21
8196,'01/22/2021',12,95.93
8197,'02/03/2021',18,45.89
8198,'03/17/2021',24,36.23
8199,'04/23/2021',24,25.66
8200,'05/04/2021',12,82.77
8201,'06/30/2021',12,69.98
8202,'07/26/2021',12,76.11
8203,'12/27/2021',36,25.12
8204,'06/06/2022',24,46.23
8205,'07/18/2022',12,84.21
8206,'11/14/2022',12,96.24
8207,'12/12/2022',18,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- contract_id
- start_date

- contract_length

创建以下度量以计算每个合同的结束日期：

```
=addmonths(start_date,contract_length, 0)
```

结果表

contract_id	start_date	contract_length	=addmonths(start_date,contract_length,0)
8188	01/13/2020	18	07/13/2021
8189	02/26/2020	24	02/26/2022
8190	03/27/2020	36	03/27/2023
8191	04/16/2020	24	04/16/2022
8192	05/21/2020	24	05/21/2022
8193	08/14/2020	12	08/14/2021
8194	10/07/2020	18	04/07/2022
8195	12/05/2020	12	12/05/2021
8196	01/22/2021	12	01/22/2022
8197	02/03/2021	18	08/03/2022
8198	03/17/2021	24	03/17/2023
8199	04/23/2021	24	04/23/2023
8200	05/04/2021	12	05/04/2022
8201	06/30/2021	12	06/30/2022
8202	07/26/2021	12	07/26/2022
8203	12/27/2021	36	12/27/2024
8204	06/06/2022	24	06/06/2024
8205	07/18/2022	12	07/18/2023
8206	11/14/2022	12	11/14/2023
8207	12/12/2022	18	06/12/2024

addyears

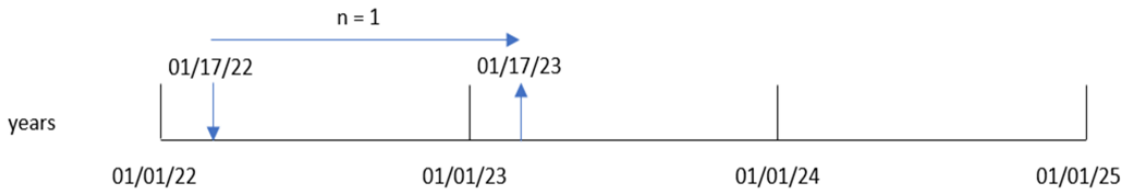
此函数用于返回在 **startdate** 后 **n** 年内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 年内发生的日期。

语法：

```
AddYears(startdate, n)
```

返回数据类型： 双

addyears() 函数的示例图表



addyears() 函数从 startdate 中加上或减去定义的年数 n。然后返回所得日期。

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。

函数示例

示例	结果
addyears ('01/29/2010',3)	返回 '01/29/2013'。
addyears ('01/29/2010',-1)	返回 '01/29/2009'。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 简单示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 two_years_later，返回交易发生后两年的日期。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        addyears(date,2) as two_years_later
        ;
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

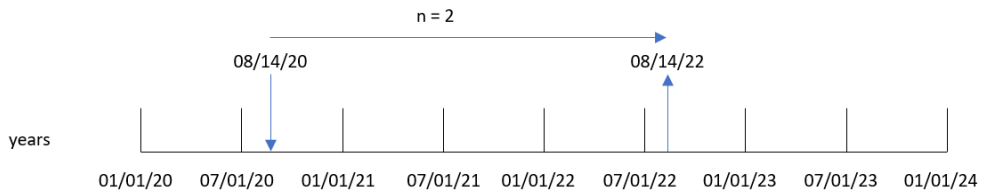
- date
- two_years_later

结果表

日期	two_years_later
01/10/2020	01/10/2022
02/28/2020	02/28/2022
04/09/2020	04/09/2022
04/16/2020	04/16/2022
05/21/2020	05/21/2022
08/14/2020	08/14/2022
10/07/2020	10/07/2022
12/05/2020	12/05/2022
01/22/2021	01/22/2023
02/03/2021	02/03/2023
03/17/2021	03/17/2023
04/23/2021	04/23/2023
05/04/2021	05/04/2023
06/30/2021	06/30/2023
07/26/2021	07/26/2023
12/27/2021	12/27/2023
02/02/2022	02/02/2024
02/26/2022	02/26/2024
03/07/2022	03/07/2024
03/11/2022	03/11/2024

two_years_later 字段是在前置 Load 语句中使用 addyears() 函数创建的。提供的第一个参数标识正在评估的日期。第二个参数是要从开始日期中加减的年数。在本例中，提供的值为 2。

addyears() 函数的图表，基本示例



交易 8193 发生在 2020 年 8 月 14日。因此，对于 two_years_later字段， addyears() 函数返回 August 14, 2022。

示例 2 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

在图表对象中，创建一个度量 prior_year_date，该度量返回交易发生前一年的日期。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量以计算每笔交易前一年的日期：

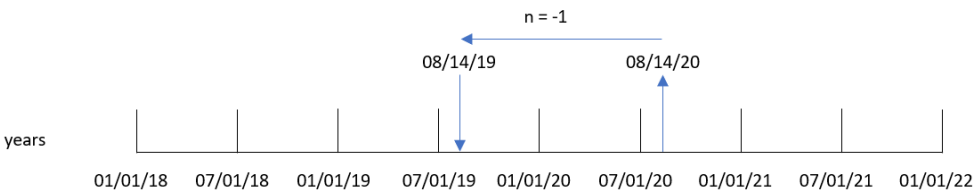
```
=addyears(date,-1)
```

结果表

日期	=addyears(date,-1)
01/10/2020	01/10/2019
02/28/2020	02/28/2019
04/09/2020	04/09/2019
04/16/2020	04/16/2019
05/21/2020	05/21/2019
08/14/2020	08/14/2019
10/07/2020	10/07/2019
12/05/2020	12/05/2019
01/22/2021	01/22/2020
02/03/2021	02/03/2020
03/17/2021	03/17/2020
04/23/2021	04/23/2020
05/04/2021	05/04/2020
06/30/2021	06/30/2020
07/26/2021	07/26/2020
12/27/2021	12/27/2020
02/02/2022	02/02/2021
02/26/2022	02/26/2021
03/07/2022	03/07/2021
03/11/2022	03/11/2021

通过使用 addyears() 函数在图表对象中创建 one_year_prior 度量。提供的第一个参数标识正在评估的日期。第二个参数是要从 startdate 中加减的年数。在本例中，提供的值为 1。

addyears() 函数的图表，图表对象示例



交易 8193 发生在 8 月 14 日。因此，对于 one_year_prior字段， addyears() 函数返回 August 14, 2019。

示例 3 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Warranties 的表中的数据集。
- 包含产品 ID、购买日期、保修期和购买价格的信息。

最终用户想要一个图表对象，该对象按产品 ID 显示每个产品的保修终止日期。

加载脚本

```
Warranties:
Load
*
Inline
[
product_id,purchase_date,warranty_length,purchase_price
8188,'01/13/2020',4,32000
8189,'02/26/2020',2,28000
8190,'03/27/2020',3,41000
8191,'04/16/2020',4,17000
8192,'05/21/2020',2,25000
8193,'08/14/2020',1,59000
8194,'10/07/2020',2,12000
8195,'12/05/2020',3,12000
8196,'01/22/2021',4,24000
8197,'02/03/2021',1,50000
8198,'03/17/2021',2,80000
8199,'04/23/2021',3,10000
8200,'05/04/2021',4,30000
8201,'06/30/2021',3,30000
8202,'07/26/2021',4,20000
8203,'12/27/2021',4,10000
8204,'06/06/2022',2,25000
8205,'07/18/2022',1,32000
8206,'11/14/2022',1,30000
8207,'12/12/2022',4,22000
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- product_id
- purchase_date

- warranty_length

创建以下度量以计算每个产品保修的结束日期：

```
=addyears(purchase_date,warranty_length)
```

结果表

product_id	purchase_date	warranty_length	=addyears(purchase_date,warranty_length)
8188	01/13/2020	4	01/13/2024
8189	02/26/2020	2	02/26/2022
8190	03/27/2020	3	03/27/2023
8191	04/16/2020	4	04/16/2024
8192	05/21/2020	2	05/21/2022
8193	08/14/2020	1	08/14/2021
8194	10/07/2020	2	10/07/2022
8195	12/05/2020	3	12/05/2023
8196	01/22/2021	4	01/22/2025
8197	02/03/2021	1	02/03/2022
8198	03/17/2021	2	03/17/2023
8199	04/23/2021	3	04/23/2024
8200	05/04/2021	4	05/04/2025
8201	06/30/2021	3	06/30/2024
8202	07/26/2021	4	07/26/2025
8203	12/27/2021	4	12/27/2025
8204	06/06/2022	2	06/06/2024
8205	07/18/2022	1	07/18/2023
8206	11/14/2022	1	11/14/2023
8207	12/12/2022	4	12/12/2026

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	描述
timestamp	时间戳或对时间戳求值的表达式都可用于转换。

参数	描述
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 *如果使用 DST 偏移量(即，指定的 **ignore_dst** 参数值计算为 False)，则必须在 **place** 参数中指定一个位置，而不是 GMT 偏移量。这是因为调整夏令时除了需要 GMT 偏移量提供的纵向信息外，还需要纬度信息。相关信息，请参阅将 GMT 偏移量与夏令时结合使用 (page 588)。* *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	如果此参数计算为 True，则忽略 DST(夏令时)。有效的参数值计算为 True，包括 -1 和 True()。 如果此参数计算为 False，则会根据夏令时调整时间戳。有效的参数值计算为 False，包括 0 和 False()。 如果 **ignore_dst** 参数值无效，则函数将计算表达式，如同 **ignore_dst** 值计算为 True 一样。如果未指定 **ignore_dst** 参数值，则函数将计算表达式，如同 **ignore_dst** 值计算为 False. 一样。

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

A-C	D-K	L-R	S-Z
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
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	-

A-C	D-K	L-R	S-Z
Chihuahua	-	Riyadh	-
Chongqing	-	Rome	-
Copenhagen	-	-	-

示例和结果：

脚本示例

示例	结果
`ConvertToLocalTime('2023-08-14 08:39:47','Paris')`	返回“2023-08-14 10:39:47”以及相应的内部时间戳表示。
`ConvertToLocalTime(UTC(), 'Stockholm')`	返回斯德哥尔摩的时间，并根据夏令时进行调整。
`ConvertToLocalTime(UTC(), 'Stockholm', -1)`	返回斯德哥尔摩的时间，不调整夏令时。
`ConvertToLocalTime(UTC(), 'GMT-05:00')`	返回北美东海岸的时间，例如纽约。由于指定了 GMT 偏移量，而不是位置，因此不会对夏令时进行调整。
`ConvertToLocalTime(UTC(), 'New York', -1)`	返回北美东海岸(纽约)的时间，不调整夏令时。
`ConvertToLocalTime(UTC(), 'New York', True())`	返回北美东海岸(纽约)的时间，不调整夏令时。
`ConvertToLocalTime(UTC(), 'New York', 0)`	返回北美东海岸(纽约)的时间，并根据夏令时进行调整。
`ConvertToLocalTime(UTC(), 'New York', False())`	返回北美东海岸(纽约)的时间，并根据夏令时进行调整。

将 GMT 偏移量与夏令时结合使用

在 Qlik Sense 实施 Unicode (ICU) 库的国际组件之后，将 GMT(格林尼治标准时间)偏移量与 DST(夏令时)结合使用需要额外的纬度信息。

GMT 是纵向(东西)偏移，而夏令时是横向(南北)偏移。例如，赫尔辛基(芬兰)和约翰内斯堡(南非)共享相同的 GMT+02:00 偏移量，但它们不共享相同的夏令时偏移量。这意味着，除了 GMT 偏移之外，任何夏令时偏移都需要关于当地时区的纬度位置的信息(地理时区输入)，以便获得关于当地夏令时条件的完整信息。

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

示例 1 – DateFormat 数据集(脚本)

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名称为 Master_Calendar 的日期数据集。DateFormat 系统变量设置为 DD/MM/YYYY。
- 使用 day() 功能创建另一个名为 day_of_month 的字段的前置 Load。
- 另一个名为 long_date 的字段，使用 date() 功能表示完整的月份名称。

加载脚本

```
SET DateFormat='DD/MM/YYYY';

Master_Calendar:
Load
        date,
        date(date,'dd-MMMM-YYYY') as long_date,
        day(date) as day_of_month
Inline
[
date
03/11/2022
03/12/2022
03/13/2022
03/14/2022
03/15/2022
03/16/2022
03/17/2022
03/18/2022
03/19/2022
03/20/2022
03/21/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- long_date
- day_of_month

结果表

日期	long_date	day_of_month
03/11/2022	2022 年 3 月 11 日	11
03/12/2022	2022 年 3 月 12 日	12
03/13/2022	2022 年 3 月 13 日	13
03/14/2022	2022 年 3 月 14 日	14
03/15/2022	2022 年 3 月 15 日	15
03/16/2022	2022 年 3 月 16 日	16
03/17/2022	2022 年 3 月 17 日	17
03/18/2022	2022 年 3 月 18 日	18
03/19/2022	2022 年 3 月 19 日	19
03/20/2022	2022 年 3 月 20 日	20
03/21/2022	2022 年 3 月 21 日	21

脚本中的 day() 函数可以正确计算月份的日期。

示例 2–ANSI 日期(脚本)

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名称为 Master_Calendar 的日期数据集。使用 DateFormat 系统变量 DD/MM/YYYY。但是，数据集中包含的日期采用 ANSI 标准日期格式。
- 使用 date() 功能创建另一个名为 day_of_month 的字段的前置 Load。
- 另一个名为 long_date 的字段，使用 date() 功能表示带完整月份名称的日期。

加载脚本

```
SET DateFormat='DD/MM/YYYY';
Master_Calendar:
Load
       date,
       date(date,'dd-MMMM-YYYY') as long_date,
       day(date) as day_of_month

Inline
[
date
2022-03-11
2022-03-12
2022-03-13
2022-03-14
2022-03-15
2022-03-16
2022-03-17
2022-03-18
2022-03-19
2022-03-20
2022-03-21
 ];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- long_date
- day_of_month

结果表

日期	long_date	day_of_month
03/11/2022	2022 年 3 月 11 日	11
03/12/2022	2022 年 3 月 12 日	12
03/13/2022	2022 年 3 月 13 日	13
03/14/2022	2022 年 3 月 14 日	14
03/15/2022	2022 年 3 月 15 日	15
03/16/2022	2022 年 3 月 16 日	16
03/17/2022	2022 年 3 月 17 日	17
03/18/2022	2022 年 3 月 18 日	18
03/19/2022	2022 年 3 月 19 日	19

日期	long_date	day_of_month
03/20/2022	2022 年 3 月 20 日	20
03/21/2022	2022 年 3 月 21 日	21

脚本中的 day() 函数可以正确计算月份的日期。

示例 3 – 未格式化日期(脚本)

加载脚本和结果

概述

打开 数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 名称为 Master_Calendar 的日期数据集。使用 DateFormat 系统变量 DD/MM/YYYY。
- 使用 day() 功能创建另一个名为 day_of_month 的字段的前置 Load。
- 原始未格式化日期, 命名为 unformatted_date。
- 另一个名为 long_date 的字段使用 date() 将数字日期转换为格式化日期字段。

加载脚本

```
SET DateFormat='DD/MM/YYYY';

Master_Calendar:
Load
      unformatted_date,
      date(unformatted_date,'dd-MMMM-YYYY') as long_date,
      day(date) as day_of_month

Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- unformatted_date
- long_date
- day_of_month

结果表

unformatted_date	long_date	day_of_month
44868	2022 年 11 月 3 日	3
44898	2022 年 12 月 3 日	3
44928	2023 年 1 月 2 日	2
44958	2023 年 2 月 1 日	1
44988	2023 年 3 月 3 日	3
45008	2023 年 3 月 23 日	23
45018	2023 年 4 月 2 日	2
45038	2023 年 4 月 22 日	22
45048	2023 年 5 月 2 日	2
45068	2023 年 5 月 22 日	22
45078	2023 年 6 月 1 日	1

脚本中的 day() 函数可以正确计算月份的日期。

示例 4 – 计算到期月 (图表)

加载脚本和图表表达式

概述

打开 数据加载编辑器 , 并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名为 orders 的 3 月订单数据集。表格包含三个字段：
  - id
  - order_date
  - 金额

加载脚本

```
Orders:
Load
        id,
        order_date,
        amount
Inline
[
id,order_date,amount
1,03/01/2022,231.24
2,03/02/2022,567.28
3,03/03/2022,364.28
4,03/04/2022,575.76
5,03/05/2022,638.68
6,03/06/2022,785.38
7,03/07/2022,967.46
8,03/08/2022,287.67
9,03/09/2022,764.45
10,03/10/2022,875.43
11,03/11/2022,957.35
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：order_date。

要计算交货日期，请创建此度量 =day(order_date+5)。

结果表

order_date	=day(order_date+5)
03/11/2022	16
03/12/2022	17
03/13/2022	18
03/14/2022	19
03/15/2022	20
03/16/2022	21
03/17/2022	22
03/18/2022	23
03/19/2022	24
03/20/2022	25
03/21/2022	26

day() 功能可正确确定 3 月 11 日的订单将在 16 日根据 5 天的交货期交货。

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]])
```

适用场景

当用户希望计算使用当天尚未发生的分数时， dayend() 函数通常用作表达式的一部分。例如，计算当天仍将发生的总费用。

返回数据类型： 双

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，可在 **day_start** 中指定一个偏移作为某日内时间的小数。例如，0.125 表示上午 3 点。 换句话说，要创建偏移，请将开始时间除以 24 小时。例如，对于 7:00 AM 开始的一天，使用分数 7/24。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
dayend('01/25/2013 16:45:00')	Returns 01/25/2013 23:59:59. PM
dayend('01/25/2013 16:45:00', -1)	Returns 01/24/2013 23:59:59. PM
dayend('01/25/2013 16:45:00', 0, 0.5)	Returns 01/26/2013 11:59:59. PM

示例 1 - 基本脚本

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含日期列表的数据集，加载到名为 "Calendar" 的表中。
- 默认 DateFormat 系统变量 (MM/DD/YYYY)。
- 使用 dayend() 功能创建另一个名为 'EOD_timestamp' 的字段的前置 Load。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Calendar:
    Load
      date,
      dayend(date) as EOD_timestamp
      ;
Load
date
Inline
[
date
03/11/2022 1:47:15 AM
03/12/2022 4:34:58 AM
03/13/2022 5:15:55 AM
03/14/2022 9:25:14 AM
03/15/2022 10:06:54 AM
03/16/2022 10:44:42 AM
03/17/2022 11:33:30 AM
03/18/2022 12:58:14 PM
03/19/2022 4:23:12 PM
03/20/2022 6:42:15 PM
03/21/2022 7:41:16 PM
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- EOD_timestamp

结果表

日期	EOD_timestamp
03/11/2022 1:47:15 AM	3/11/2022 11:59:59 PM
03/12/2022 4:34:58 AM	3/12/2022 11:59:59 PM
03/13/2022 5:15:55 AM	3/13/2022 11:59:59 PM
03/14/2022 9:25:14 AM	3/14/2022 11:59:59 PM
03/15/2022 10:06:54 AM	3/15/2022 11:59:59 PM
03/16/2022 10:44:42 AM	3/16/2022 11:59:59 PM
03/17/2022 11:33:30 AM	3/17/2022 11:59:59 PM
03/18/2022 12:58:14 PM	3/18/2022 11:59:59 PM
03/19/2022 4:23:12 PM	3/19/2022 11:59:59 PM
03/20/2022 6:42:15 PM	3/20/2022 11:59:59 PM
03/21/2022 7:41:16 PM	3/21/2022 11:59:59 PM

如上表所示，为数据集中的每个日期生成了一天结束时间戳。时间戳采用系统变量 `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT` 的格式。

示例 2 – period_no

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

您将把包含服务预订的数据集加载到名为“Services”的表中。

数据集包括以下字段：

- `service_id`
- `service_date`
- `amount`

您将在表中创建两个新字段：

- `deposit_due_date`：应收到存款的日期。这是 `service_date` 前三天的一天结束时间。
- `final_payment_due_date`：应收到最终付款的日期。这是 `service_date` 后七天的一天结束时间。

上述两个字段是在前置 Load 中使用 `dayend()` 函数创建的，它们提供前两个参数，`time` 以及 `period_no`。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Services:
   Load
     *,
     dayend(service_date,-3) as deposit_due_date,
     dayend(service_date,7) as final_payment_due_date
     ;
Load
service_id,
service_date,
amount
Inline
[
service_id, service_date,amount
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- service_date
- deposit_due_date
- final_payment_due_date

结果表

service_date	deposit_due_date	final_payment_due_date
03/11/2022 9:25:14 AM	3/8/2022 11:59:59 PM	3/18/2022 11:59:59 PM
03/12/2022 10:06:54 AM	3/9/2022 11:59:59 PM	3/19/2022 11:59:59 PM
03/13/2022 10:44:42 AM	3/10/2022 11:59:59 PM	3/20/2022 11:59:59 PM
03/14/2022 11:33:30 AM	3/11/2022 11:59:59 PM	3/21/2022 11:59:59 PM
03/15/2022 12:58:14 PM	3/12/2022 11:59:59 PM	3/22/2022 11:59:59 PM
03/16/2022 4:23:12 PM	3/13/2022 11:59:59 PM	3/23/2022 11:59:59 PM

service_date	deposit_due_date	final_payment_due_date
03/17/2022 6:42:15 PM	3/14/2022 11:59:59 PM	3/24/2022 11:59:59 PM
03/18/2022 7:41:16 PM	3/15/2022 11:59:59 PM	3/25/2022 11:59:59 PM
03/19/2022 8:14:15 PM	3/16/2022 11:59:59 PM	3/26/2022 11:59:59 PM
03/20/2022 9:23:51 PM	3/17/2022 11:59:59 PM	3/27/2022 11:59:59 PM
03/21/2022 10:04:41 PM	3/18/2022 11:59:59 PM	3/28/2022 11:59:59 PM

新字段的值位于 `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT` 中。因为使用了函数 `dayend()`，所以时间戳值都是一天中的最后一毫秒。

由于 `dayend()` 函数中传递的第二个参数为负数，存款到期日值比服务日期早三天。

由于 `dayend()` 函数中传递的第二个参数为正数，最终付款到期日值为服务日期后七天。

示例 3 – day_start script

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

本示例中使用的数据集和场景与前一示例中的相同。

与上一个示例一样，您将创建两个新字段：

- `deposit_due_date`：应收到存款的日期。这是 `service_date` 前三天的一天结束时间。
- `final_payment_due_date`：应收到最终付款的日期。这是 `service_date` 后七天的一天结束时间。

但是，您的公司希望在工作日从下午 5 点开始到第二天下午 5 点结束的政策下运营。然后，您的公司可以监控在这些工作时间内发生的交易。

为了满足这些要求，在前置 Load 中使用 `dayend()` 函数创建了上述两个字段，并使用所有三个参数 `time`、`period_no` 和 `day_start`。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Services:
   Load
     *,
     dayend(service_date,-3,17/24) as deposit_due_date,
     dayend(service_date,7,17/24) as final_payment_due_date
     ;
Load
service_id,
service_date,
amount
```

```
Inline
[
service_id, service_date,amount
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- service_date
- deposit_due_date
- final_payment_due_date

结果表

service_date	deposit_due_date	final_payment_due_date
03/11/2022 9:25:14 AM	3/8/2022 4:59:59 PM	3/18/2022 4:59:59 PM
03/12/2022 10:06:54 AM	3/9/2022 4:59:59 PM	3/19/2022 4:59:59 PM
03/13/2022 10:44:42 AM	3/10/2022 4:59:59 PM	3/20/2022 4:59:59 PM
03/14/2022 11:33:30 AM	3/11/2022 4:59:59 PM	3/21/2022 4:59:59 PM
03/15/2022 12:58:14 PM	3/12/2022 4:59:59 PM	3/22/2022 4:59:59 PM
03/16/2022 4:23:12 PM	3/13/2022 4:59:59 PM	3/23/2022 4:59:59 PM
03/17/2022 6:42:15 PM	3/14/2022 4:59:59 PM	3/24/2022 4:59:59 PM
03/18/2022 7:41:16 PM	3/15/2022 4:59:59 PM	3/25/2022 4:59:59 PM
03/19/2022 8:14:15 PM	3/16/2022 4:59:59 PM	3/26/2022 4:59:59 PM
03/20/2022 9:23:51 PM	3/17/2022 4:59:59 PM	3/27/2022 4:59:59 PM
03/21/2022 10:04:41 PM	3/18/2022 4:59:59 PM	3/28/2022 4:59:59 PM

虽然日期与示例 2 中的日期保持相同，但由于传递给 dayend() 函数的第三个参数 day_start 的值为 17/24，因此日期现在具有下午 5 点之前最后一毫秒的时间戳。

示例 4 – 图表示例

加载脚本和图表表达式

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

本示例中使用的数据集和场景与前两个示例中的相同。公司希望在工作日从下午 5 点开始到第二天下午 5 点结束的政策下运营。

与上一个示例一样，您将创建两个新字段：

- deposit_due_date：应收到存款的日期。这是 service_date 前三天的一天结束时间。
- final_payment_due_date：应收到最终付款的日期。这是 service_date 后七天的一天结束时间。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Services:
Load
service_id,
service_date,
amount
Inline
[
service_id, service_date,amount
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

service_date。

要创建 deposit_due_date 字段，请创建此度量：

=dayend(service_date,-3,17/24)。

然后，要创建 final_payment_due_date 字段，请创建此度量：

=dayend(service_date,7,17/24)。

结果表

service_date	=dayend(service_date,-3,17/24)	=dayend(service_date,7,17/24
03/11/2022	3/8/2022 16:59:59 PM	3/18/2022 16:59:59 PM
03/12/2022	3/9/2022 16:59:59 PM	3/19/2022 16:59:59 PM
03/13/2022	3/10/2022 16:59:59 PM	3/20/2022 16:59:59 PM
03/14/2022	3/11/2022 16:59:59 PM	3/21/2022 16:59:59 PM
03/15/2022	3/12/2022 16:59:59 PM	3/22/2022 16:59:59 PM
03/16/2022	3/13/2022 16:59:59 PM	3/23/2022 16:59:59 PM
03/17/2022	3/14/2022 16:59:59 PM	3/24/2022 16:59:59 PM
03/18/2022	3/15/2022 16:59:59 PM	3/25/2022 16:59:59 PM
03/19/2022	3/16/2022 16:59:59 PM	3/26/2022 16:59:59 PM
03/20/2022	3/17/2022 16:59:59 PM	3/27/2022 16:59:59 PM
03/21/2022	3/18/2022 16:59:59 PM	3/28/2022 16:59:59 PM

新字段的值位于 `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT` 中。因为使用了函数 `dayend()`，所以时间戳值都是一天中的最后一毫秒。

由于 `dayend()` 函数中传递的第二个参数为负数，付款到期日值比服务日期早三天。

由于 `dayend()` 函数中传递的第二个参数为正数，最终付款到期日值为服务日期后七天。

但由于传递给 `dayend()` 函数的第三个参数 `day_start` 的值为 17/24，因此日期现在具有下午 5 点之前最后一毫秒的时间戳。

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，可在 **day_start** 中指定一个偏移作为某日内时间的小数。例如，0.125 表示上午 3 点。

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。
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，可在 **day_start** 中指定一个偏移作为某日内时间的小数。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。创建主日历时使用此功能。

语法：

```
DayNumberOfQuarter(timestamp[,start_month])
```

返回数据类型：整数

参数

参数	说明
timestamp	要评估的日期或时间戳。
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。

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。

函数示例

示例	结果
`DayNumberOfQuarter('12/09/2014')`	返回 74，当前季度的天数。

示例	结果
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 1 – 年初 1 月(脚本)

加载脚本和结果

概述

打开 数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 包含日期列表的简单数据集, 该数据集加载到名为 Calendar 的表中。使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 使用 DayNumberOfQuarter() 功能创建另一个名为 DayNrQtr 的字段的前置 Load。

除了日期之外, 没有为函数提供其他参数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
        date,
        DayNumberOfQuarter(date) as DayNrQtr
        ;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- date
- daynrqtr

结果表

日期	daynrqtr
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
02/28/2022	59
03/01/2022	61
03/31/2022	91
04/01/2022	1

一年中的第一天是 1 月 1 日，因为没有向 DayNumberOfQuarter() 函数传递第二个参数。

1 月 1 日是季度的第一天，而 2 月 1 日是季度的第 32 天。3 月 31 日是本季度的第 91 天也是最后一天，而 4 月 1 日是第二季度的第一天。

示例 2 – 年初 2 月 (脚本)

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个示例中的相同数据集。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 2 月 1 日开始的 start_month 参数。此项将财政年度设置为 2 月 1 日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
       date,
       DayNumberOfQuarter(date,2) as DayNrQtr
       ;
Load
date
```

```
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- daynrqtr

结果表

日期	daynrqtr
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	1
02/10/2022	10
02/28/2022	28
03/01/2022	30
03/31/2022	60
04/01/2022	61

一年的第一天是 2 月 1 日，因为传递给 DayNumberOfQuarter() 函数的第二个参数是 2。

今年第一季度的营业时间为 2 月至 4 月，而第四季度的营业时间为 11 月至 1 月。结果表中显示了这一点，其中2月1日是本季度的第一天，而1月31日是本季度的第92天也是最后一天。

示例 3 – 年初 1 月 (图表)

加载脚本和图表表达式

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个示例中的相同数据集。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。

然而，在本例中，未更改的数据集被加载到应用程序中。季度中某一天的值是通过图表对象中的度量计算的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量：

```
=daynumberofquarter(date)
```

结果表

日期	=daynumberofquarter(date)
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
02/28/2022	59
03/01/2022	61
03/31/2022	91
04/01/2022	1

一年中的第一天是 1 月 1 日，因为没有向 DayNumberOfQuarter() 函数传递第二个参数。

1 月 1 日是季度的第一天，而 2 月 1 日是季度的第 32 天。3 月 31 日是本季度的第 91 天也是最后一天，而 4 月 1 日是第二季度的第一天。

示例 4 – 年初 2 月 (图表)

加载脚本和图表表达式

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个示例中的相同数据集。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 财政年度从 2 月 1 日至 1 月 31 日。

然而，在本例中，未更改的数据集被加载到应用程序中。季度中某一天的值是通过图表对象中的度量计算的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

图表对象

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量：

```
=daynumberofquarter(date,2)
```

结果

结果表

日期	=daynumberofquarter(date,2)
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	1
02/10/2022	10
02/28/2022	28
03/01/2022	30
03/31/2022	60
04/01/2022	61

一年的第一天是 1 月 1 日，因为传递给 `DayNumberOfQuarter()` 函数的第二个参数是 2。

今年第一季度的营业时间为 2 月至 4 月，而第四季度的营业时间为 11 月至 1 月。结果表中证明了这一点，其中2月1日是本季度的第一天，而1月31日是本季度的第92天也是最后一天。

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

参数

参数	说明
timestamp	要评估的日期或时间戳。
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。

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。

函数示例

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 1 – 年初 1 月 (脚本)

加载脚本和结果

概述

打开 数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 包含日期列表的简单数据集, 该数据集加载到名为 Calendar 的表中。使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 使用 DayNumberOfYear() 功能创建另一个名为 daynryear 的字段的前置 Load。

除了日期之外, 没有为函数提供其他参数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
       date,
       DayNumberOfYear(date) as daynryear
       ;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- date
- daynryear

结果表

日期	daynryear
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
06/30/2022	182
07/26/2022	208
10/31/2022	305
11/01/2022	306
12/31/2022	366

一年中的第一天是 1 月 1 日，因为没有向 DayNumberOfYear() 函数传递第二个参数。

1 月 1 日是季度的第一天，而 2 月 1 日是年度的第 32 天。6 月 30 日是第 182 日，12 月 31日 是一年中的第 366 天，也是最后一天。

示例 2 – 年初 11 月 (脚本)

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个示例中的相同数据集。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY
- 11 月 1 日开始的 start_month 参数。此项将财政年度设置为 11 月 1 日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
       date,
       DayNumberOfYear(date,11) as daynryear
       ;
```

```
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- daynryear

结果表

日期	daynryear
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	93
02/10/2022	102
06/30/2022	243
07/26/2022	269
10/31/2022	366
11/01/2022	1
12/31/2022	61

一年的第一天是 11 月 1 日，因为传递给 `DayNumberOfYear()` 函数的第二个参数是 11。

1 月 1 日是季度的第一天，而 2 月 1 日是年度的第 32 天。6 月 30 日是第 182 日，12 月 31日 是一年中的第 366 天，也是最后一天。

示例 3 – 年初 1 月 (图表)

加载脚本和图表表达式

概述

打开 数据加载编辑器 , 并将下面的加载脚本添加到新选项卡。

加载脚本包含 :

- 第一个示例中的相同数据集。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。

然而 , 在本例中 , 未更改的数据集被加载到应用程序中。季度中某一天的值是通过图表对象中的度量计算的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度 : date。

创建以下度量 :

```
=daynumberofyear(date)
```

结果表

日期	=daynumberofyear(date)
01/01/2022	1
01/10/2022	10

日期	=daynumberofyear(date)
01/31/2022	31
02/01/2022	32
02/10/2022	41
06/30/2022	182
07/26/2022	208
10/31/2022	305
11/01/2022	306
12/31/2022	366

一年中的第一天是 1 月 1 日，因为没有向 DayNumberOfYear() 函数传递第二个参数。

1 月 1 日是年度的第一天，而 2 月 1 日是季度的第 32 天。6 月 30 日是第 182 日，12 月 31日 是一年中的第 366 天，也是最后一天。

示例 4 – 年初 11 月 (图表)

加载脚本和图表表达式

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个示例中的相同数据集。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 财政年度从 11 月 1 日至 10 月 31 日。

然而，在本例中，未更改的数据集被加载到应用程序中。年度中某一天的值是通过图表对象中的度量计算的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
```

```
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量：

```
=daynumberofyear(date)
```

结果表

日期	=daynumberofyear(date,11)
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	93
02/10/2022	102
06/30/2022	243
07/26/2022	269
10/31/2022	366
11/01/2022	1
12/31/2022	61

一年的第一天是 11 月 1 日，因为传递给 DayNumberOfYear() 函数的第二个参数是 11。

财政年度从 11 月持续至 10 月。结果表中显示了这一点，其中 11 月 1 日是本年度的第一天，而 10 月 31 日是年度的第 366 天也是最后一天。

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	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，可在 **day_start** 中指定一个偏移作为某日内时间的小数。例如，0.125 表示上午 3 点。 换句话说，要创建偏移，请将开始时间除以 24 小时。例如，对于 7:00 AM 开始的一天，使用分数 7/24。

适用场景

当用户希望计算使用到目前为止已过的一天的部分时，daystart() 函数通常用作表达式的一部分。例如，它可以用于计算员工迄今为止挣得的总工资。

以下示例使用时间戳格式 'M/D/YYYY h:mm:ss[.fff] TT'。时间戳格式已经在数据加载脚本顶部的 SET TimeStamp 语句中指定。可以根据要求更改示例中的格式。

函数示例

示例	结果
daystart('01/25/2013 4:45:00 PM')	返回 1/25/2013 12:00:00 AM。
daystart('1/25/2013 4:45:00 PM', -1)	返回 1/24/2013 12:00:00 AM。
daystart('1/25/2013 16:45:00',0,0.5)	返回 1/25/2013 12:00:00 PM。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 简单示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含日期列表的简单数据集，该数据集加载到名为 Calendar 的表中。
- 使用默认 TimeStampFormat 系统变量 (M/D/YYYY h:mm:ss[.fff] TT。
- 使用 daystart() 功能创建另一个名为 SOD_timestamp 的字段的前置 Load。

除了日期之外，没有为函数提供其他参数。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Calendar:
    Load
        date,
        daystart(date) as SOD_timestamp
        ;
Load
date
Inline
[
date
03/11/2022 1:47:15 AM
03/12/2022 4:34:58 AM
03/13/2022 5:15:55 AM
03/14/2022 9:25:14 AM
03/15/2022 10:06:54 AM
03/16/2022 10:44:42 AM
03/17/2022 11:33:30 AM
03/18/2022 12:58:14 PM
03/19/2022 4:23:12 PM
03/20/2022 6:42:15 PM
03/21/2022 7:41:16 PM
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- SOD_timestamp

结果表

日期	SOD_timestamp
03/11/2022 1:47:15 AM	3/11/2022 12:00:00 AM
03/12/2022 4:34:58 AM	3/12/2022 12:00:00 AM
03/13/2022 5:15:55 AM	3/13/2022 12:00:00 AM
03/14/2022 9:25:14 AM	3/14/2022 12:00:00 AM
03/15/2022 10:06:54 AM	3/15/2022 12:00:00 AM
03/16/2022 10:44:42 AM	3/16/2022 12:00:00 AM
03/17/2022 11:33:30 AM	3/17/2022 12:00:00 AM
03/18/2022 12:58:14 PM	3/18/2022 12:00:00 AM
03/19/2022 4:23:12 PM	3/19/2022 12:00:00 AM
03/20/2022 6:42:15 PM	3/20/2022 12:00:00 AM
03/21/2022 7:41:16 PM	3/21/2022 12:00:00 AM

如上表所示，为数据集中的每个日期生成了一天结束时间戳。时间戳采用系统变量 `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT` 的格式。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含停车罚款的数据集，将其加载到名为 `Fines` 的表中。数据集包括以下字段：
  - `id`
  - `due_date`
  - `number_plate`
  - `amount`
- 使用 `daystart()` 函数并提供所有以下三个参数的前置 Load：`time`、`period_no` 和 `day_start`。前置 Load 创建了以下两个新的日期字段：
  - `early_repayment_period` 日期字段，从付款到期前七天开始。
  - `late_penalty_period` 日期字段，从付款到期后 14 天开始。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Fines:
```

```
    Load
        *,
        daystart(due_date,-7) as early_repayment_period,
        daystart(due_date,14) as late_penalty_period
        ;
Load
*
Inline
[
id, due_date, number_plate,amount
1,02/11/2022, 573RJG,50.00
2,03/25/2022, SC41854,50.00
3,04/14/2022, 8EHZ378,50.00
4,06/28/2022, 8HSS198,50.00
5,08/15/2022, 1221665,50.00
6,11/16/2022, EAK473,50.00
7,01/17/2023, KD6822,50.00
8,03/22/2023, 1GGLB,50.00
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- due_date
- early_repayment_period
- late_penalty_period

结果表

due_date	early_repayment_period	late_penalty_period
02/11/2022 9:25:14 AM	2/4/2022 12:00:00 AM	2/25/2022 12:00:00 AM
03/25/2022 10:06:54 AM	3/18/2022 12:00:00 AM	4/8/2022 12:00:00 AM
04/14/2022 10:44:42 AM	4/7/2022 12:00:00 AM	4/28/2022 12:00:00 AM
06/28/2022 11:33:30 AM	6/21/2022 12:00:00 AM	7/12/2022 12:00:00 AM
08/15/2022 12:58:14 PM	8/8/2022 12:00:00 AM	8/29/2022 12:00:00 AM
11/16/2022 4:23:12 PM	11/9/2022 12:00:00 AM	11/30/2022 12:00:00 AM
01/17/2023 6:42:15 PM	1/10/2023 12:00:00 AM	1/31/2023 12:00:00 AM
03/22/2023 7:41:16 PM	3/15/2023 12:00:00 AM	4/5/2023 12:00:00 AM

新字段的值位于 TimestampFormat M/DD/YYYY tt 中。因为使用了函数 daystart()，所以时间戳值都是一天中的第一毫秒。

由于 daystart() 函数中传递的第二个参数为负数，因此提前还款期值为到期日前七天。

由于 daystart() 函数中传递的第二个参数为正，逾期还款期值为到期日后 14 天。

示例 3 – day_start

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与上一个示例相同的数据集和场景。
- 与前一示例相同的前置 Load。

在本例中，我们将工作日设置为每天早上 7:00 开始和结束。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Fines:
    Load
        *,
        daystart(due_date,-7,7/24) as early_repayment_period,
        daystart(due_date,14, 7/24) as late_penalty_period
        ;
Load
*
Inline
[
id, due_date, number_plate,amount
1,02/11/2022, 573RJG,50.00
2,03/25/2022, SC41854,50.00
3,04/14/2022, 8EHZ378,50.00
4,06/28/2022, 8HSS198,50.00
5,08/15/2022, 1221665,50.00
6,11/16/2022, EAK473,50.00
7,01/17/2023, KD6822,50.00
8,03/22/2023, 1GGLB,50.00
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- due_date
- early_repayment_period
- late_penalty_period

结果表

due_date	early_repayment_period	late_penalty_period
02/11/2022	2/3/2022 7:00:00 AM	2/24/2022 7:00:00 AM
03/25/2022	3/17/2022 7:00:00 AM	4/7/2022 7:00:00 AM
04/14/2022	4/6/2022 7:00:00 AM	4/27/2022 7:00:00 AM
06/28/2022	6/20/2022 7:00:00 AM	7/11/2022 7:00:00 AM
08/15/2022	8/7/2022 7:00:00 AM	8/28/2022 7:00:00 AM
11/16/2022	11/8/2022 7:00:00 AM	11/29/2022 7:00:00 AM
01/17/2023	1/9/2023 7:00:00 AM	1/30/2023 7:00:00 AM
03/22/2023	3/14/2023 7:00:00 AM	4/4/2023 7:00:00 AM

现在日期的时间戳为上午 7:00，因为传递给 daystart() 函数的 day_start 参数值为 7/24。这将一天的开始时间设置为早上 7 点。

因为 due_date 字段没有时间戳，所以它被视为 12:00 AM，这仍然是前一天的一部分，因为这些天的开始和结束时间都是早上 7:00。因此，2 月 11 日到期的罚款的提前还款期从 2 月 3 日早上 7:00 开始。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

该示例使用与上一个示例相同的数据集和场景。

但是，只有原始 Fines 表加载到应用程序中，在图表对象中计算两个额外的截止日期值。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Fines:
    Load
*
Inline
[
id, due_date, numer_plate,amount
1,02/11/2022 9:25:14 AM, 573RJG,50.00
2,03/25/2022 10:06:54 AM, SC41854,50.00
3,04/14/2022 10:44:42 AM, 8EHZ378,50.00
4,06/28/2022 11:33:30 AM, 8HSS198,50.00
5,08/15/2022 12:58:14 PM, 1221665,50.00
6,11/16/2022 4:23:12 PM, EAK473,50.00
```

```
7,01/17/2023 6:42:15 PM, KD6822,50.00
8,03/22/2023 7:41:16 PM, 1GGLB,50.00
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将该字段添加为维度：due_date。
2. 要创建 early_repayment_period 字段，请创建以下度量。
   =daystart(due_date,-7,7/24)
3. 要创建 late_penalty_period 字段，请创建以下度量：
   =daystart(due_date,14,7/24)

结果表

due_date	=daystart(due_date,-7,7/24)	=daystart(due_date,14,7/24)
02/11/2022 9:25:14 AM	2/4/2022 7:00:00 AM	2/25/2022 7:00:00 AM
03/25/2022 10:06:54 AM	3/18/2022 7:00:00 AM	4/8/2022 7:00:00 AM
04/14/2022 10:44:42 AM	4/7/2022 7:00:00 AM	4/28/2022 7:00:00 AM
06/28/2022 11:33:30 AM	6/21/2022 7:00:00 AM	7/12/2022 7:00:00 AM
08/15/2022 12:58:14 PM	8/8/2022 7:00:00 AM	8/29/2022 7:00:00 AM
11/16/2022 4:23:12 PM	11/9/2022 7:00:00 AM	11/30/2022 7:00:00 AM
01/17/2023 6:42:15 PM	1/10/2023 7:00:00 AM	1/31/2023 7:00:00 AM
03/22/2023 7:41:16 PM	3/15/2023 7:00:00 AM	4/5/2023 7:00:00 AM

新字段的值位于 TimestampFormat M/D/YYYY h:mm:ss[.fff] TT 中。因为使用了函数 daystart()，所以时间戳值对应于一天中的第一毫秒。

由于 daystart() 函数中传递的第二个参数为负数，因此提前还款期值为到期日前七天。

由于 daystart() 函数中传递的第二个参数为正，逾期还款期值为到期日后 14 天。

日期的时间戳为上午 7:00，因为传递给 day_start 函数的 daystart() 第三参数值为 7/24。

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。假日表示为字符串常量日期。您可以指定多个假期日期，以逗号分隔。 **示例：** `'12/25/2013', '12/26/2013', '12/31/2013', '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地区设置的当前 Greenwich Mean Time。函数以 TimestampFormat 系统变量格式返回值。

每当重新加载应用程序时，使用 GMT 函数的任何加载脚本表、变量或图表对象都将调整为从系统时钟导出的最新格林威治标准时间。

语法：

```
GMT( )
```

返回数据类型： 双

以下示例使用时间戳格式 M/D/YYYY h:mm:ss[.fff] TT。日期格式已经在数据加载脚本顶部的 SET TimestampFormat 语句中指定。可以根据要求更改示例中的格式。

函数示例

示例	结果
GMT()	3/28/2022 2:47:36 PM

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 变量(脚本)

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。本例将使用 GMT 函数将当前格林威治标准时间设置为加载脚本中的变量。

加载脚本

```
LET vGMT = GMT();
```

结果

加载数据并创建工作表。使用**文本和图像**图表对象创建文本框。

将此度量添加到文本框:

```
=vGMT
```

文本框应包含一行带有日期和时间的文本,类似于下图:

```
3/28/2022 2:47:36 PM
```

示例 2 – 年初 11 月(脚本)

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 包含过期库书籍的数据集,将其加载到名为 Overdue 的表中。使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 创建一个名为 days_overdue 的新字段,该字段计算每本书的过期天数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Overdue:
    Load
        *,
        Floor(GMT()-due_date) as days_overdue
        ;
Load
*
Inline
[
```

```
cust_id,book_id,due_date
1,4,01/01/2021,
2,24,01/10/2021,
6,173,01/31/2021,
31,281,02/01/2021,
86,265,02/10/2021,
52,465,06/30/2021,
26,537,07/26/2021,
92,275,10/31/2021,
27,455,11/01/2021,
27,46,12/31/2021
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- due_date
- book_id
- days_overdue

结果表

due_date	book_id	days_overdue
01/01/2021	4	455
01/10/2021	24	446
01/31/2021	173	425
02/01/2021	281	424
02/10/2021	265	415
06/30/2021	465	275
07/26/2021	537	249
10/31/2021	275	152
11/01/2021	455	151
12/31/2021	46	91

days_overdue 字段中的值是通过使用 GMT() 函数查找当前格林威治标准时间与原始到期日之间的差值来计算的。为了只计算天数，使用 Floor() 函数将结果四舍五入到最接近的整数。

示例 3–图表对象(图表)

加载脚本和图表表达式

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。加载脚本包含与前一示例相同的数据集。使用默认 DateFormat 系统变量 MM/DD/YYYY。

然而,在本例中,未更改的数据集被加载到应用程序中。过期天数的值通过图表对象中的度量计算。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Overdue:
Load
*
Inline
[
cust_id,book_id,due_date
1,4,01/01/2021,
2,24,01/10/2021,
6,173,01/31/2021,
31,281,02/01/2021,
86,265,02/10/2021,
52,465,06/30/2021,
26,537,07/26/2021,
92,275,10/31/2021,
27,455,11/01/2021,
27,46,12/31/2021
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- due_date
- book_id

创建以下度量:

```
=Floor(GMT() - due_date)
```

结果表

due_date	book_id	=Floor(GMT()-due_date)
01/01/2021	4	455
01/10/2021	24	446

due_date	book_id	=Floor(GMT()-due_date)
01/31/2021	173	425
02/01/2021	281	424
02/10/2021	265	415
06/30/2021	465	275
07/26/2021	537	249
10/31/2021	275	152
11/01/2021	455	151
12/31/2021	46	91

days_overdue 字段中的值是通过使用 GMT() 函数查找当前格林威治标准时间与原始到期日之间的差值来计算的。为了只计算天数，使用 Floor() 函数将结果四舍五入到最接近的整数。

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

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
hour('09:14:36')	提供的文本字符串隐式转换为时间戳，因为它与 TimestampFormat 变量中定义的时间戳格式相匹配。表达式返回 9。
hour('0.5555')	表达式返回 13(因为 0.5555 = 13:19:55)

示例 1 – 变量(脚本)

加载脚本和结果

概述

打开 数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 按时间戳包含交易的数据集
- 默认 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT)

创建字段 'hour', 计算购买发生的时间。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
  Load
    *,
    hour(date) as hour
    ;
Load
*
Inline
[
id,date,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- date
- hour

结果表

日期	hour
2022-01-01 22:10:22	22
2022-01-02 08:35:54	8
2022-01-03 05:40:49	5
2022-01-03 14:21:53	14
2022-01-04 18:49:38	18
2022-01-04 22:58:34	22
2022-01-05 19:04:57	19
2022-01-05 19:34:46	19
2022-01-06 08:49:09	8
2022-01-06 11:29:38	11

小时字段中的值是通过使用 hour() 函数并将日期作为前面 Load 语句中的表达式传递来创建的。

示例 2–图表对象(图表)

加载脚本和图表表达式

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个示例中的相同数据集。
- 默认 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT)。

然而，在本例中，未更改的数据集被加载到应用程序中。'hour' 通过图表对象中的度量计算值。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
*
Inline
[
id,date,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
```

```
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

要计算 'hour', 创建以下度量：

```
=hour(date)
```

结果表

due_date	=hour(date)
2022-01-01 22:10:22	22
2022-01-02 08:35:54	8
2022-01-03 05:40:49	5
2022-01-03 14:21:53	14
2022-01-04 18:49:38	18
2022-01-04 22:58:34	22
2022-01-05 19:04:57	19
2022-01-05 19:34:46	19
2022-01-06 08:49:09	8
2022-01-06 11:29:38	11

'hour' 的值是通过使用 hour() 函数并将日期作为表达式传递给图表对象的度量来创建的。

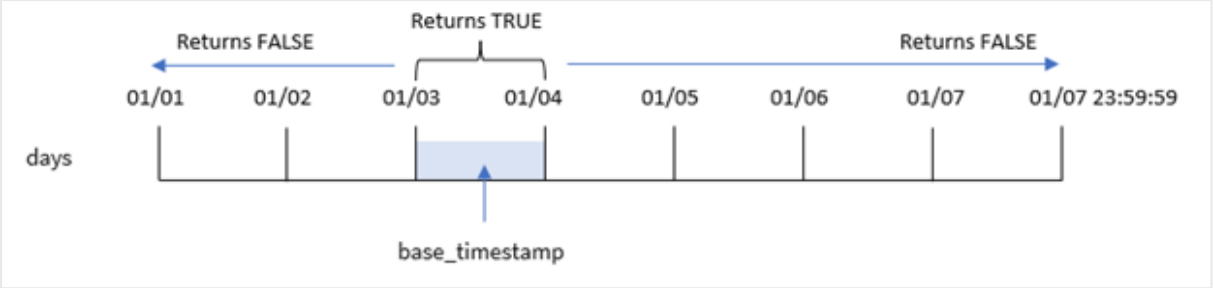
inday

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])
```

inday 函数的图表



inday() 函数使用base_timestamp 参数来标识时间戳属于哪一天。一天的开始时间默认为午夜;但是可以使用 inday() 函数的 day_start 参数更改一天的开始时间。一旦定义了这一天,当将规定的时间戳值与当天进行比较时,该函数将返回布尔结果。

适用场景

inday() 函数返回布尔值结果。通常,这种类型的函数将用作 if expression 中的条件。这将返回一个聚合或计算,具体取决于评估的日期是否发生在相关时间戳的当天。

例如,inday() 函数可用于识别给定日期内制造的所有设备。

返回数据类型: 布尔值

在 Qlik Sense 中,布尔 true 值由 -1 表示,false 值由 0 表示。

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。

区域设置

除非另有规定,本主题中的示例使用以下日期格式:MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素,系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者,您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典,则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0)	返回 True
inday ('01/12/2006 12:23:00 PM', '01/13/2006 12:00:00 AM', 0)	返回 False
inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', -1)	返回 False
(inday'01/11/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', -1)	返回 True
inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0, 0.5)	返回 False
inday ('01/12/2006 11:23:00 AM', '01/12/2006 12:00:00 AM', 0, 0.5)	返回 True

示例 1 – Load 语句(脚本)

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 按时间戳包含交易的数据集，加载到名为 Transactions 的表格中。
- 以 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT) 格式提供的日期字段。
- 包含设置为 in_day 字段的 inday() 函数的前置 Load。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        inday(date,'01/05/2022 12:00:00 AM', 0) as in_day
        ;
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
```

```
9506,'01/06/2022 8:49:09 AM',74.23
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- date
- in_day

结果表

日期	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	-1
01/05/2022 7:04:57 PM	-1
01/06/2022 8:49:09 AM	0

in_day 字段是在前置 Load 语句中使用 inday()函数创建的，并将日期字段、1月 5 日的硬编码时间戳和为 0 的 period_no 作为函数的参数传递。

示例 2 – period_no

加载脚本和结果

概述

加载脚本使用与第一个示例中相同的数据集和场景。

然而，在本例中，任务是计算交易日期是否发生在 1月 5 日前两天。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        inday(date,'01/05/2022 12:00:00 AM', -2) as in_day
        ;
Load
```

```
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_day

结果表

日期	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	-1
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	0
01/05/2022 11:29:38 AM	0
01/05/2022 7:04:57 PM	0
01/06/2022 8:49:09 AM	0

在本例中，由于 inday() 函数中使用了为 -2 的 period_no 作为偏移参数，因此函数确定每个交易日期是否发生在 1 月 3 日。这可以在输出表中进行验证，其中一个交易返回的布尔结果为 TRUE。

示例 3 – day_start

加载脚本和结果

概述

加载脚本使用与之前示例中相同的数据集和场景。

然而，在本例中，公司的政策是工作日从早上 7 点开始，到早上 7 点结束。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        inday(date,'01/05/2022 12:00:00 AM', 0, 7/24) as in_day
        ;
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_day

结果表

日期	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	-1
01/04/2022 10:58:34 PM	-1
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	0

日期	in_day
01/05/2022 7:04:57 PM	0
01/06/2022 8:49:09 AM	0

由于 inday() 函数中使用了 7/24 的 start_day 参数，即上午 7 点，因此函数确定每个交易日期是否发生在 1 月 4 日上午 7 点和 1 月 5 日早上 7 点之前。

这可以在输出表中进行验证，其中 1 月 4 日早上 7 点之后发生的交易返回布尔值结果 TRUE，而 1 月 5 日早上 7 点之后发生的事务返回布尔值结果 FALSE。

示例 4–图表对象(图表)

加载脚本和图表表达式

概述

加载脚本使用与之前示例中相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。您将通过在图表对象中创建度量来计算以确定交易是否在 1 月 5 日发生。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- date

要计算交易是否在 1 月 5 日发生，请创建以下度量：

```
=inday(date,'01/05/2022 12:00:00 AM',0)
```

结果表

日期	inday(date,'01/05/2022 12:00:00 AM',0)
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	-1
01/05/2022 7:04:57 PM	-1
01/06/2022 8:49:09 AM	0

示例 5 – 场景

加载脚本和结果

概述

在本例中，已确定由于设备错误，1月 5日制造的产品存在缺陷。最终用户想要一个图表对象，该对象按日期显示制造的“有缺陷”或“无缺陷”产品的状态以及 1月 5日制造的产品的成本。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为“Products”的表中的数据集。
- 表格包含以下字段：
    - 产品 ID
    - 制造时间
    - 成本价

加载脚本

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
```

```
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
=dayname(manufacture_date)
```

创建以下度量：

- `=if(only(InDay(manufacture_date,makedate(2022,01,05),0)),'Defective','Faultless')`
- `=sum(cost_price)`

将度量的**数字格式**设置为**金额**。

在**外观**下，关闭**总计**。

结果表

dayname (manufacture_ date)	=if(only(InDay(manufacture_date,makedate (2022,01,05),0)),'Defective','Faultless')	=sum (cost_ price)
01/01/2022	Faultless	44.67
01/02/2022	Faultless	36.34
01/03/2022	Faultless	51.75
01/04/2022	Faultless	89.69
01/05/2022	Defective	170.78
01/06/2022	Faultless	74.23

`inday()` 函数在评估每个产品的制造日期时返回布尔值。对于1月5日生产的任何产品，`inday()` 函数返回布尔值 TRUE，并将产品标记为 'Defective'。对于任何返回 FALSE 值的产品，由于不是在当天制造的，它将产品标记为 'Faultless'。

indaytotime

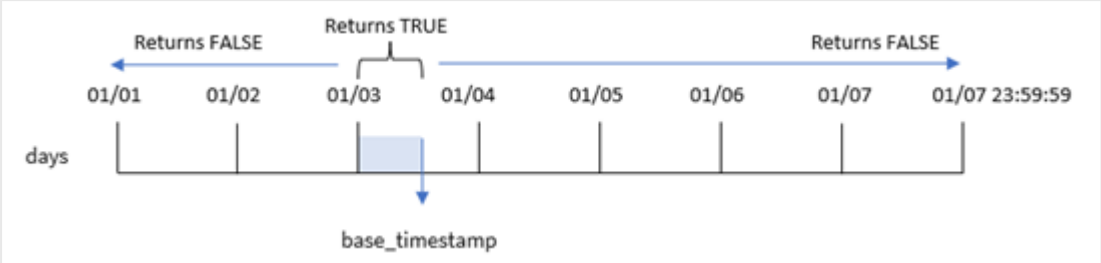
此函数用于返回 True，如果 **timestamp** 位于包含 **base_timestamp** 为止以及包括 **base_timestamp** 精确毫秒的日子部分以内。

语法：

```
InDayToTime (timestamp, base_timestamp, period_no[, day_start])
```

`indaytotime()` 函数根据时间戳值在一天中的某段时间内出现的时间返回布尔值结果。该段的开始边界是一天的开始，默认设置为午夜；可以通过 `indaytotime()` 函数的 `day_start` 参数修改一天的开始。日段的结束边界由函数的 `base_timestamp` 参数确定。

indaytotime 函数的图表。



适用场景

indaytotime() 函数返回布尔值结果。通常，这种类型的函数将用作 if expression 中的条件。indaytotime() 函数返回一个聚合或计算，具体取决于时间戳是否发生在截至并包括基本时间戳时间的时间段内。

例如，该 indaytotime() 函数可用于显示迄今为止已进行的演出的门票销售总额。

返回数据类型： 布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

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，可在 day_start 中指定一个偏移作为某日内时间的小数。例如，使用 0.125 表示凌晨 3 点。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
indaytotime ('01/12/2006 12:23:00 PM', '01/12/2006 11:59:00 PM', 0)	返回 True
indaytotime ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0)	返回 False
indaytotime '01/11/2006 12:23:00 PM', '01/12/2006 11:59:00 PM', -1)	返回 True

示例 1 – 没有其他参数

加载脚本和结果

概述

打开 数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 包含 1 月 4 日至 5 日期间的一组事务的数据集被载入名为 'Transactions' 的表格。
- 以 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT) 格式提供的日期字段。
- 一种前置 Load, 其中包含设置为 'in_day_to_time' 的 indaytotime() 函数, 该字段确定每个事务是否在早上 9:00 之前发生。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        indaytotime(date,'01/05/2022 9:00:00 AM',0) as in_day_to_time
        ;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
```

```
8202,'01/05/2022 11:09:09 PM',95.93
];
```

结果

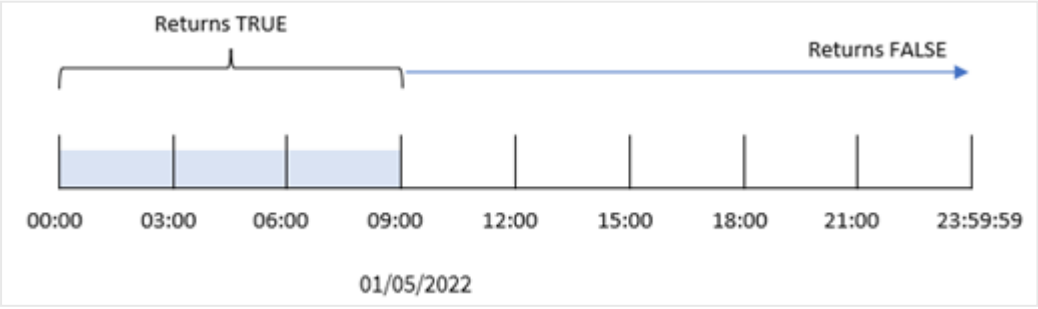
加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_day_to_time

结果表

日期	in_day_to_time
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	-1
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

示例 1 上午 9:00 限制的 indaytotime 函数图。



in_day_to_time field 字段是在前置 Load 语句中使用 indaytotime() 函数创建的，并将日期字段、1 月 5 日 9:00 AM 的硬编码时间戳和为 0 的偏移作为函数的参数传递。1 月 5 日午夜至上午 9:00 之间发生的任何交易都将返回 TRUE。

示例 2 – period_no

加载脚本和结果

概述

加载脚本使用与第一个示例中相同的数据集和场景。

但是，在本例中，您将计算交易日期是否发生在 1 月 5 日上午 9:00 前的一天。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        indaytotime(date,'01/05/2022 9:00:00 AM', -1) as in_day_to_time
        ;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

结果

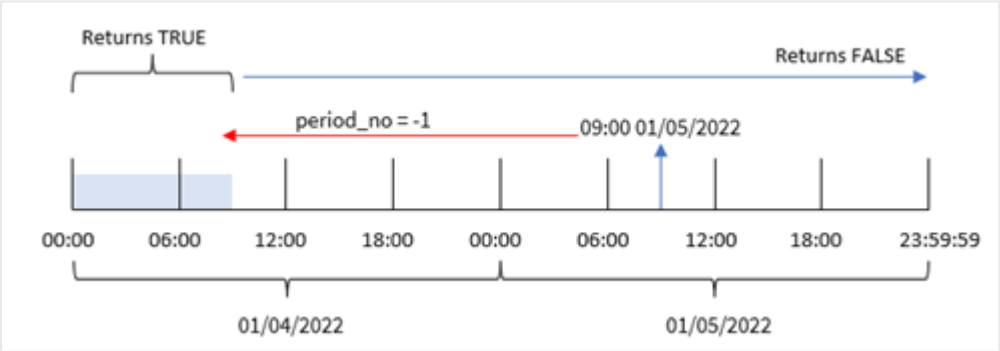
加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_day_to_time

结果表

日期	in_day_to_time
01/04/2022 3:41:54 AM	-1
01/04/2022 4:19:43 AM	-1
01/04/2022 04:53:47 AM	-1
01/04/2022 8:38:53 AM	-1
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	0
01/05/2022 8:44:36 AM	0
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

示例 2:1月 4 日起交易的 indaytotime 函数图表。



在本例中，由于 indaytotime() 函数中使用了 -1 的偏移作为偏移参数，因此函数确定每个交易日期是否发生在 1 月 4 日早上 9 点之前。这可以在输出表中进行验证，其中一个交易返回的布尔结果为 TRUE。

示例 3 – day_start

加载脚本和结果

概述

使用与第一个示例相同的数据集和场景。

然而，在本例中，公司的政策是工作日从早上 8 点开始，到早上 8 点结束。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        indaytotime(date,'01/05/2022 9:00:00 AM', 0,8/24) as in_day_to_time
        ;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

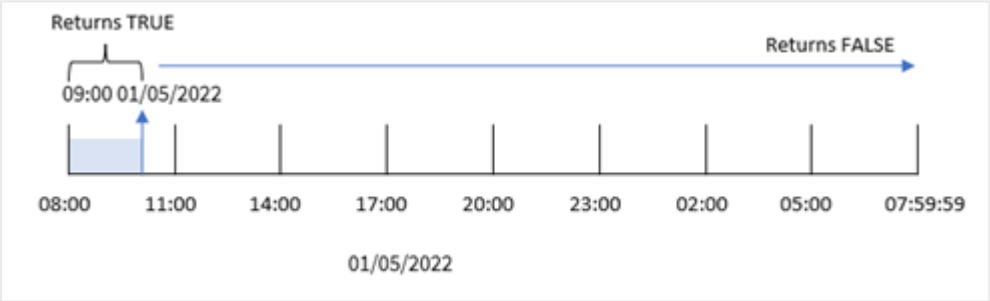
- date
- in_day_to_time

结果表

日期	in_day_to_time
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0

日期	in_day_to_time
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	0
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

示例 3:早上 8:00 至早上 9:00 的交易处理 indaytotime 函数图。



因为indaytotime() 函数中使用了 8/24 的 start_day 参数，相当于早上 8:00，因此每天的开始和结束时间都是早上 8:00。因此，对于 1 月 5 日早上 8:00 到 9:00 之间发生的任何事务，indaytotime() 函数将返回布尔结果 TRUE。

示例 4–图表对象(图表)

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。您将通过在图表对象中创建度量来计算以确定交易是否在 1 月 5 日早上 9:00 之前发生。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
```

```
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

`date`。

要确定交易是否在 1 月 5 日早上 9:00 之前发生，请创建以下度量：

```
=indaytotime(date,'01/05/2022 9:00:00 AM',0)
```

结果表

日期	=indaytotime(date,'01/05/2022 9:00:00 AM',0)
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	-1
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

in_day_to_time 度量是在图表对象中使用 indaytotime() 函数创建的，并将日期字段、1 月 5 日 9:00 AM 的硬编码时间戳和为 0 的偏移作为函数的参数传递。1 月 5 日午夜至上午 9:00 之间发生的任何交易都将返回 TRUE。这在结果表中得到验证。

示例 5 – 场景

加载脚本和结果

概述

在本例中，包含本地影院门票销售的数据集加载到名为 Ticket_Sales 的表中。今天是 2022 年 5 月 3 日时间是上午 11:00。

用户希望 KPI 图表对象显示迄今为止所有节目的收入。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Ticket_Sales:
Load
*
Inline
[
sale ID, show time, ticket price
1,05/01/2022 09:30:00 AM,10.50
2,05/03/2022 05:30:00 PM,21.00
3,05/03/2022 09:30:00 AM,10.50
4,05/03/2022 09:30:00 AM,31.50
5,05/03/2022 09:30:00 AM,10.50
6,05/03/2022 12:00:00 PM,42.00
7,05/03/2022 12:00:00 PM,10.50
8,05/03/2022 05:30:00 PM,42.00
9,05/03/2022 08:00:00 PM,31.50
10,05/04/2022 10:30:00 AM,31.50
11,05/04/2022 12:00:00 PM,10.50
12,05/04/2022 05:30:00 PM,10.50
13,05/05/2022 05:30:00 PM,21.00
14,05/06/2022 12:00:00 PM,21.00
15,05/07/2022 09:30:00 AM,42.00
16,05/07/2022 10:30:00 AM,42.00
17,05/07/2022 10:30:00 AM,10.50
18,05/07/2022 05:30:00 PM,10.50
19,05/08/2022 05:30:00 PM,21.00
20,05/11/2022 09:30:00 AM,10.50
];
```

结果

进行以下操作：

1. 创建 KPI 对象。
2. 使用 indaytotime() 函数创建一个度量，该度量将显示到目前为止已进行的演出的所有门票销售的总和：

```
=sum(if(indaytotime([show time],'05/03/2022 11:00:00 AM',0),[ticket price],0))
```

3. 为 KPI 对象 'Current Revenue' 创建标签。
4. 将度量的**数字格式**设置为**金额**。

截至 2022 年 5 月 3 日上午 11:00，门票销售总额为 52.50 美元。

indaytotime () 函数在将每次售票的放映时间与当前时间 ('05/03/2022 11:00:00 AM') 进行比较时返回一个布尔值。对于 5 月 3 日上午 11:00 之前的任何节目，indaytotime () 函数返回布尔值 TRUE，其票价将包含在总和中。

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 中的农历周定义为将 1 月 1 日计算为一周的第一天。除了一年中的最后一周外，每周都会有七天。

语法：

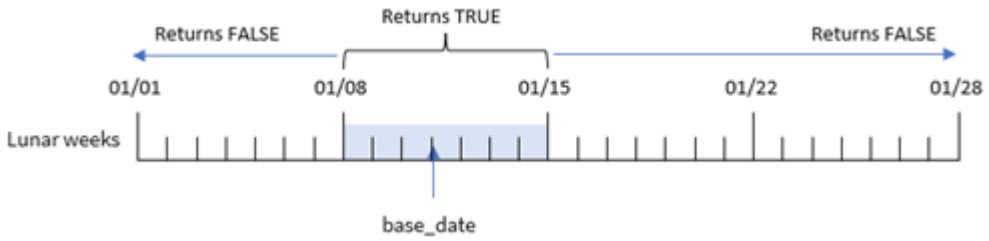
```
InLunarWeek (timestamp, base_date, period_no[, first_week_day])
```

返回数据类型：布尔值

> *在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。*

inlunarweek() 函数决定 base_date 属于哪个农历周。一旦确定每个时间戳值是否与 base_date 发生在同一个农历周内，它就会返回布尔值结果。

inlunarweek() 函数的图表



适用场景

inlunarweek() 函数返回布尔值结果。通常，这种类型的函数将用作 IF 表达式中的条件。这将返回聚合或计算结果，具体取决于评估的日期是否发生在所讨论的农历周。

例如，inlunarweek() 函数可用于识别特定农历周内制造的所有设备。

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。

函数示例

示例	结果
`inlunarweek ('01/12/2013', '01/14/2013', 0)`	由于 `timestamp`、01/12/2013 的值，属于 01/08/2013 至 01/14/2013 的周之内，返回 `TRUE`。
`inlunarweek ('01/12/2013', '01/07/2013', 0)`	由于 `base_date` 01/07/2013 处于定义为 01/01/2013 至 01/07/2013 的农历周，返回 `FALSE`。
`inlunarweek ('01/12/2013', '01/14/2013', -1)`	返回 `FALSE`。将 period_no 的值指定为 -1，表示将周移动到前一周，即 01/01/2013 到 01/07/2013。
`inlunarweek ('01/07/2013', 01/14/2013', -1)`	返回 `TRUE`。与前面的示例相比，`timestamp` 是在考虑到向后移动后的后续周内。
`inlunarweek ('01/11/2006', '01/08/2006', 0, 3)`	返回 `FALSE`。为 `first_week_day` 指定值 3 表示从 01/04/2013 开始计算年初。因此，`base_date` 的值在第一周下降，而 `timestamp` 的值在周 01/11/2013 内下降到 01/17/2013。

`inlunarweek()` 函数通常与以下功能结合使用：

相关函数

函数	交互
lunarweekname (page 819)	此函数用于确定输入日期所在年份的农历周数。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 1 月交易的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

创建一个字段 in_lunar_week，确定交易是否发生在 1 月 10 日的同一个农历周。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 0) as in_lunar_week
        ;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
```

```
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

结果

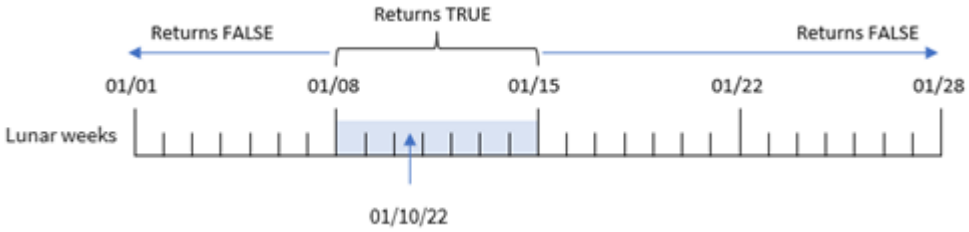
加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_lunar_week

结果表

日期	in_lunar_week
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	-1
1/14/2022	-1
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	0
1/23/2022	0

inlunarweek() 函数，基本示例



in_lunar_week 字段是在前置 Load 语句中创建的，方法是使用 inlunarweek() 函数，然后将以下内容作为函数的参数传递：

- date 字段
- 1 月 10 日的硬编码日期，作为 base_date
- 为 0 的period_no

因为农历周从 1 月 1 日开始，所以 1 月 10 日会在 1 月 8日开始、1 月 14 日结束的农历周中。因此，在 1 月份这两个日期之间发生的任何交易都将返回布尔值 TRUE。这在结果表中得到验证。

示例 2 – period_no

和结果：

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 日期字段以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

然而，在本例中，任务是创建一个字段 2_lunar_weeks_later，用于确定交易是否发生在 1 月 10 日之后的两个农历周。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 2) as [2_lunar_weeks_later]
        ;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
```

```
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

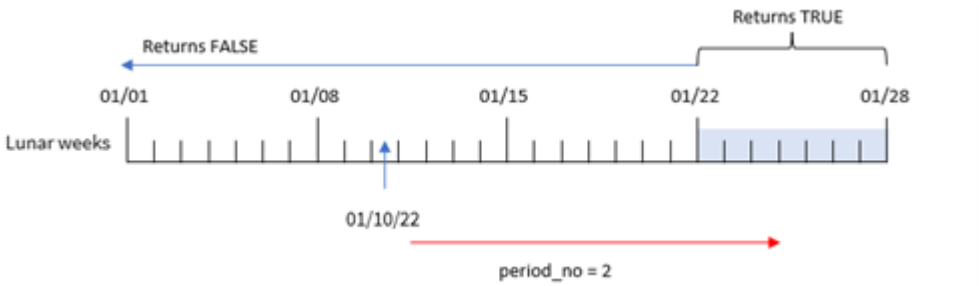
- date
- 2_lunar_weeks_later

结果表

日期	2_lunar_weeks_later
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	0
1/9/2022	0
1/10/2022	0
1/11/2022	0
1/12/2022	0
1/13/2022	0
1/14/2022	0
1/15/2022	0
1/16/2022	0
1/17/2022	0

日期	2_lunar_weeks_later
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	-1
1/23/2022	-1

inlunarweek() 函数，period_no 示例



在本例中，由于 inlunarweek() 函数中使用了 2 的 period_no作为偏移参数，因此该函数将从1月22日开始的一周定义为用于验证事务的农历周。因此，2020 年 1 月 1 日至 7 月 26 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本使用与第一个示例相同的数据集和场景。然而，在这个例子中，我们将农历周设置为 1 月 6 日开始。

- 与第一个示例相同的数据集和场景。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。
- first_week_day 参数为 5。这将设定农历周从 1 月 5 日开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 0,5) as in_lunar_week
```

```
        ;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

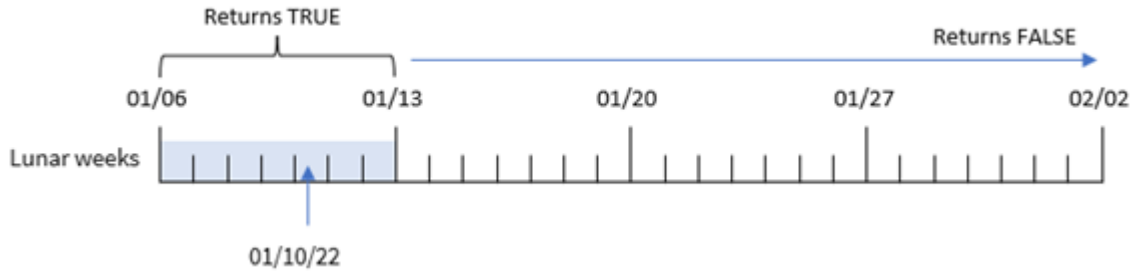
- `date`
- `in_lunar_week`

结果表

日期	in_lunar_week
1/5/2022	0
1/6/2022	-1
1/7/2022	-1
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	0

日期	in_lunar_week
1/14/2022	0
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	0
1/23/2022	0

inlunarweek() 函数，first_week_day 示例



在本例中，由于 inlunarweek() 函数中使用了 first_week_date 参数，因此它将农历周的开始日期偏移到 1 月 6 日。因此，1 月 10 日属于从 1 月 6 日开始到 1 月 12 日结束的农历周。这两个日期之间的任何事务都将返回布尔值 TRUE。

示例 4 – 图表对象

加载脚本和图表表达式：

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 日期字段以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

然而，在本例中，未更改的数据集被加载到应用程序中。确定交易是否与 1 月 10 日发生在同一个农历周的计算被创建为应用程序图表对象中的度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：`date`。

要计算交易是否发生在包含 1 月 10 日的农历周，请创建以下度量：

```
= inlunarweek(date,'01/10/2022', 0)
```

结果表

日期	=inlunarweek(date,'01/10/2022', 0)
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1

日期	=inlunarweek(date,'01/10/2022', 0)
1/11/2022	-1
1/12/2022	-1
1/13/2022	-1
1/14/2022	-1
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	0
1/23/2022	0

示例 5 – 场景

加载脚本和图表表达式:

概述

打开数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 加载到名为 Products 的表中的数据集。
- 包含产品 ID、制造日期和成本价格的信息。

现已查明, 由于设备错误, 包括 1 月 12 日在内的农历周制造的产品存在缺陷。最终用户希望使用一个图表对象, 通过农历周名称显示所生产产品的“缺陷”或“无缺陷”状态以及当月生产的产品的成本。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8183,'1/5/2022',42.32
```

```
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 创建维度以显示月名称：
   `=lunarweekname(manufacture_date)`
3. 使用 `inlunarweek()` 函数创建一个度量，以识别哪些产品有缺陷，哪些无缺陷：
   `=if(only(inlunarweek(manufacture_date,makedate(2022,01,12),0)), 'Defective','Faultless')`
4. 创建一个度量来求对产品的 `cost_price` 求和：
   `=sum(cost_price)`
5. 将度量的**数字格式**设置为**金额**。
6. 在**外观**下，关闭**总计**。

结果表

lunarweekname (manufacture_date)	=if(only(inlunarweek(manufacture_ date,makedate(2022,01,12),0)), 'Defective','Faultless')	sum(cost_ price)
2022/01	Faultless	$125.79
2022/02	Defective	$316.38
2022/03	Faultless	$455.75
2022/04	Faultless	$146.09

`inlunarweek()` 函数在评估每个产品的制造日期时返回布尔值。对于在农历周生产的包含 1 月 10 日的任何产品，`inlunarweek()` 函数返回布尔值 `TRUE`，并将产品标记为“缺陷”。对于任何返回 `FALSE`值的产品，由于不是在该周制造的，它将产品标记为 ‘Faultless’。

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 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

语法：

```
InLunarWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

返回数据类型： 布尔值

> *在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。*

inlunarweektodate() 函数的示例图表

inlunarweektodate() 函数作为农历周的终点。相比之下，inlunarweek() 函数决定 base_date 属于哪个农历周。例如，如果 base_date 是 1 月 5 日，那么 1 月 1 日到 1 月 5 号之间的任何时间戳都将返回布尔值结果 TRUE，而 1 月 6 日和 7 日以及之后的日期将返回布尔值结果 FALSE。

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。

适用场景

inlunarweektodate() 函数返回布尔值结果。通常，这种类型的函数将用作 IF 表达式中的条件。inlunarweektodate() 函数将在用户希望计算返回聚合或计算时使用，具体取决于评估日期是否发生在所讨论的一周的特定时段。

例如，inlunarweektodate() 函数可用于识别特定日期之前(包括特定日期)在特定周内制造的所有设备。

函数示例

示例	结果
inlunarweektodate ('01/12/2013', '01/13/2013', 0)	由于 timestamp、01/12/2013 的值，属于 01/08/2013 至 01/13/2013 的周之内，返回 TRUE。
inlunarweektodate ('01/12/2013', '01/11/2013', 0)	由于 timestamp 的值晚于 base_date 的值，尽管这两个日期都属于 01/12/2012 之前的同一阴历周，返回 FALSE。
inlunarweektodate ('01/12/2006', '01/05/2006', 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。

inlunarweektodate() 函数通常与以下功能结合使用：

相关函数

函数	交互
lunarweekname (page 819)	此函数用于确定输入日期所在年份的农历周数。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 1 月交易集的数据集，该数据集加载到名为 Transactions 的表中。使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 创建一个字段 in_lunar_week_to_date，用于确定截至 1 月 10 日的农历周中发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inlunarweektodate(date,'01/10/2022', 0) as in_lunar_week_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_lunar_week_to_date

结果表

date	in_lunar_week_to_date
1/1/2022	0
1/4/2022	0
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0

date	in_lunar_week_to_date
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 函数，没有其他参数

in_lunar_week_to_date 字段是在前置 Load 语句中使用 inlunarweektodate() 函数创建的，并将 date 字段、作为我们 base_date 的 1 月 10 日的硬编码日期和 0 的偏移值作为函数的参数传递。

因为农历周从 1 月 1 日开始，所以 1 月 10 日正好是从 1 月 8 日开始的农历周；因为我们正在使用 inlunarweektodate() 函数，所以农历周将在 10 号结束。因此，在 1 月份这两个日期之间发生的任何交易都将返回布尔值 TRUE。这在结果表中得到验证。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，任务是创建一个字段 2_lunar_weeks_later，用于确定交易是否发生在日期 1 月 1 日的农历周后两周。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inlunarweektodate(date,'01/10/2022', 2) as [2_lunar_weeks_later]
```

```
        ;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

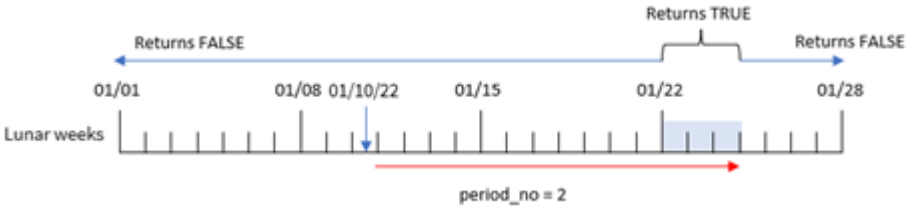
- date
- 2_lunar_weeks_later

结果表

date	2_lunar_weeks_later
1/1/2022	0
1/4/2022	0
1/10/2022	0
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	-1

date	2_lunar_weeks_later
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 函数，period_no 示例



在这种情况下，inlunarweektodate() 函数确定截至 1 月 10 日的农历周等于三天(1 月 8 日、9 日、10 日)。由于使用了值为 2 的 period_no 作为偏移参数，因此本农历周偏移了 14 天。因此，这定义了三天的农历周以包括 1 月 22 日、23 日和 24 日。1 月 22 日至 1 月 24 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。
- first_week_date 参数为 3。这将设定农历周从 1 月 3 日开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 0,3) as in_lunar_week_to_date
        ;
Load
*
Inline
[
```

```
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `in_lunar_week_to_date`

结果表

date	in_lunar_week_to_date
1/1/2022	0
1/4/2022	-1
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 函数，first_week_day 示例

在本例中，因为 inlunarweek() 函数中使用了值为 3 的参数 the first_week_date，所以农历第一周将从 1 月 3 日到 1 月 10 日。由于 1 月 10 日也是 base_date，这两个日期之间的任何交易都将返回布尔值 TRUE。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。确定交易是否发生在 1 月 10 日之前的农历周的计算被创建为应用程序图表对象中的度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
```

```
8202,'1/4/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

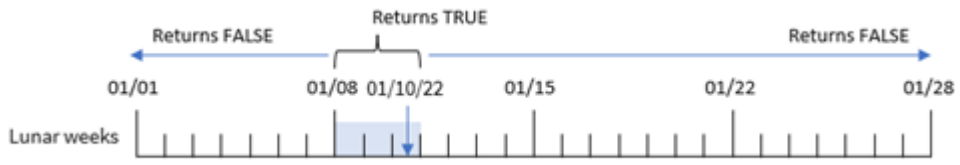
创建以下度量：

```
=inlunarweektodate(date,'01/10/2022', 0)
```

结果表

date	=inlunarweektodate(date,'01/10/2022', 0)
1/1/2022	0
1/4/2022	0
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 函数，图表对象示例



in_lunar_week_to_date 度量是在图表对象中使用 inlunarweektodate() 函数创建的，并将日期字段、作为我们的 base_date 的 1 月 10 日的硬编码日期和 0 的偏移值作为函数的参数传递。

因为农历周从 1 月 1 日开始，所以 1 月 10 日正好是从 1 月 8 日开始的农历周。此外，因为我们正在使用 inlunarweektodate() 函数，所以农历周将在 10 号终止。因此，在 1 月份这两个日期之间发生的任何交易都将返回布尔值 TRUE。这在结果表中得到验证。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Products 的表中的数据集。
- 包含产品 ID、制造日期和成本价格的信息。

已确定，由于设备错误，1 月 12 日的农历周内生产的产品存在缺陷。该问题于 1 月 13 日得到解决。最终用户想要一个图表对象，它按周显示生产的产品是“有缺陷”还是“无缺陷”的状态，以及该周生产的产品的成本。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';

Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'01/02/2022 12:22:06',37.23
8189,'01/05/2022 01:02:30',17.17
8190,'01/06/2022 15:36:20',88.27
8191,'01/08/2022 10:58:35',57.42
8192,'01/09/2022 08:53:32',53.80
8193,'01/10/2022 21:13:01',82.06
8194,'01/11/2022 00:57:13',40.39
8195,'01/12/2022 09:26:02',87.21
8196,'01/13/2022 15:05:09',95.93
8197,'01/14/2022 18:44:57',45.89
8198,'01/15/2022 06:10:46',36.23
8199,'01/16/2022 06:39:27',25.66
8200,'01/17/2022 10:44:16',82.77
8201,'01/18/2022 18:48:17',69.98
8202,'01/26/2022 04:36:03',76.11
8203,'01/27/2022 08:07:49',25.12
8204,'01/28/2022 12:24:29',46.23
8205,'01/30/2022 11:56:56',84.21
8206,'01/30/2022 14:40:19',96.24
8207,'01/31/2022 05:28:21',67.67
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 创建维度以显示周名称：
   `=weekname(manufacture_date)`
3. 接下来，创建一个维度，其使用 `inlunarweektodate()` 函数识别哪些产品有缺陷，哪些无缺陷：
   `=if(inlunarweektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')`
4. 创建一个度量来求对产品的 `cost_price` 求和：
   `=sum(cost_price)`
5. 将度量的**数字格式**设置为**金额**。

结果表

=lunarweekname (manufacture_date)	=if(InLunarWeekToDate(manufacture_date,makedate (2022,01,12),0),'Defective','Faultless')	=Sum(cost_price)
2022/01	Faultless	$142.67
2022/02	Defective	$320.88
2022/02	Faultless	$141.82
2022/03	Faultless	$214.64
2022/04	Faultless	$147.46
2022/05	Faultless	$248.12

`inlunarweektodate()` 函数在评估每个产品的制造日期时返回布尔值。对于返回布尔值 `TRUE` 的产品，它将产品标记为 `'Defective'`。对于返回值为 `FALSE` 的任何产品（因此在截至 1 月 12 日的农历周内未生产），它将产品标记为 `'Faultless'`。

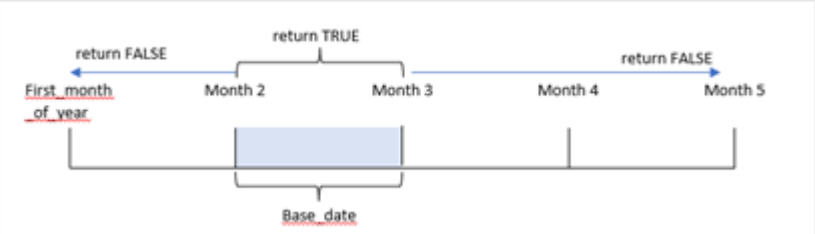
inmonth

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)
```

indaytotime 函数的图表。



换言之，inmonth() 函数确定一组日期是否属于该月，并基于标识该月的 base_date 返回一个布尔值。

适用场景

inmonth() 函数返回布尔值结果。通常，这种类型的函数将用作 if expression 中的条件。这将返回一个聚合或计算，具体取决于某个日期是否发生在该月，包括所讨论的日期。

例如，inmonth() 函数可用于识别特定月份内制造的所有设备。

返回数据类型： 布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

参数

参数	说明
时间戳	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。需要注意的是，base_date 可以是一个月内的任何一天。
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。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
inmonth ('25/01/2013', '01/01/2013', 0)	返回 True
inmonth('25/01/2013', '23/04/2013', 0)	返回 False
inmonth ('25/01/2013', '01/01/2013', -1)	返回 False
inmonth ('25/12/2012', '17/01/2013', -1)	返回 True

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年上半年一组交易的数据集。
- 带有附加变量 'in_month' 的前置 Load，该变量确定交易是否发生在 4 月份。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inmonth(date,'04/01/2022', 0) as in_month
    ;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
8197,'4/15/2022',45.89
8198,'4/25/2022',36.23
8199,'5/20/2022',25.66
8200,'5/22/2022',82.77
8201,'6/19/2022',69.98
8202,'6/22/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_month

函数示例

日期	in_month
1/10/2022	0

日期	in_month
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	0
2/2/2022	0
2/20/2022	0
4/11/2022	-1
4/13/2022	-1
4/15/2022	-1
4/25/2022	-1
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

'in_month' 字段的创建方式是在前置 Load 语句中使用 inmonth() 函数并传递日期字段，其为 4 月 1 日的硬编码日期，作为我们的 base_date，并传递为 0 的 period_no，作为函数的参数。

base_date 标识将返回布尔结果为 TRUE 的月份。因此，4 月份发生的所有交易都返回 TRUE，这在结果表中得到验证。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个示例相同的数据集和场景。

但是，在本例中，您将创建一个字段 '2_months_prior'，用于确定交易是否发生在 4 月前两个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
   *,
   inmonth(date,'04/01/2022', -2) as [2_months_prior]
Inline
[
id,date,amount
8188,'1/10/2022',37.23
```

```
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
8197,'4/15/2022',45.89
8198,'4/25/2022',36.23
8199,'5/20/2022',25.66
8200,'5/22/2022',82.77
8201,'6/19/2022',69.98
8202,'6/22/2022',76.11
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `2_months_prior`

函数示例

日期	2_months_prior
1/10/2022	0
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	-1
2/2/2022	-1
2/20/2022	-1
4/11/2022	0
4/13/2022	0
4/15/2022	0
4/25/2022	0
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

在 `inmonth()` 函数中使用 `-2` 作为 `period_no` 参数将 `base_date` 参数定义的月份移到之前两个月。在本例中，它将定义的月份从 4 月更改为 2 月。

因此，二月份发生的任何交易都将返回布尔结果 TRUE。

示例 3 – 图表对象

加载脚本和图表表达式

概述

使用与前一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。确定交易是否发生在 4 月的计算是作为应用程序的图表对象中的度量创建的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
8197,'4/15/2022',45.89
8198,'4/25/2022',36.23
8199,'5/20/2022',25.66
8200,'5/22/2022',82.77
8201,'6/19/2022',69.98
8202,'6/22/2022',76.11
];
```

图表对象

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
date
```

要计算交易是否在 4 月发生，请创建以下度量：

```
=inmonth(date,'04/01/2022', 0)
```

结果

	函数示例
日期	**=inmonth(date,'04/01/2022', 0)**
1/10/2022	0
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	0
2/2/2022	0
2/20/2022	0
4/11/2022	-1
4/13/2022	-1
4/15/2022	-1
4/25/2022	-1
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

示例 4 – 场景

加载脚本和结果

概述

在本例中，将数据集加载到名为 'Products' 的表中。表格包含以下字段：

- 产品 ID
- 制造日期
- 成本价

由于设备错误，2022 年 7 月制造的产品存在缺陷。该问题于 2022 年 7 月 27 日得到解决。

最终用户需要一个图表，按月显示制造为“有缺陷”(布尔值为 TRUE) 或“无缺陷”(布尔值为 FALSE) 的产品的状态，以及当月制造的产品的成本。

加载脚本

```
Products:
Load
*
Inline
```

```
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
=monthname(manufacture_date)
```

创建以下度量

- `=sum(cost_price)`
- `=if(only(inmonth(manufacture_date,makedate(2022,07,01),0)),'Defective','Faultless')`

1. 将度量的**数字格式**设置为**金额**。
2. 在**外观**下，关闭**总计**。

结果表

monthname (manufacture_date)	=if(only(inmonth(manufacture_date,makedate (2022,07,01),0)),'Defective','Faultless')	sum(cost_ price)
Jan 2022	Faultless	$54.40
Feb 2022	Faultless	$145.69
Mar 2022	Faultless	$53.80
Apr 2022	Faultless	$82.06
May 2022	Faultless	$127.60
Jun 2022	Faultless	$141.82

monthname (manufacture_date)	=if(only(inmonth(manufacture_date,makedate (2022,07,01),0)),'Defective','Faultless')	sum(cost_ price)
Jul 2022	Defective	$214.64
Aug 2022	Faultless	$147.46
Sep 2022	Faultless	$84.21
Oct 2022	Faultless	$163.91

inmonth() 函数在评估每个产品的制造日期时返回布尔值。对于 2022 年 7 月生产的任何产品，inmonth() 函数返回布尔值 True，并将产品标记为 'Defective'。对于任何返回 False 值的产品，由于不是在 7 月制造的，它将产品标记为 'Faultless'。

inmonths

此函数用于查找时间戳是否在作为基准日期的同一个月、两个月、季度、四个月期间或半年内。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。

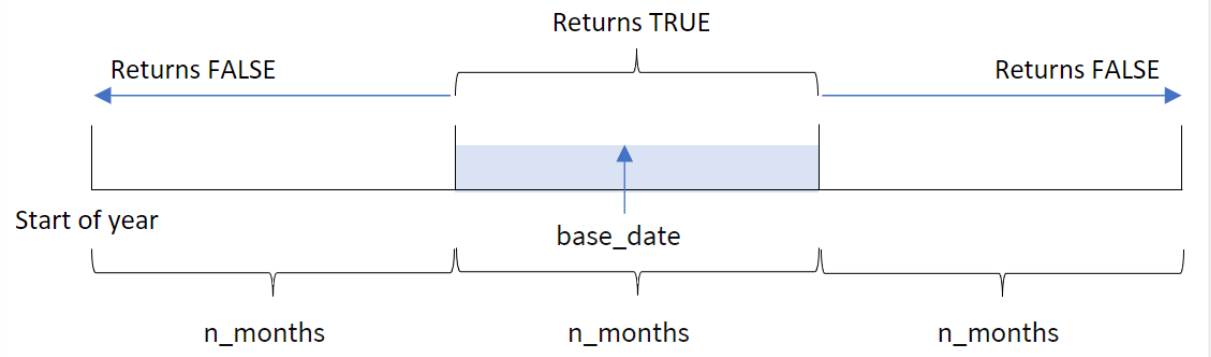
语法：

```
InMonths(n_months, timestamp, base_date, period_no [, first_month_of_year])
```

返回数据类型：布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

inmonths() 函数的图表



inmonths() 函数根据提供的 n_months 参数将一年划分为若干段。然后，它确定评估的每个时间戳是否与 base_date参数属于同一段。但是，如果提供了 period_no 参数，该函数将确定时间戳是属于 base_date 的上一个时段还是下一个时段。

函数中提供了一年中的以下时段作为 n_month 参数。

n_month 参数

期间	月数
月	1

期间	月数
双月	2
季	3
四个月	4
半年	6

适用场景

inmonths() 函数返回布尔值结果。通常，这种类型的函数将用作 if expression 中的条件。通过使用 inmonths() 函数，您可以选择要评估的期间。例如，让用户识别在某一期间的月份、季度或半年内生产的产品。

返回数据类型： 布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

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(四个月期间)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。

可以使用以下值在 first_month_of_year 参数中设置一年中的第一个月：

first_month_of_year 值

月	值
二月	2
三月	3
四月	4
五月	5
六月	6

月	值
七月	7
八月	8
九月	9
十月	10
十一月	11
十二月	12

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
inmonths(4, '01/25/2013', '04/25/2013', 0)	返回 TRUE。因为 时间戳，01/25/2013，在四个月内 01/01/2013 到 04/30/2013，其中 base_date 的值 04/25/2013 就位于其中。
inmonths(4, '05/25/2013', '04/25/2013', 0)	返回 FALSE。因为 05/25/2013 与前一个示例不在同一时间段内。
inmonths(4, '11/25/2012', '02/01/2013', -1)	返回 TRUE。因为 period_no 的值 -1，将搜索周期向后移动四个月（值为 n 个月），这表示搜索期间 09/01/2012 到 12/31/2012.
inmonths(4, '05/25/2006', '03/01/2006', 0, 3)	返回 TRUE。因为 first_month_of_year 设置为 3，这就是搜索期间 03/01/2006 到 07/30/2006 而不是 01/01/2006 到 04/30/2006。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 带有附加变量 'in_months' 的前置 Load，用于确定哪些交易发生在 2022 年 5 月 15 日的同一季度。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       inmonths(3,date,'05/15/2022', 0) as in_months
       ;
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

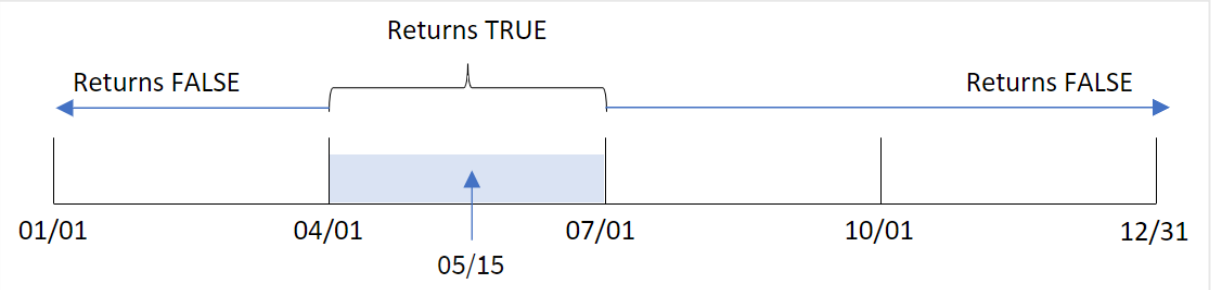
- date
- in_months

结果表

日期	in_months
2/19/2022	0
3/7/2022	0
3/30/2022	0
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

'in_months'" 字段是在前置 Load 语句中使用 `inmonths()` 函数创建的。提供的第一个参数是 `3`，其将一年划分为季度分段。第二个参数标识要计算的字段，本例中为日期字段。第三个参数是 5 月 15 日的硬编码日期，其为 `base_date` 并且为 `0` 的 `period_no` 是最后一个参数。

带季度段的 `inmonths()` 函数的图表



五月是一年的第二季度。因此，4 月 1 日至 6 月 30 日之间发生的任何交易都将返回布尔值结果 TRUE。这在结果表中得到验证。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 带有附加变量 'previous_quarter' 的前置 Load，用于确定交易是否发生在 2022 年 5 月 15 日之前的季度。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       inmonths(3,date,'05/15/2022', -1) as previous_quarter
       ;
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

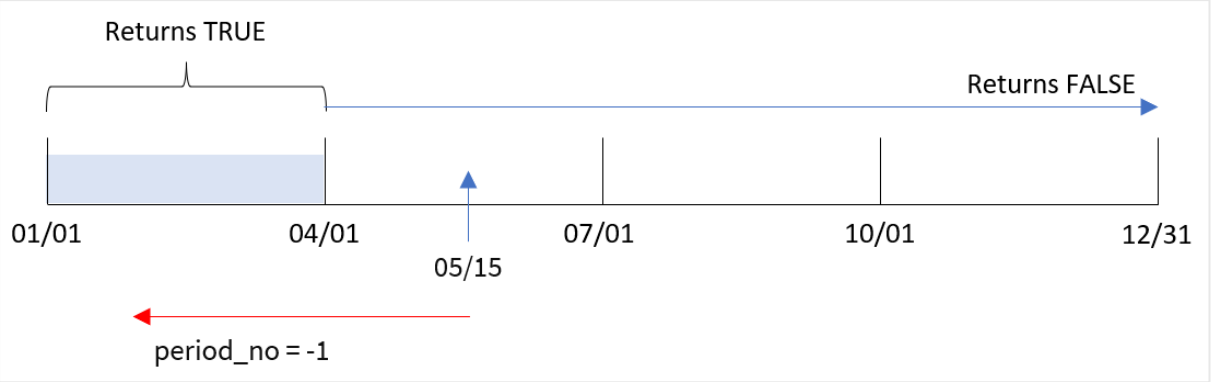
- date
- previous_quarter

结果表

日期	上一季度
2/19/2022	-1
3/7/2022	-1
3/30/2022	-1
4/5/2022	0
4/16/2022	0
5/1/2022	0
5/7/2022	0
5/22/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

该函数将 -1 用作 inmonths() 函数中的 period_no 参数，以评估交易是否发生在一年的第一季度。5 月 15 日是 base_date，属于一年的第二季度（4 月至 6 月）。

带季度段并且 period_no 设置为 -1 的 inmonths() 函数的图表



因此，1 月和 3 月之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 带有附加变量 'in_months' 的前置 Load，用于确定哪些交易发生在 2022 年 5 月 15 日的同一季度。

在本例中，组织策略是将三月作为财政年度的第一个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       inmonths(3,date,'05/15/2022', 0, 3) as in_months
       ;
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
```

```
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `in_months`

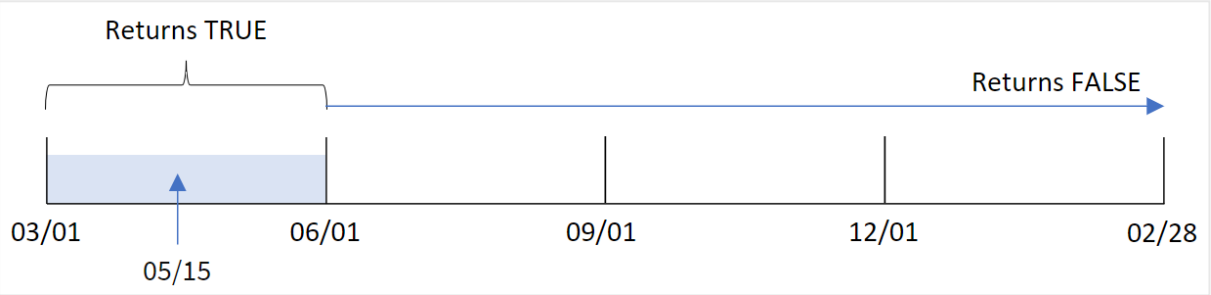
结果表

日期	in_months
2/19/2022	0
3/7/2022	-1
3/30/2022	-1
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

日期	in_months
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过在 inmonths() 函数中使用 3 作为 first_month_of_year 参数，函数从 3 月 1 日开始一年。然后 inmonths() 函数将一年分成四个季度。3 月 - 5 月、6 月 - 8 月、9 月 - 11 月、12 月 - 2 月。因此，5 月 15 日属于一年的第一季度(3 月 - 5 月)。

inmonths() 函数的图表，3 月为一年中的第一个月。



这几个月内发生的任何交易都将返回布尔值结果 TRUE。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

使用与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。确定交易是否发生在 2022 年 5 月 15 日的同一季度的计算是在应用程序的图表中创建的一个度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
```

```
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- date

要计算交易是否与 5 月 15 日发生在同一季度，请创建以下度量：

```
=inmonths(3,date,'05/15/2022', 0)
```

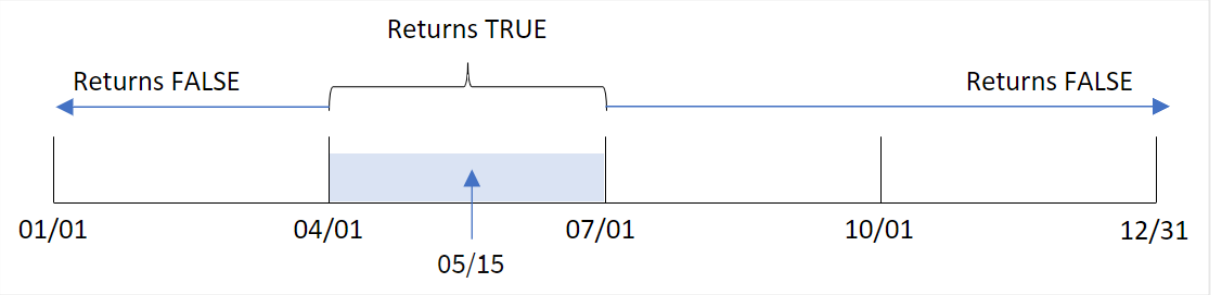
结果表

日期	=inmonths(3,date,'05/15/2022', 0)
2/19/2022	0
3/7/2022	0
3/30/2022	0
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0

日期	=inmonths(3,date,'05/15/2022', 0)
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

使用 inmonths() 函数在图表中创建 'in_months' 字段。提供的第一个参数是 3，其将一年划分为季度分段。第二个参数标识要计算的字段，本例中为日期字段。第三个参数是 5 月 15 日的硬编码日期，其为 base_date 并且为 0 的 period_no 是最后一个参数。

带季度段的 inmonths() 函数的图表



五月是一年的第二季度。因此，4 月 1 日至 6 月 30 日之间发生的任何交易都将返回布尔值结果 TRUE。这在结果表中得到验证。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 'Products' 的表中的数据集。
- 表格包含以下字段：
  - 产品 ID
  - 产品类型
  - 制造日期
  - 成本价

最终用户想要一个图表，按产品类型显示 2021 年第一阶段制造的产品的成本。用户希望能够定义此段的长度。

加载脚本

```
SET vPeriod = 1;

Products:
Load
*
Inline
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'2/19/2022',37.23
8189,product D,'3/7/2022',17.17
8190,product C,'3/30/2022',88.27
8191,product B,'4/5/2022',57.42
8192,product D,'4/16/2022',53.80
8193,product D,'5/1/2022',82.06
8194,product A,'5/7/2022',40.39
8195,product B,'5/22/2022',87.21
8196,product C,'6/15/2022',95.93
8197,product B,'6/26/2022',45.89
8198,product C,'7/9/2022',36.23
8199,product D,'7/22/2022',25.66
8200,product D,'7/23/2022',82.77
8201,product A,'7/27/2022',69.98
8202,product A,'8/2/2022',76.11
8203,product B,'8/8/2022',25.12
8204,product B,'8/19/2022',46.23
8205,product B,'9/26/2022',84.21
8206,product C,'10/14/2022',96.24
8207,product D,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。

在加载脚本开始时，创建了一个变量 `vPeriod`，该变量将绑定到变量输入控件。

进行以下操作：

1. 在资产面板中，单击**自定义对象**。
2. 选择 **Qlik 仪表板捆绑**，并创建**变量输入**对象。
3. 输入图表对象的标题。
4. 在**变量**下，选择 **vPeriod** 作为名称，并将对象设置为显示为**下拉列表**。
5. 在**值**下，单击**动态**值。输入以下内容：
   `='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.`
6. 将新表添加到工作表中。
7. 在属性面板中的**数据**下，将 `product_type` 添加为维度。

8. 添加以下表达式作为度量：
   `=sum(if(inmonths($(vPeriod),manufacture_date,makedate(2022,01,01),0),cost_price,0))`
9. 将度量的**数字格式**设置为**金额**。

结果表

product_type	=sum(if(inmonths($(vPeriod),manufacture_date,makedate (2022,01,01),0),cost_price,0))
product A	$88.27
product B	$37.23
product C	$17.17
product D	$0.00

`inmonths()` 函数使用用户输入作为参数来定义一年起始段的大小。函数将每个产品的生产日期作为 `inmonths()` 函数的第二个参数传递。通过使用 1 月 1 日作为 `inmonths()` 函数中的第三个参数，生产日期位于年初段的产品将返回布尔值 TRUE，因此总和函数将添加这些产品的成本。

inmonthstodate

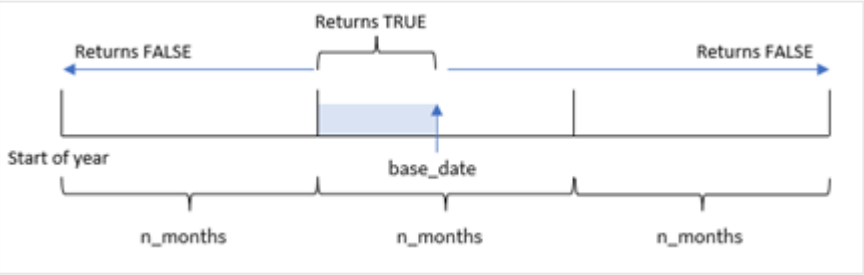
此函数用于判断时间戳是否位于截止以及包括 `base_date` 的最后毫秒的某个月、两个月、季度、四个月期间或半年周期的一部分以内。另外，它也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

语法：

```
InMonths (n_months, timestamp, base_date, period_no[, first_month_of_year ])
```

返回数据类型： 布尔值

`inmonthstodate` 函数的图表。



参数

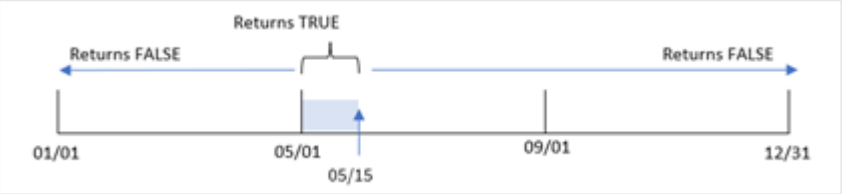
参数	说明
n_months	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1（相当于 inmonth() 函数）、2（两个月）、3（相当于 inquarter()函数）、4（四个月期间）或 6（半年）。
timestamp	想要用来与 **base_date** 进行比较的日期。

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。

在 inmonthstodate() 函数中，base_date 充当它所属的特定年份段的终点。

例如，如果年份被分成三段，并且 base_date 是 5 月 15 日，那么 1 月初到 4 月底之间的任何时间戳都将返回布尔值 FALSE。5 月 1 日至 5 月 15 日之间的日期将返回 TRUE。今年剩余时间将返回 FALSE。

inmonthstodate 函数的布尔值结果范围图。



函数中提供了一年中的以下时段作为 n_month 参数。

n_month 参数

期间	月数
月	1
双月	2
季	3
四个月	4
半年	6

适用场景

inmonthstodate() 函数返回布尔值结果。通常，这种类型的函数用作 if expression 中的条件。通过使用 inmonthstodate() 函数，您可以选择要评估的期间。例如，提供一个输入变量，使用户可以识别在某个期间的某个月、季度或半年到某个日期生产的产品。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
`inmonthstodate(4, '01/25/2013', '04/25/2013', 0)`	返回 True，因为 timestamp、01/25/2013 的值属于 01/01/2013 至 04/25/2013 结束的四个月期间内，base_date、04/25/2013 的值位于其内。
`inmonthstodate(4, '04/26/2013', '04/25/2006', 0)`	返回 False，因为 04/26/2013 与上一示例不在同一期间内。
`inmonthstodate(4, '09/25/2005', '02/01/2006', -1)`	返回 True，因为 period_no、-1 的值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 02/01/2006 之间。
`inmonthstodate(4, '04/25/2006', '06/01/2006', 0, 3)`	返回 True，因为 first_month_of_year 的值设置为 3，这使得搜索周期介于 03/01/2006 至 06/01/2006 之内，而不是 05/01/2006 至 06/01/2006。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 采用 DateFormat 系统变量 (MM/DD/YYYY) 格式的日期字段。
- 前置 Load 语句包含：
  - 设置为字段 'in_months_to_date' 的 inmonthstodate() 函数。这决定了截至 2022 年 5 月 15 日止，本季度发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    inmonthstodate(3,date,'05/15/2022', 0) as in_months_to_date
    ;
Load
*
```

```
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_months_to_date

结果表

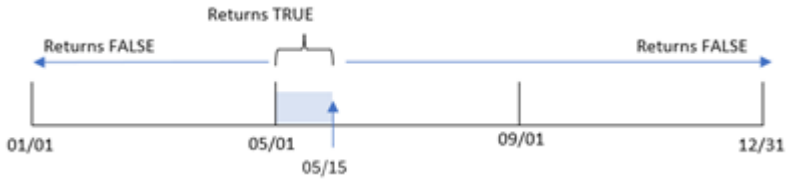
日期	in_months_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0

日期	in_months_to_date
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

in_months_to_date 字段是在前置 Load 语句中使用 inmonthstodate() 函数创建的。

提供的第一个参数是 3，其将一年划分为季度分段。第二个参数标识要计算的字段。第三个参数是 5 月 15 日的硬编码日期，其为 base_date，定义了段的结束边界。0 的 period_no 为最终参数。

inmonthstodate 函数图表，没有额外参数。



4 月 1 日至 5 月 15 日之间发生的任何交易都将返回布尔值结果 TRUE。超出该期间的交易日期返回 FALSE。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，任务是创建一个字段 'previous_qtr_to_date'，用于确定交易是否在 5 月 15 日之前的一个季度发生。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
```

```
    *,
    inmonthstodate(3,date,'05/15/2022', -1) as previous_qtr_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_qtr_to_date

结果表

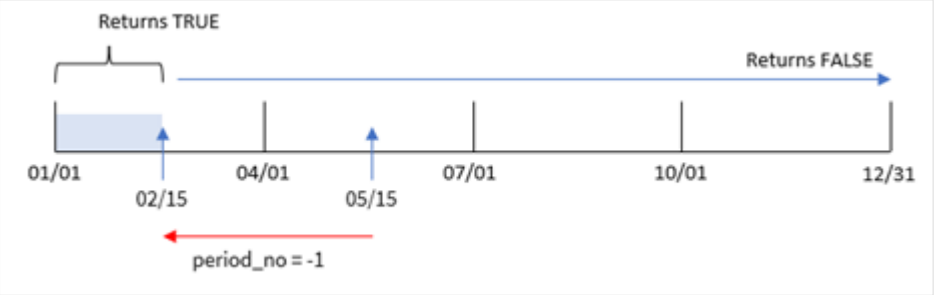
日期	previous_qtr_to_date
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0

日期	previous_qtr_to_date
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过在 inmonthstodate() 函数中将 -1 用作 period_no 参数，该函数将比较器年份段的边界移动了四分之一。

5 月 15 日属于一年的第二季度，因此该部分最初等于 4 月 1 日至 5 月 15 之间。period_no 参数通过负三个月偏移了该段。日期边界为 1 月 1 日至 2 月 15 日。

inmonthstodate 函数的图表，其中 period_no 设置为 -1.



因此，1 月 1 日至 2 月 15 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_month_of_year

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

在本例中，组织策略是将三月作为财政年度的第一个月。

创建一个字段 'in_months_to_date', 用于确定截至 2022 年 5 月 15 日的相同季度内发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    inmonthstodate(3,date,'05/15/2022', 0,3) as in_months_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_months_to_date

结果表

日期	previous_qtr_to_date
1/7/2022	0
1/19/2022	0

日期	previous_qtr_to_date
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过在 `inmonthstodate()` 函数中使用 `3` 作为 `first_month_of_year` 参数，函数从 3 月 1 日开始一年。然后根据提供的第一参数将一年分成四个季度。因此，四分之一部分为：

- 3 月-5 月
- 6 月-8 月
- 9 月-11 月
- 12 月 - 2 月

然后，为 5 月 15 日的 `base_date` 通过将其结束边界设置为 5 月 15 号，细分 3 月至 5 月季度。

inmonthstodate 函数的图表，3 月为一年中的第一个月。

因此，在 3 月 1 日和 5 月 15 日之间发生的任何交易将返回布尔结果 TRUE，而日期在这些边界之外的交易将返回值 FALSE。

示例 4 – 图表示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

在本例中，未更改的数据集被加载到应用程序中。任务是创建一个计算，确定交易是否发生在 5 月 15 日的同一季度，作为应用程序图表中的一个度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
```

```
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
date
```

要计算交易是否与 5 月 15 日发生在同一季度，请创建以下度量：

```
=inmonthstodate(3,date,'05/15/2022', 0)
```

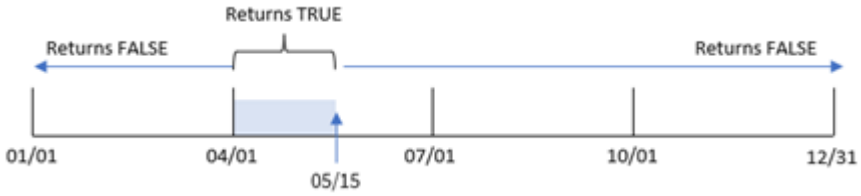
结果表

日期	=inmonthstodate(3,date,'05/15/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过使用 `inmonthstodate()` 函数在图表中创建 `‘in_months_to_date` 度量。

提供的第一个参数是 3，其将一年划分为季度分段。第二个参数标识要计算的字段。第三个参数是 5 月 15 日的硬编码日期，其为 base_date，定义了段的结束边界。0 的 period_no 为最终参数。

带季度段的 inmonthstodate 函数的图表。



4 月 1 日至 5 月 15 日之间发生的任何交易都会返回布尔值结果 TRUE。超出该段的交易日期返回 FALSE。

示例 5 – 场景

加载脚本和结果

概述

在本例中，将数据集加载到名为 'Sales' 的表中。表格包含以下字段：

- 产品 ID
- Product Type
- Sales date
- Sales price

最终用户希望有一个图表，按产品类型显示截至 2022 年 12 月 24 日的销售情况。用户希望能够定义此期间的长度。

加载脚本

```
SET vPeriod = 1;

Products:
Load
*
Inline
[
product_id,product_type,sales_date,sales_price
8188,product A,'9/19/2022',37.23
8189,product D,'10/27/2022',17.17
8190,product C,'10/30/2022',88.27
8191,product B,'10/31/2022',57.42
8192,product D,'11/16/2022',53.80
8193,product D,'11/28/2022',82.06
8194,product A,'12/2/2022',40.39
8195,product B,'12/5/2022',87.21
8196,product C,'12/15/2022',95.93
8197,product B,'12/16/2022',45.89
8198,product C,'12/19/2022',36.23
```

```
8199,product D,'12/22/2022',25.66
8200,product D,'12/23/2022',82.77
8201,product A,'12/24/2022',69.98
8202,product A,'12/24/2022',76.11
8203,product B,'12/26/2022',25.12
8204,product B,'12/27/2022',46.23
8205,product B,'12/27/2022',84.21
8206,product C,'12/28/2022',96.24
8207,product D,'12/29/2022',67.67
];
```

结果

加载数据并打开工作表。

在加载脚本开始时，创建了一个变量 `vPeriod`，该变量将绑定到变量输入控件。

进行以下操作：

1. 在资产面板中，单击**自定义对象**。
2. 选择 **Qlik 仪表板捆绑**并将**变量输入**添加到工作表中。
3. 输入图表的标题。
4. 在**变量**下，选择 **vPeriod** 作为名称，并将对象设置为显示为**下拉列表**。
5. 在**值**下，单击**动态**值。输入以下内容：
   `='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.`
6. 将新表添加到工作表中。
7. 在属性面板中的**数据**下，将 `product_type` 添加为维度。
8. 添加以下表达式作为度量：
   `=sum(if(inmonthstodate($(vPeriod),sales_date,makedate(2022,12,24),0),sales_price,0))`
9. 将度量的**数字格式**设置为**金额**。

结果表

product_type	=sum(if(inmonthstodate($(vPeriod),sales_date,makedate(2022,12,24),0),sales_price,0))
product A	$186.48
product B	$190.52
product C	$220.43
product D	$261.46

`inmonthstodate()` 函数使用用户输入作为参数来定义一年起始段的大小。

函数将每个产品的销售日期作为 `inmonthstodate()` 函数的第二个参数传递。通过将 12 月 24 日用作 `inmonthstodate()` 函数中的第三个参数，销售日期在定义的时间段（包括 12 月 24 号）之前的产品将返回布尔值 TRUE。sum 函数用于添加这些产品的销售额。

inmonthtodate

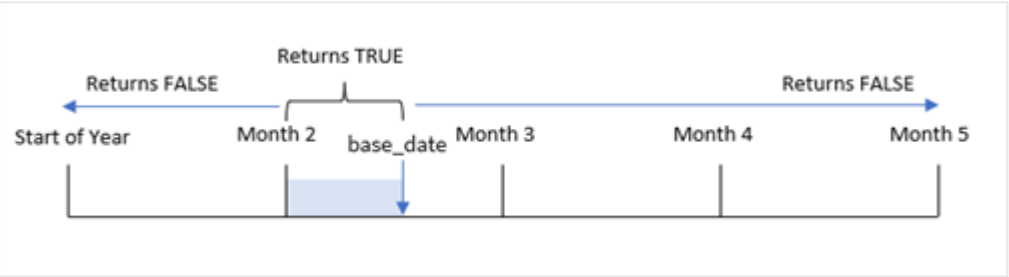
用于返回 True，如果 **date** 位于包含 **basedate** 为止以及包括 **basedate** 最后毫秒的月份部分以内。

语法：

```
InMonthToDate (timestamp, base_date, period_no)
```

返回数据类型：布尔值

inmonthtodate 函数的图表。



inmonthtodate() 函数将所选月份标识为段。开始边界是月初。结束边界可以设置为当月的较晚日期。然后，它确定一组日期是否属于此段，返回 TRUE 或 FALSE 布尔值。

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。

适用场景

inmonthtodate() 函数返回布尔值结果。通常，这种类型的函数用作 if expression 中的条件。inmonthtodate() 函数返回一个聚合或计算，该聚合或计算取决于某个日期是否发生在相关日期之前的月份中。

例如，inmonthtodate() 函数可用于识别截至特定日期的一个月份内制造的所有设备。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
inmonthtodate ('01/25/2013', '25/01/2013', 0)	返回 True
inmonthtodate ('01/25/2013', '24/01/2013', 0)	返回 False
inmonthtodate ('01/25/2013', '28/02/2013', -1)	返回 True

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供的日期字段。
- 前置 Load 语句包含：
    - 设置为 'in_month_to_date' 字段的 inmonthtodate() 函数。这决定了哪些交易发生在 2022 年 7 月 1 日至 26 日之间。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    inmonthtodate(date,'07/26/2022', 0) as in_month_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
```

```
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_month_to_date

结果表

日期	in_month_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	-1
7/22/2022	-1
7/23/2022	-1
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

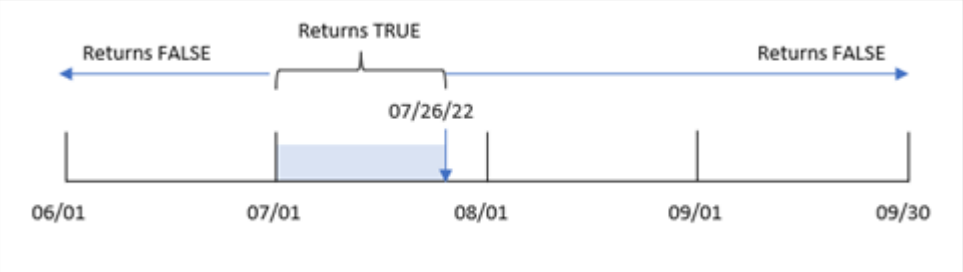
日期	in_month_to_date
9/26/2022	0
10/14/2022	0
10/29/2022	0

in_month_to_date 字段是在前置 Load 语句中使用 inmonthtodate() 函数创建的。

第一个参数标识正在评估的字段。第二个参数是硬编码日期，7 月 26 日，其为 base_date。该 base_date 参数标识分段的月份和该分段的结束边界。

为 0 的 period_no 是最后一个参数，这意味着该函数不比较分段月份之前或之后的月份。

inmonthtodate 函数图表，没有额外参数。



因此，7 月 1 日至 26 日之间发生的任何交易都会返回布尔结果 TRUE。7 月 26 日之后 7 月发生的任何交易都会返回布尔值 FALSE 结果，这与一年中其他月份的任何交易一样。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

在本例中，任务是创建一个字段 'six_months_prior'，用于确定在 7 月 1 日和 7 月 26 日之前整整六个月内发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    inmonthtodate(date,'07/26/2022', -6) as six_months_prior
    ;
Load
*
Inline
[
```

```
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `six_months_prior`

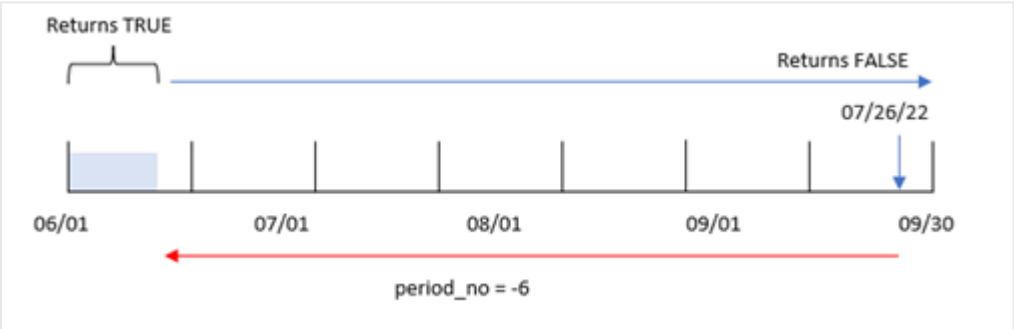
结果表

日期	six_months_prior
1/7/2022	-1
1/19/2022	-1
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0

日期	six_months_prior
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过在 inmonthtodate() 函数中使用 -6 作为 period_no 参数，比较器月份段的边界移动了六个月。最初，月份段相当于 7 月 1 日至 7 月 26 日。然后，period_no 将此部分抵消六个月的负值，日期边界将在 1 月 1 日至 1 月 26 日之间移动。

inmonthtodate 函数的图表，其中 period_no 设置为 -6.



因此，在 1 月 1 日到 1 月 26 日之间发生的任何事务都将返回布尔结果 TRUE。

示例 3 – 图表示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

在本例中，未更改的数据集被加载到应用程序中。任务是创建一个计算，确定交易是否发生在 7 月 1 日至 7 月 26 日之间，作为应用程序图表中的一个度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
```

```
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
date
```

要计算交易是否发生在 7 月 1 日至 26 日之间，请创建以下度量：

```
=inmonthtodate(date,'07/26/2022', 0)
```

结果表

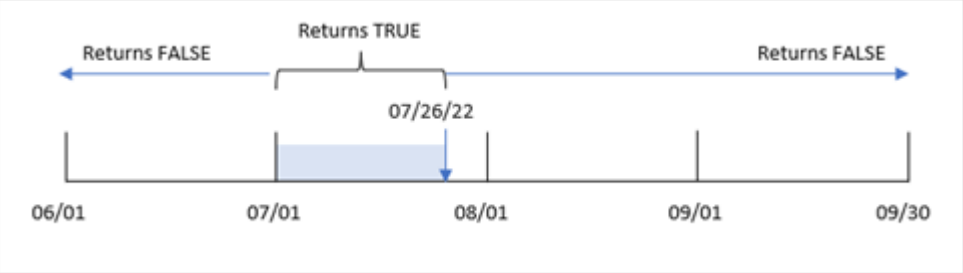
日期	=inmonthtodate(date,'07/26/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0

日期	=inmonthtodate(date,'07/26/2022', 0)
6/15/2022	0
6/26/2022	0
7/9/2022	-1
7/22/2022	-1
7/23/2022	-1
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

使用 inmonthtodate() 函数在图表中创建 'in_month_to_date' 字段度量。

第一个参数标识正在评估的字段。第二个参数是硬编码日期，7 月 26 日，其为 base_date。该 base_date 参数标识分段的月份和该分段的结束边界。为 0 的 period_no 为最终参数。这意味着该函数不会比较分段月份之前或之后的月份。

inmonthtodate 函数图表，没有额外参数。



因此，7 月 1 日至 26 日之间发生的任何交易都会返回布尔结果 TRUE。7 月 26 日之后 7 月发生的任何交易都会返回布尔值 FALSE 结果，这与一年中其他月份的任何交易一样。

示例 4 – 场景

加载脚本和结果

概述

在本例中，将数据集加载到名为 'Products' 的表中。表格包含以下字段：

- 产品 ID
- 制造日期
- 成本价

由于设备错误，2022 年 7 月制造的产品存在缺陷。该问题于 2022 年 7 月 27 日得到解决。

最终用户需要一个图表，按月显示制造为“有缺陷”(布尔值为 TRUE) 或“无缺陷”(布尔值为 FALSE) 的产品的状态，以及当月制造的产品的成本。

加载脚本

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `=monthname(manufacture_date)`
- `=if(Inmonthtodate(manufacture_date,makedate(2022,07,26),0),'Defective','Faultless')`

要计算产品的总成本，请创建此度量：

```
=sum(cost_price)
```

将度量的**数字格式**设置为**金额**。

结果表

monthname (manufacture_date)	if(Inmonthtodate(manufacture_date,makedate (2022,07,26),0),'Defective','Faultless')	Sum(cost_ price)
Jan 2022	Faultless	$54.40
Feb 2022	Faultless	$145.69
Mar 2022	Faultless	$53.80
Apr 2022	Faultless	$82.06
May 2022	Faultless	$127.60
Jun 2022	Faultless	$141.82
Jul 2022	Defective	$144.66
Jul 2022	Faultless	$69.98
Aug 2022	Faultless	$147.46
Sep 2022	Faultless	$84.21
Oct 2022	Faultless	$163.91

`inmonthtodate()` 函数在评估每个产品的制造日期时返回布尔值。

对于返回布尔值 TRUE 的日期，产品标记为“缺陷”。对于返回 FALSE 值的任何产品，因此在 7 月 26 日之前（包括 7 月 26 号）的一个月内未生产，它将产品标记为“无故障”。

inquarter

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的季度以内。

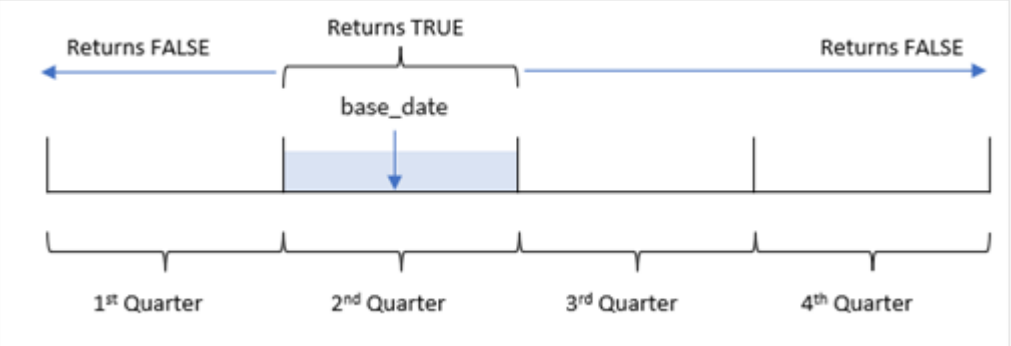
语法：

```
InQuarter (timestamp, base_date, period_no[, first_month_of_year])
```

返回数据类型： 布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

`inquarter()` 函数范围的图表



换句话说，inquarter() 函数在 1 月 1 日至 12 月 31 日期间将一年分为四个相等的季度。您可以使用 first_month_of_year 参数更改应用程序中的第一个月，季度将根据该参数进行更改。函数 base_date 确定应将哪个季度用作函数的比较器。最后，该函数在将日期值与该季度段进行比较时返回布尔结果。

适用场景

inquarter() 函数返回布尔值结果。通常，这种类型的函数将用作 if expression 中的条件。这将返回一个聚合或计算，该聚合或计算取决于日期是否发生在所选季度。

例如，inquarter() 函数可用于根据设备制造日期识别季度段内制造的所有设备。

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。

可以使用以下值在 first_month_of_year 参数中设置一年中的第一个月：

first_month_of_year 值

月	值
二月	2
三月	3
四月	4
五月	5
六月	6
七月	7
八月	8
九月	9
十月	10
十一月	11
十二月	12

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
inquarter ('01/25/2013', '01/01/2013', 0)	返回 TRUE
inquarter ('01/25/2013', '04/01/2013', 0)	返回 FALSE
inquarter ('01/25/2013', '01/01/2013', -1)	返回 FALSE
inquarter ('12/25/2012', '01/01/2013', -1)	返回 TRUE
inquarter ('01/25/2013', '03/01/2013', 0, 3)	返回 FALSE
inquarter ('03/25/2013', '03/01/2013', 0, 3)	返回 TRUE

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 前置 Load，其中包含设置为 'in_quarter' 字段的 inquarter() 函数，并确定哪些事务发生在 2022 年 5 月 15 日的同一季度。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       inquarter (date,'05/15/2022', 0) as in_quarter
       ;
Load
```

```
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `in_quarter`

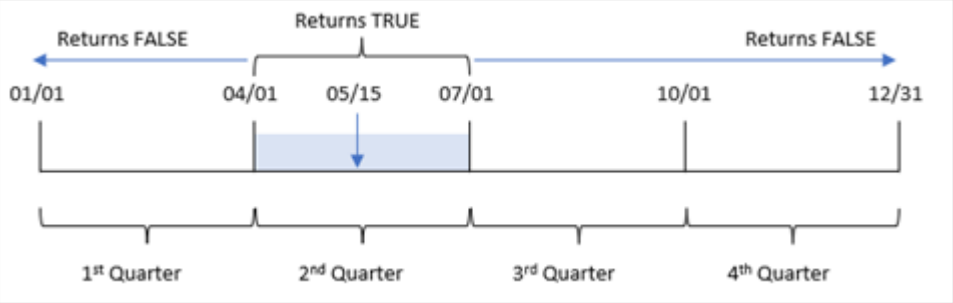
结果表

日期	in_quarter
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	-1
6/26/2022	-1

日期	in_quarter
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

in_quarter 字段是在前置 Load 语句中使用 inquarter() 函数创建的。第一个参数标识正在评估的字段。第二个参数是 5 月 15 日的硬编码日期，用于确定要将哪个季度定义为比较器。为 0 的 period_no 是最后一个参数，它确保 inquarter() 函数不会比较分段四分之一之前或之后的四分之一。

以 5 月 15 日为基准日的 inquarter() 函数的图表



4 月 1 日至 6 月 30 日结束之间发生的任何交易都将返回布尔值结果 TRUE。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 前置 Load，其中包含设置为 'previous_quarter' 字段的 inquarter() 函数，并确定哪些事务发生在 2022 年 5 月 15 日所在季度之前的一个季度。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       inquarter (date,'05/15/2022', -1) as previous_qtr
       ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_qtr

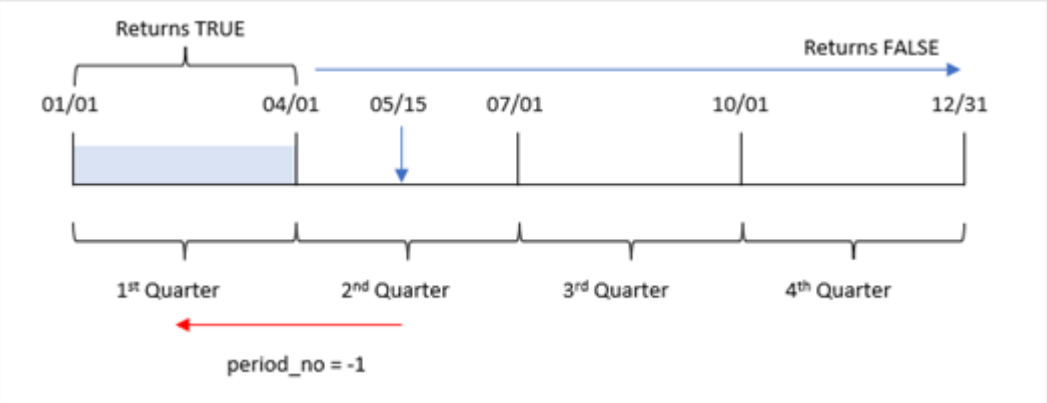
结果表

日期	previous_qtr
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	-1

日期	previous_qtr
3/16/2022	-1
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

将 -1 用作 inquarter() 函数中的 period_no 参数将比较器的边界向后移动四分之一。5 月 15 日属于一年的第二季度，因此该部分最初相当于 4 月 1 日至 6 月 30 日的季度。period_no 将此段抵消负三个月，并导致日期边界变为 1 月 1 日至 3 月 30 日。

以 5 月 15 日为基准日的 inquarter() 函数的图表



因此，1 月 1 日和 3 月 30 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 前置 Load，其中包含设置为 'in_quarter' 字段的 inquarter() 函数，并确定哪些事务发生在 2022 年 5 月 15 日的同一季度。

但是在本例中，组织策略是将三月作为财政年度的第一个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       inquarter (date,'05/15/2022', 0, 3) as in_quarter
       ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

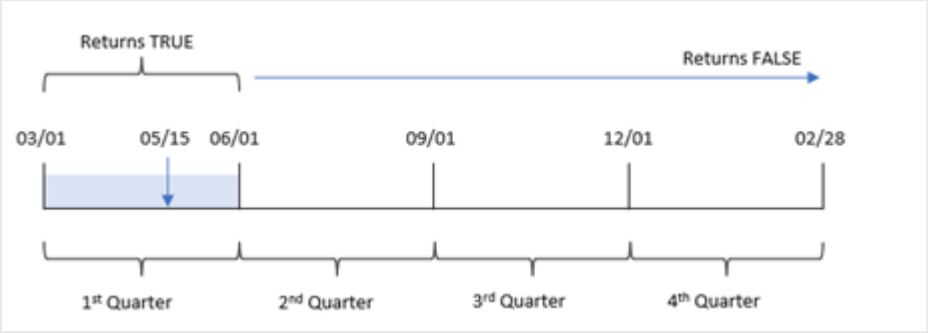
- date
- previous_qtr

结果表

日期	previous_qtr
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

在 inquarter() 函数中使用 3 作为 first_month_of_year 参数将 3 月 1 日设置为一年的开始，然后将一年划分为几个季度。因此，季度细分为 3 - 5 月、6 - 8 月、9 - 11 月、12-2 月。5 月 15 日的 base_date 将 3-5 月季度设置为函数的比较季度。

inquarter() 函数的图表，3 月为一年中的第一个月



因此，3 月 1 日至 5 月 31 日之间发生的任何交易都将返回布尔结果 TRUE。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 前置 Load，其中包含设置为 'in_quarter' 字段的 inquarter() 函数，并确定哪些事务发生在 2022 年 5 月 15 日的同一季度。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
```

```
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- date

创建以下度量以计算交易是否与 5 月 15 日发生在同一季度：

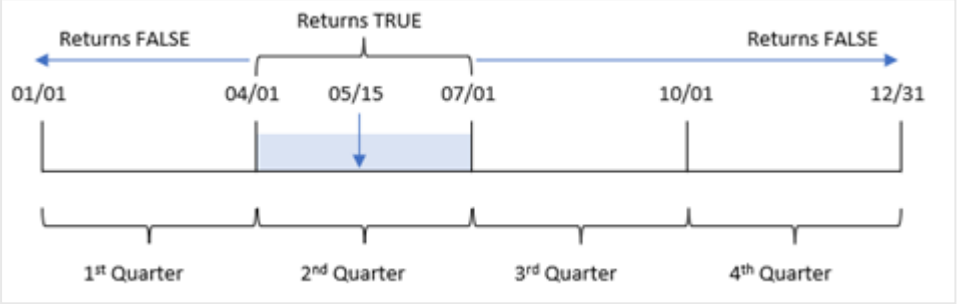
```
=inquarter(date,'05/15/2022', 0)
```

结果表

日期	in_quarter
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过使用 inquarter() 函数在图表中创建 'in_quarter' 度量。第一个参数标识正在评估的字段。第二个参数是 5 月 15 日的硬编码日期，用于确定要将哪个季度定义为比较器。为 0 的 period_no 是最后一个参数，它确保 inquarter() 函数不会比较分段四分之一之前或之后的四分之一。

以 5 月 15 日为基准日的 inquarter() 函数的图表



4 月 1 日至 6 月 30 日结束之间发生的任何交易都将返回布尔值结果 TRUE。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 'Products' 的表中的数据集。
- 表格包含以下字段：
  - 产品 ID
  - 产品类型
  - 制造日期
  - 成本价

现已确定，由于设备错误，2022 年 5 月 15 日所在季度生产的产品存在缺陷。最终用户想要一个图表，按季度名称显示生产的产品“有缺陷”或“无缺陷”的状态以及该季度生产的产品的成本。

加载脚本

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
```

```
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
=quartername(manufacture_date)
```

创建以下度量：

- `=if(only(InQuarter(manufacture_date,makedate(2022,05,15),0)),'Defective','Faultless')`，用于使用 `inquarter()` 函数识别哪些产品有缺陷，哪些无缺陷：
- `=sum(cost_price)`，用于显示每个产品的成本总和。

执行以下操作：

1. 将度量的**数字格式**设置为**金额**。
2. 在**外观**下，关闭**总计**。

结果表

quartername (manufacture_date)	=if(only(InQuarter(manufacture_date,makedate (2022,05,15),0)),'Defective','Faultless')	Sum (cost_ price)
Jan-Mar 2022	Faultless	253.89
Apr-Jun 2022	Defective	351.48
Jul-Sep 2022	Faultless	446.31
Oct-Dec 2022	Faultless	163.91

`inquarter()` 函数在评估每个产品的制造日期时返回布尔值。对于包含 5 月 15 日所在季度内生产的任何产品，`inquarter()` 函数返回布尔值 TRUE，并将产品标记为“缺陷”。对于返回 FALSE 值，因此未在该季度生产的任何产品，它将产品标记为“无故障”。

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。

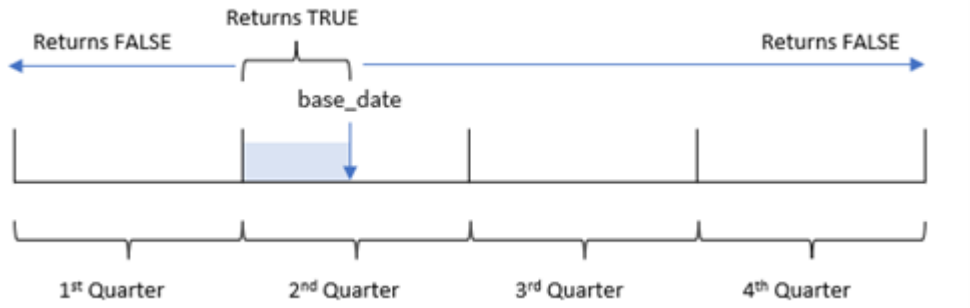
语法：

```
InQuarterToDate (timestamp, base_date, period_no [, first_month_of_year])
```

返回数据类型：布尔值

> *在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。*

InquarteTodate 函数图表



inquartertodate() 函数将一年分为四个相等的季度，即 1 月 1 日至 12 月 31 日（或用户定义的年初及其相应的结束日期）。使用 base_date，该函数将对特定的季度进行分段，同时 base_date 识别该季度分段的哪个季度和允许的最大日期。最后，当将规定的日期值与该段进行比较时，该函数返回布尔值结果。

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。

适用场景

inquartertodate() 函数返回布尔结果。通常，这种类型的函数将用作 if 表达式中的条件。inquartertodate() 函数将用于返回一个聚合或计算，这取决于评估日期是否发生在该日期之前的季度。

例如，inquartertodate() 函数可用于识别截至特定日期的一个季度内制造的所有设备。

函数示例

示例	结果
inquartertodate ('01/25/2013', '03/25/2013', 0)	返回 TRUE, 由于 timestamp 的值, 01/25/2013, 其在 01/01/2013 至 03/25/2013 的三个月内，其中有 base_date 的值，03/25/2013。
inquartertodate ('04/26/2013', '03/25/2013', 0)	返回 FALSE, 因为 04/26/2013 在与前一示例相同的时间段之外。
inquartertodate ('02/25/2013', '06/09/2013', -1)	返回 TRUE, 因为 period_no 的值 -1 将搜索期间切换回三个月长度(一年的一个季度)的期间。这让搜索期间为 01/01/2013 至 03/09/2013.
inquartertodate ('03/25/2006', '04/15/2006', 0, 2)	返回 TRUE, 因为 first_month_of_year 的值设置为 2, 这让搜索期间为 02/01/2006 至 04/15/2006 而非 04/01/2006 至 04/15/2006。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 in_quarter_to_date，用于确定截至 2022 年 5 月 15 日的季度内发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
```

```
    Load
        *,
        inquartertodate(date,'05/15/2022', 0) as in_quarter_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_quarter_to_date

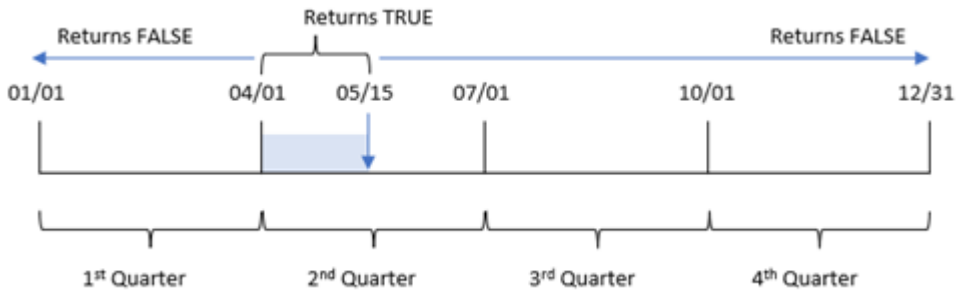
结果表

日期	in_quarter_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1

日期	in_quarter_to_date
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

in_quarter_to_date 字段是在前置 Load 语句中使用 inquartertodate() 函数创建的。提供的第一个参数标识正在评估的字段。第二个参数是 5 月 15 日的硬编码日期，它是标识要分段的季度的 base_date 并定义该分段的结束边界。为 0 的 period_no 是最后一个参数，这意味着该函数不比较分段季度之前或之后的季度。

InquarteTodate 函数图表，没有额外参数



4 月 1 日至 5 月 15 日之间发生的任何交易都将返回布尔结果 TRUE。5 月 16 日及之后的交易日期将返回 FALSE，4 月 1 日之前的任何交易也将返回。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_qtr_to_date，确定在 2022 年 5 月 15 日结束的季度段之前的整个季度发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inquartertodate(date,'05/15/2022', -1) as previous_qtr_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

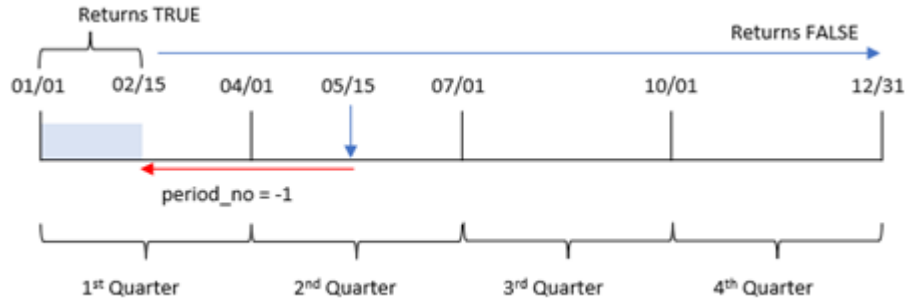
- date
- previous_qtr_to_date

结果表

日期	previous_qtr_to_date
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

值为 -1 的 period_no 指示 inquartertodate () 函数将输入季度段与前一季度进行比较。5 月 15 日属于一年的第二季度，因此该部分最初等于 4 月 1 日至 5 月 15 之间。然后 period_no 将该段提前三个月偏移，导致日期边界变为 1 月 1 日至 2 月 15 日。

InquarteTodate 函数 period_no 示例的图表



因此，1 月 1 日至 2 月 15 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 in_quarter_to_date，用于确定截至 2022 年 5 月 15 日的相同季度内发生了哪些交易。

在本例中，我们将 3 月设置为会计年度的第一个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inquartertodate(date,'05/15/2022', 0,3) as in_quarter_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
```

```
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `in_quarter_to_date`

结果表

日期	in_quarter_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

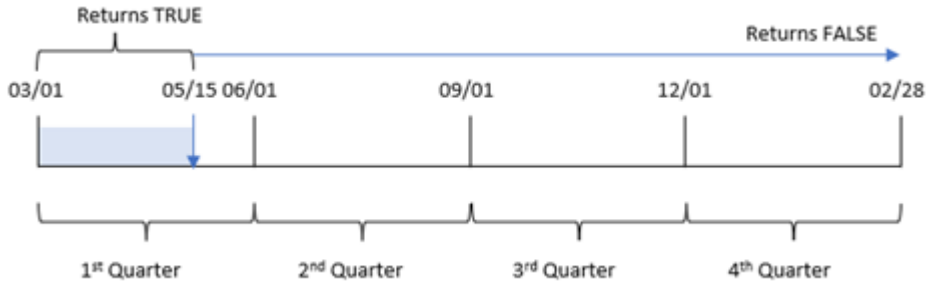
日期	in_quarter_to_date
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过在 inquartertodate() 函数中使用 3 作为 first_month_of_year 参数，函数从 3 月 1 日开始一年，然后将一年分成四个季度。因此，四分之一部分为：

- 三月至五月
- 六月至八月
- 九月至十一月
- 十二月至二月

然后，为 5 月 15 日的 base_date 通过将其结束边界设置为 5 月 15 号，细分 3 月至 5 月季度。

InquarteTodate 函数 first_month_of_year 示例的图表



因此，在 3 月 1 日和 5 月 15 日之间发生的任何交易将返回布尔结果 TRUE，而日期在这些边界之外的交易将返回值 FALSE。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。确定与 5 月 15 日在同一季度发生的交易的计算将作为图表对象中的度量创建。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
```

```
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量：

```
=inquartertodate(date,'05/15/2022', 0)
```

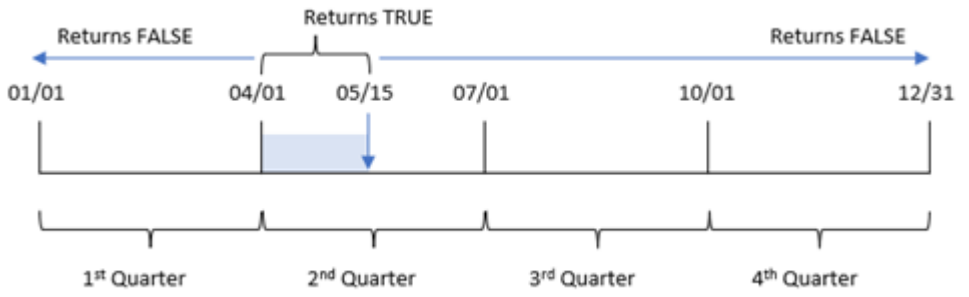
结果表

日期	=inquartertodate(date,'05/15/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0

日期	=inquartertodate(date,'05/15/2022', 0)
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

通过使用 inquartertodate() 函数在图表对象中创建 in_quarter_to_date 度量。第一个参数是要计算的日期字段。第二个参数是 5 月 15 日的硬编码日期，它是标识要分段的季度的 base_date 并定义该分段的结束边界。为 0 的 period_no 是最后一个参数，这意味着该函数不比较分段季度之前或之后的季度。

InquarteTodate 函数的图表，图表对象示例



4 月 1 日至 5 月 15 日之间发生的任何交易都将返回布尔结果 TRUE。5 月 16 日及之后的交易日期将返回 FALSE，4 月 1 日之前的任何交易也将返回。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Products 的表中的数据集。
- 有关产品 ID、制造日期和成本价格的信息。

2022 年 5 月 15 日，在制造过程中发现并解决了一个设备错误。在该季度生产的产品将有缺陷。最终用户需要一个图表对象，该对象按季度名称显示产品是否“有缺陷”或“无缺陷”的状态，以及该季度迄今为止生产的产品的成本。

加载脚本

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。创建维度以显示季度名称：
   `=quartername(manufacture_date)`
2. 接下来，创建一个维度，以确定哪些产品有缺陷，哪些无缺陷：
   `=if(inquartertodate(manufacture_date,makedate(2022,05,15),0),'Defective','Faultless')`
3. 创建一个度量来求对产品的 cost_price 求和：
   `=sum(cost_price)`
4. 将度量的**数字格式**设置为**金额**。

结果表

quartername (manufacture_date)	if(inquartertodate(manufacture_date,makedate (2022,05,15),0),'Defective','Faultless')	Sum(cost_ price)
Jan-Mar 2022	Faultless	$253.89
Apr-Jun 2022	Faultless	$229.03
Apr-Jun 2022	Defective	$122.45
Jul-Sep 2022	Faultless	$446.31
Oct-Dec 2022	Faultless	$163.91

inquartertodate() 函数在评估每个产品的制造日期时返回布尔值。对于返回布尔值 TRUE 的产品，它将产品标记为 'Defective'。对于返回值为 FALSE 的任何产品，因此在截至 5 月 15 日(含)的季度内未生产，其将产品标记为 'Faultless'。

inweek

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的星期以内。

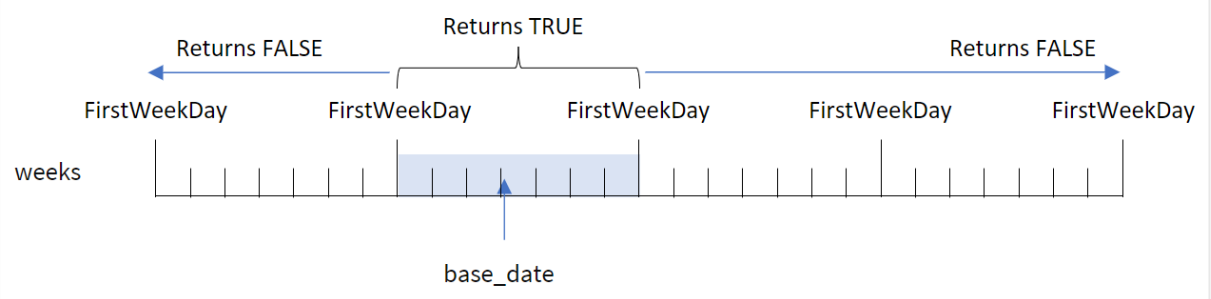
语法：

```
InWeek (timestamp, base_date, period_no[, first_week_day])
```

返回数据类型：布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

inweek() 函数范围的图表



inweek() 函数使用base_date 参数来确定日期所属的七天期间。一周的开始日期基于 FirstWeekDay 系统变量。但是，您可以使用 inweek() 函数中的 first_week_day 参数更改将哪个月设置为第一个月。

定义选定的周后，将指定的日期值与该周段进行比较时，函数将返回布尔值结果。

适用场景

inweek() 函数返回布尔值结果。通常，这种类型的函数将用作 if expression 中的条件。inweek() 函数返回一个聚合或计算，该聚合或计算取决于所计算的日期是否发生在 base_date 参数的选定日期所在的周内。

例如，inweek() 函数可用于识别特定周内制造的所有设备。

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，一周的第一天是星期日（由 FirstWeekDay 系统变量确定），从星期六和星期日之间的午夜开始。**first_week_day** 参数取代 **FirstWeekDay** 变量。要指示一周从另外一天开始，指定一个介于 0 和 6 之间的标志。

first_week_day 值

日	值
星期一	0
星期二	1
星期三	2
星期四	3
星期五	4
星期六	5
星期日	6

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
`inweek ('01/12/2006', '01/14/2006', 0)`	返回 TRUE

示例	结果
inweek ('01/12/2006', '01/20/2006', 0)	返回 FALSE
inweek ('01/12/2006', '01/14/2006', -1)	返回 FALSE
inweek ('01/07/2006', '01/14/2006', -1)	返回 TRUE
inweek ('01/12/2006', '01/09/2006', 0, 3)	返回 FALSE是因为 first_week_day 指定为 3(星期四), 这使得 01/12/2006 称为包含 01/09/2006 的一周的下一周的第一天.

这些主题可以帮助您使用此函数:

相关主题

主题	默认标志/值	说明
FirstWeekDay (page 218)	6 / 周日	定义每周的开始日期。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 包含 2022 年 1 月交易集的数据集, 该数据集加载到名为 'Transactions' 的表中。
- 设置为 6(星期日) 的 FirstWeekDay 系统变量。
- 包含以下内容的前置 Load:
  - inweek() 函数设置为字段 'in_week', 用于确定 2022 年 1 月 14 日这一周发生了哪些交易。
  - weekday() 函数设置为字段 'week_day', 显示一周中的哪一天对应于每个日期。

加载脚本

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       weekday(date) as week_day,
       inweek(date,'01/14/2022', 0) as in_week
       ;
Load
*
Inline
```

```
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- week_day
- in_week

结果表

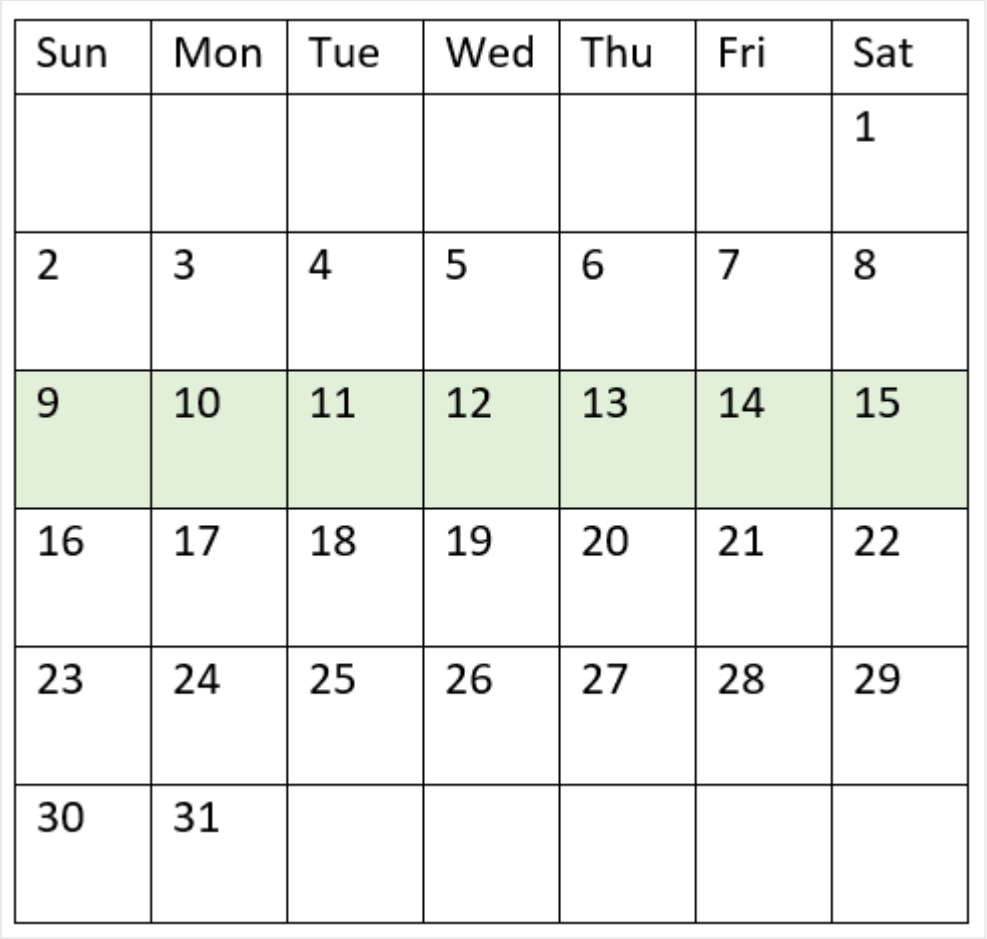
日期	week_day	in_week
01/02/2022	Sun	0
01/05/2022	Wed	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	Sun	-1
01/10/2022	Mon	-1
01/11/2022	Tue	-1
01/12/2022	Wed	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1

日期	week_day	in_week
01/15/2022	Sat	-1
01/16/2022	Sun	0
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	Sun	0
01/31/2022	Mon	0

in_week 字段是在前置 Load 语句中使用 inweek() 函数创建的。第一个参数标识正在评估的字段。第二个参数是硬编码日期，1 月 14 日，其为 base_date。base_date 参数与 FirstWeekDay 系统变量配合使用，以标识比较周。为 0 的 period_no - 这意味着该函数不是分段周之前或之后的比较周，而是最后一个参数。

FirstWeekDay 系统变量确定星期从星期日开始，到星期六结束。因此，根据下图，1 月将分为几周，1 月 9 日至 15 日之间的日期为 inweek() 计算的有效期：

突出显示 inweek() 函数范围的日历图



Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1月 9 日至 1月 15 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的相同数据集，该数据集加载到名为 Transactions 的表中。
- 设置为 6(星期日) 的 FirstWeekDay 系统变量。
- 包含以下内容的前置 Load：
  - inweek () 函数设置为字段 'prev_week'，用于确定 2022 年 1月 14 日这一周之前的整周发生了哪些交易。
  - weekday() 函数设置为字段 'week_day'，显示一周中的哪一天对应于每个日期。

加载脚本

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       weekday(date) as week_day,
       inweek(date,'01/14/2022', -1) as prev_week
       ;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- week_day
- prev_week

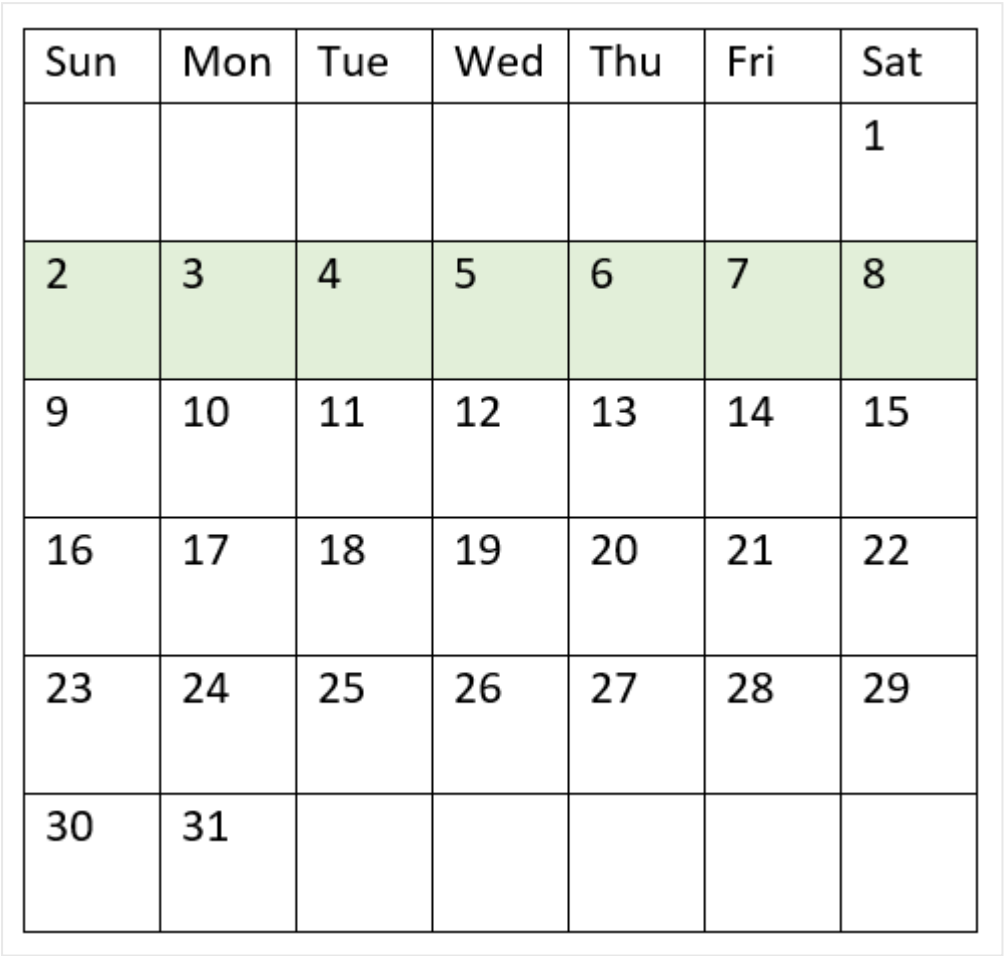
结果表

日期	week_day	prev_week
01/02/2022	Sun	-1
01/05/2022	Wed	-1

日期	week_day	prev_week
01/06/2022	Thu	-1
01/08/2022	Sat	-1
01/09/2022	Sun	0
01/10/2022	Mon	0
01/11/2022	Tue	0
01/12/2022	Wed	0
01/13/2022	Thu	0
01/14/2022	Fri	0
01/15/2022	Sat	0
01/16/2022	Sun	0
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	Sun	0
01/31/2022	Mon	0

将 `-1` 用作 `inweek()` 函数中的 `period_no` 参数将比较周的边界向后移动全部七天。在具有为 `0` 的 `period_no` 的情况下，一周的时间将在1月9日至15日之间。但在本例中，为 `-1` 的 `period_no` 将此段的开始和结束边界向后移动一周。日期边界为 1 月 2 日至 1 月 8 日。

突出显示 inweek() 函数范围的日历图



Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

因此，1月 2日至 1月 8日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的相同数据集，该数据集加载到名为 Transactions 的表中。
- 设置为 6(星期日)的 FirstWeekDay 系统变量。
- 包含以下内容的前置 Load：
  - inweek() 函数设置为字段 'in_week'，用于确定 2022 年 1月 14 日这一周发生了哪些交易。
  - weekday() 函数设置为字段 'week_day'，显示一周中的哪一天对应于每个日期。

加载脚本

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       weekday(date) as week_day,
       inweek(date,'01/14/2022', 0, 0) as in_week
       ;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- week_day
- in_week

结果表

日期	week_day	in_week
01/02/2022	Sun	0
01/05/2022	Wed	0

日期	week_day	in_week
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	Sun	0
01/10/2022	Mon	-1
01/11/2022	Tue	-1
01/12/2022	Wed	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	Sun	-1
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	Sun	0
01/31/2022	Mon	0

通过在 `inweek()` 函数中使用 `0` 作为 `first_week_day` 参数，该参数将取代 `FirstWeekDay` 系统变量，并将星期一设置为一周的第一天。

突出显示 inweek() 函数范围的日历图

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

因此，1月 10 日和 16 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。在结果表中创建一个度量，以确定 2022 年 1月 14 日这一周发生了哪些交易。

加载脚本

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
```

```
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- `date`

创建以下度量：

- `=inweek (date,'01/14/2022',0)`，用于计算交易是否发生在 1 月 14 日的同一周。
- `=weekday(date)`，用于显示每个日期对应一周中的哪一天。

结果表

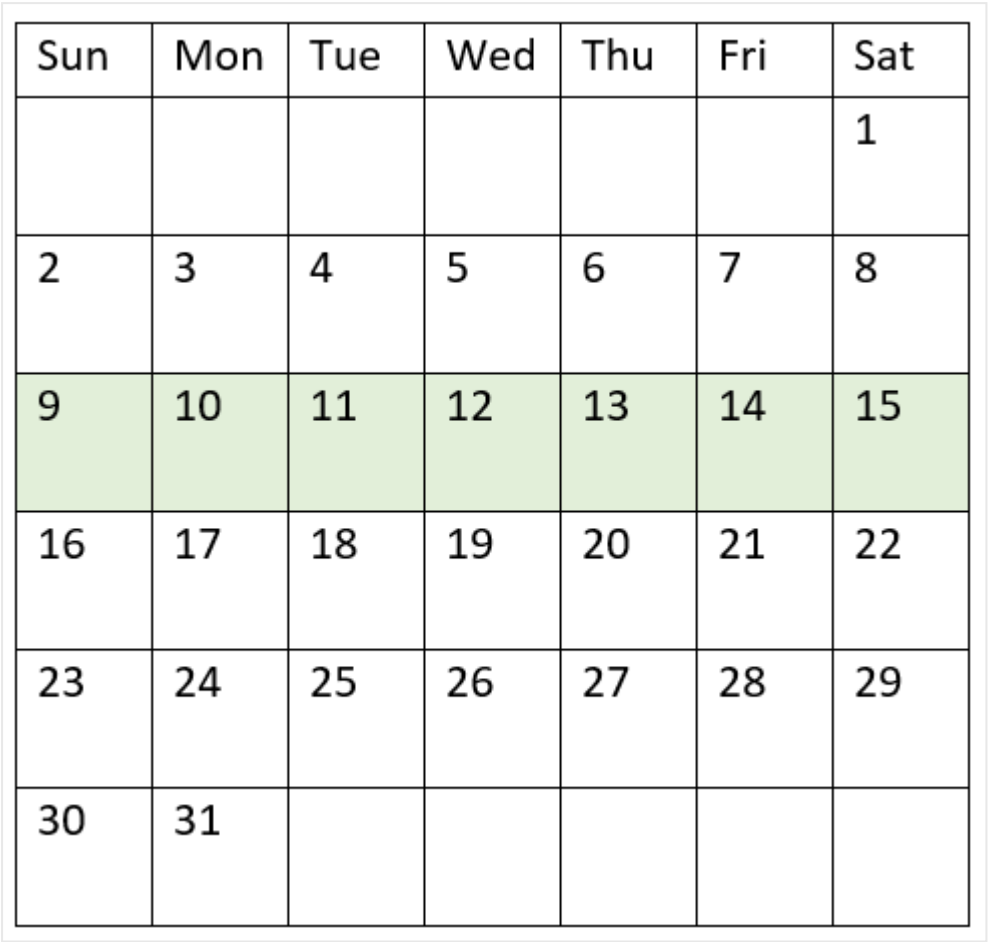
日期	week_day	=inweek (date,'01/14/2022',0)
01/02/2022	Sun	0
01/05/2022	Wed	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	Sun	-1
01/10/2022	Mon	-1
01/11/2022	Tue	-1
01/12/2022	Wed	-1

日期	week_day	=inweek (date,'01/14/2022',0)
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	Sun	0
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	Sun	0
01/31/2022	Mon	0

通过使用 inweek() 函数在图表中创建 'in_week' 度量。第一个参数标识正在评估的字段。第二个参数是硬编码日期，1 月 14 日，其为 base_date。base_date 参数与 FirstWeekDay 系统变量配合使用，以标识比较周。0 的 period_no 为最终参数。

FirstWeekDay 系统变量确定星期从星期日开始，到星期六结束。因此，根据下图，1 月将分为几周，1 月 9 日至 15 日之间的日期为 inweek() 计算的有效期：

突出显示 inweek() 函数范围的日历图



Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1月 9日至 1月 15日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 'Products' 的表中的数据集。
- 表格包含以下字段：
  - 产品 ID
  - 产品类型
  - 制造日期
  - 成本价

已确定，由于设备错误，1月 12 日一周内生产的产品存在缺陷。最终用户想要一个图表，它按周显示生产的产品是“有缺陷”还是“无缺陷”的状态，以及该周生产的产品的成本。

加载脚本

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- `=weekname(manufacture_date)`

创建以下度量：

- `=if(only(inweek(manufacture_date,makedate(2022,01,12),0)),'Defective','Faultless')`，用于使用 `inweek()` 函数识别哪些产品有缺陷，哪些无缺陷：
- `=sum(cost_price)`，用于显示每个产品的成本总和。

执行以下操作：

1. 将度量的**数字格式**设置为**金额**。
2. 在**外观**下，关闭**总计**。

结果表

weekname (manufacture_date)	=if(only(inweek(manufacture_date,makedate (2022,01,12),0)), 'Defective','Faultless')	Sum(cost_ price)
2022/02	Faultless	200.09
2022/03	Defective	441.51
2022/04	Faultless	178.41
2022/05	Faultless	231.67
2022/06	Faultless	163.91

inweek() 函数在评估每个产品的制造日期时返回布尔值。对于 1 月 12 日所在周生产的任何产品，inweek() 函数返回布尔值 TRUE，并将产品标记为 'Defective'。对于任何返回 FALSE 值的产品，由于不是在该周制造的，它将产品标记为 'Faultless'。

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。

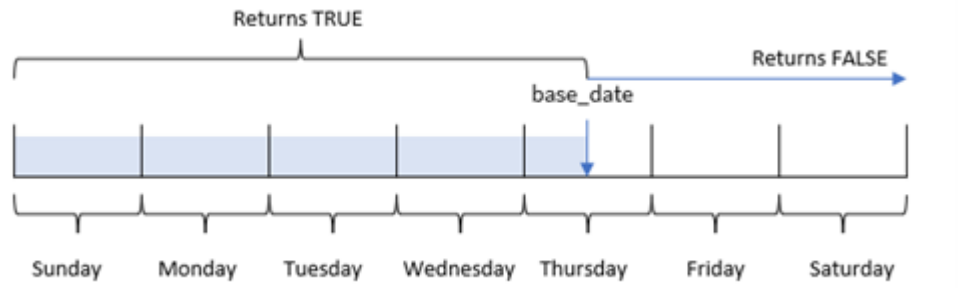
语法：

```
InWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

返回数据类型： 布尔值

> *在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。*

inweektodate 函数图表



inweektodate() 函数使用 base_date 参数来识别一周段的最大边界日期，以及基于 FirstWeekDay 系统变量(或用户定义的 first_week_day 参数)的一周开始的相应日期。定义本周段后，当将规定的日期值与该段进行比较时，函数将返回布尔结果。

适用场景

inweektodate() 函数返回布尔结果。通常，这种类型的函数将用作 if 表达式中的条件。这将返回一个聚合或计算，这取决于评估的日期是否发生在特定日期之前的一周内。

例如，inweektodate() 函数可用于计算截至特定日期的指定周内的所有销售额。

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，一周的第一天是星期日（由 FirstWeekDay 系统变量确定），从星期六和星期日之间的午夜开始。**first_week_day** 参数取代 **FirstWeekDay** 变量。要指示一周从另外一天开始，指定一个介于 0 和 6 之间的标志。 对于从周一开始到周日结束的一周，周一使用 0，周二使用 1，周三使用 2，周四使用 3，周五使用 4，周六使用 5，周日使用 6。

函数示例

示例	交互
inweektodate('01/12/2006', '01/12/2006', 0)	返回 TRUE。
inweektodate('01/12/2006', '01/11/2006', 0)	返回 FALSE。
inweektodate('01/12/2006', '01/18/2006', -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 01/11/2006。
inweektodate('01/11/2006', '01/12/2006', 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 01/12/2006 之后一周包含 01/12/2006。

这些主题可以帮助您使用此函数：

相关主题

主题	默认标志/值	说明
FirstWeekDay (page 218)	6 / 周日	定义每周的开始日期。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年 1 月交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 以格式 TimestampFormat='M/D/YYYY h:mm:ss[.fff]' 提供数据字段。
- 创建一个字段 in_week_to_date，用于确定截至 2022 年 1 月 14 日的周内发生了哪些交易。
- 使用 weekday() 函数创建另一个名为 weekday 的字段。创建此新字段是为了显示每个日期对应一周中的哪一天。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';
SET FirstWeekDay=6;
Transactions:
    Load
        *,
        weekday(date) as week_day,
        inweektodate(date,'01/14/2022', 0) as in_week_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
```

```
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- week_day
- in_week_to_date

结果表

日期	week_day	in_week_to_date
2022-01-02 12:22:06	Sun	0
2022-01-05 01:02:30	Wed	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	-1
2022-01-10 21:13:01	Mon	-1
2022-01-11 00:57:13	Tue	-1
2022-01-12 09:26:02	Wed	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

in_week_to_date 字段是在前置 Load 语句中使用 inweektodate() 函数创建的。提供的第一个参数标识正在评估的字段。第二个参数是 1 月 14 日的硬编码日期，它是标识要分段的周的 base_date 并定义该分段的结束边界。为 0 的 period_no 是最后一个参数，这意味着该函数不比较分段周之前或之后的周。

FirstWeekDay 系统变量确定星期从星期日开始，到星期六结束。因此，根据下图，1 月将分为几周，1 月 9 日至 14 日之间的日期为 inweekdodate() 计算的有效期：

显示事务日期的日历图，返回布尔值为 TRUE

Sun	Mon	Tue	Wed	Thur	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1 月 9 日至 14 日之间发生的任何交易都将返回布尔值结果 TRUE。日期之前和之后的交易返回的布尔结果为 FALSE。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 prev_week_to_date，确定在 2022 年 1 月 14 日结束的周段之前的整个周发生了哪些交易。
- 使用 weekday() 函数创建另一个名为 weekday 的字段。创建此项是为了显示每个日期对应一周中的哪一天。

加载脚本

```
SET FirstWeekDay=6;
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';
Transactions:
    Load
        *,
```

```
        weekday(date) as week_day,
        inweektodate(date,'01/14/2022', -1) as prev_week_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `week_day`
- `prev_week_to_date`

结果表

日期	week_day	prev_week_to_date
2022-01-02 12:22:06	Sun	-1
2022-01-05 01:02:30	Wed	-1
2022-01-06 15:36:20	Thu	-1
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	0
2022-01-10 21:13:01	Mon	0

日期	week_day	prev_week_to_date
2022-01-11 00:57:13	Tue	0
2022-01-12 09:26:02	Wed	0
2022-01-13 15:05:09	Thu	0
2022-01-14 18:44:57	Fri	0
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

值为 -1 的 period_no 指示 inweektodate () 函数将输入季度段与前一周进行比较。周段最初等于 1 月 9 日至 1 月 14 日之间。然后 period_no 将该段的开始边界和结束边界偏移到一周前，导致日期边界变为 1 月 2 日至 1 月 7 日。

显示事务日期的日历图，返回布尔值为 TRUE

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

因此，1 月 2 日至 8 日之间（包括 1 月 8 本身日）发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 in_week_to_date，用于确定截至 2022 年 1 月 14 日的周内发生了哪些交易。
- 使用 weekday() 函数创建另一个名为 weekday 的字段。创建此项是为了显示每个日期对应一周中的哪一天。

在此示例中，我们使用 Monday 作为一周的第一天。

加载脚本

```
SET FirstWeekDay=6;
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';

Transactions:
    Load
        *,
        weekday(date) as week_day,
        inweektodate(date,'01/14/2022', 0, 0) as in_week_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
```

```
8207,'2022-01-31 05:28:21',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- week_day
- in_week_to_date

结果表

日期	week_day	in_week_to_date
2022-01-02 12:22:06	Sun	0
2022-01-05 01:02:30	Wed	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	0
2022-01-10 21:13:01	Mon	-1
2022-01-11 00:57:13	Tue	-1
2022-01-12 09:26:02	Wed	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

通过在 inweektodate() 函数中使用 0 作为 first_week_day 参数，函数参数将取代 FirstWeekDay 系统变量，并将星期一设置为一周的第一天。

显示事务日期的日历图，返回布尔值为 TRUE

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	17
24	25	26	27	28	29	30
31						

因此，在 1 月 10 日和 14 日之间发生的任何交易将返回布尔结果 TRUE，而日期在这些边界之外的交易将返回值 FALSE。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。确定截至 2022 年 1 月 14 日的一周内发生了哪些交易的计算将在图表对象中创建为度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
```

```
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将该字段添加为维度：`date`。
2. 要计算截至 1 月 14 日的交易是否发生在同一周，请创建以下度量：
   `=inweektodate(date,'01/14/2022',0)`
3. 要显示一周中的哪一天对应于每个日期，请创建一个附加度量：
   `=weekday(date)`

结果表

日期	week_day	in_week_to_date
2022-01-02 12:22:06	Sun	0
2022-01-05 01:02:30	Wed	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	-1
2022-01-10 21:13:01	Mon	-1
2022-01-11 00:57:13	Tue	-1
2022-01-12 09:26:02	Wed	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0

日期	week_day	in_week_to_date
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

通过使用 inweektodate() 函数在图表对象中创建 in_week_to_date 字段作为度量。提供的第一个参数标识正在评估的字段。第二个参数是 1 月 14 日的硬编码日期，它是标识要分段的周的 base_date 并定义该分段的结束边界。为 0 的 period_no 是最后一个参数，这意味着该函数不比较分段周之前或之后的周。

FirstWeekDay 系统变量确定星期从星期日开始，到星期六结束。因此，根据下图，1 月将分为几周，1 月 9 日至 14 日之间的日期为 inweekdodate() 计算的有效期：

显示事务日期的日历图，返回布尔值为 TRUE

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1 月 9 日至 14 日之间发生的任何交易都将返回布尔值结果 TRUE。日期之前和之后的交易返回的布尔结果为 FALSE。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Products 的表中的数据集。
- 有关产品 ID、制造日期和成本价格的信息。

已确定，由于设备错误，1 月 12 日一周内生产的产品存在缺陷。该问题于 1 月 13 日得到解决。最终用户想要一个图表对象，它按周显示生产的产品是“有缺陷”还是“无缺陷”的状态，以及该周生产的产品的成本。

加载脚本

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。创建维度以显示周名称：
   `=weekname(manufacture_date)`
2. 接下来，创建一个维度，以确定哪些产品有缺陷，哪些无缺陷：
   `=if(inWeektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')`
3. 创建一个度量来求对产品的 cost_price 求和：
   `=sum(cost_price)`
4. 将度量的**数字格式**设置为**金额**。

结果表

weekname (manufacture_date)	if(inweektodate(manufacture_date,makedate (2022,01,12),0),'Defective','Faultless')	Sum(cost_ price)
2022/02	Faultless	$200.09
2022/03	Defective	$263.46
2022/03	Faultless	$178.05
2022/04	Faultless	$178.41
2022/05	Faultless	$147.46
2022/06	Faultless	$248.12

inweektodate() 函数在评估每个产品的制造日期时返回布尔值。对于返回布尔值 TRUE 的产品，它将产品标记为 'Defective'。对于返回值为 FALSE 的任何产品(因此在截至 1 月 12 日的一周内未生产)，它将产品标记为 'Faultless'。

inyear

此函数用于返回 True，如果 **timestamp** 位于包含 **base_date** 的年份以内。

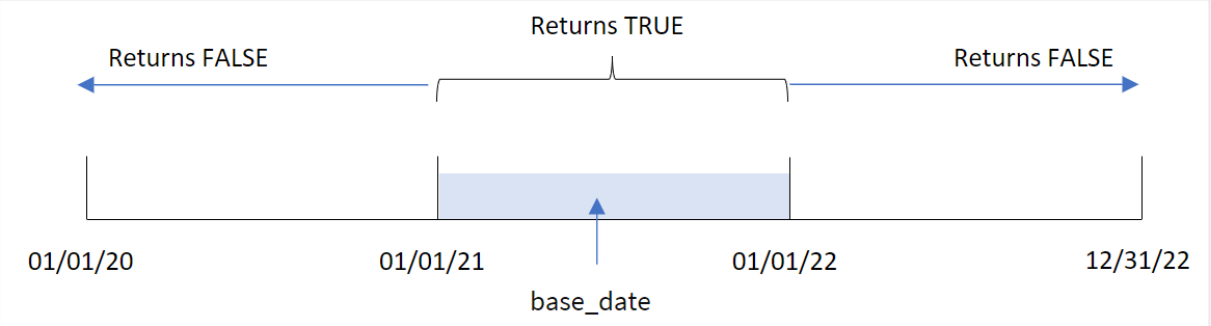
语法：

```
InYear (timestamp, base_date, period_no [, first_month_of_year])
```

返回数据类型：布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

inyear() 函数范围的图表



inyear() 函数将所选日期值与 base_date 定义的年份进行比较时，返回布尔值结果。

适用场景

inyear() 函数返回布尔值结果。通常，这种类型的函数将用作 if expression 中的条件。这将返回一个聚合或计算，具体取决于评估的日期是否发生在相关年份。例如，inyear() 函数可用于识别定义年份中发生的所有销售。

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。

可以使用以下值在 `first_month_of_year` 参数中设置一年中的第一个月：

first_month_of_year 值

月	值
二月	2
三月	3
四月	4
五月	5
六月	6
七月	7
八月	8
九月	9
十月	10
十一月	11
十二月	12

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
inyear ('01/25/2013', '01/01/2013', 0)	返回 TRUE
inyear ('01/25/2012', '01/01/2013', 0)	返回 FALSE
inyear ('01/25/2013', '01/01/2013', -1)	返回 FALSE
inyear ('01/25/2012', '01/01/2013', -1)	返回 TRUE
inyear ('01/25/2013', '01/01/2013', 0, 3)	返回 TRUE base_date 和 first_month_of_year 的值 指定时间戳 必须在 01/03/2012 和 02/28/2013 之间
inyear ('03/25/2013', '07/01/2013', 0, 3)	返回 TRUE

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 前置 Load，其中包含设置为 'in_year' 字段的 inyear() 函数，并确定哪些交易发生在 2021 年 7 月 26 日的同一年。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inyear(date,'07/26/2021', 0) as in_year
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
```

```
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_year

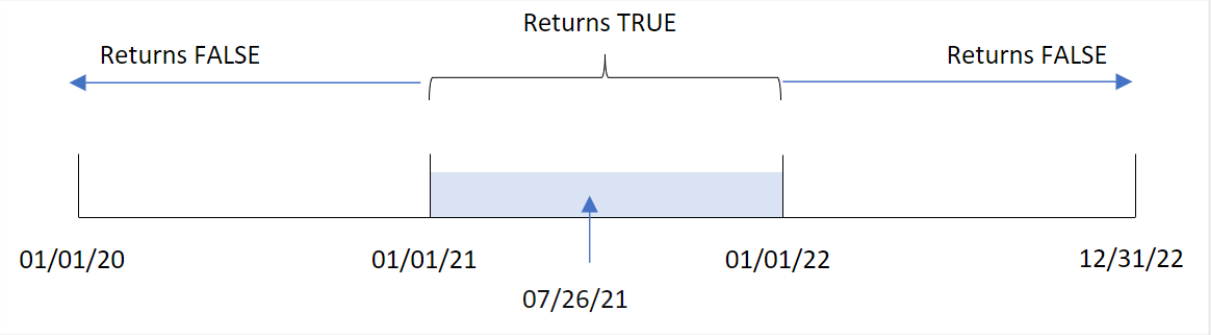
结果表

日期	in_year
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1

日期	in_year
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

in_year 字段是在前置 Load 语句中使用 inyear() 函数创建的。第一个参数标识正在评估的字段。第二个参数是 2021 年 7 月 26 日的硬编码日期，这是确定比较年的日期的 base_date。为 0 的 period_no 是最后一个参数，这意味着 inyear() 函数不比较年之前或之后的年份。

以 7 月 26 日为基准日期的 inyear() 函数范围图



2021 年发生的任何交易都会返回布尔值结果 TRUE。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 前置 Load，其中包含设置为 'previous_year' 字段的 inyear() 函数，并确定 2021 年 7 月 26 日之前发生的交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
```

```
        inyear(date,'07/26/2021', -1) as previous_year
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_year

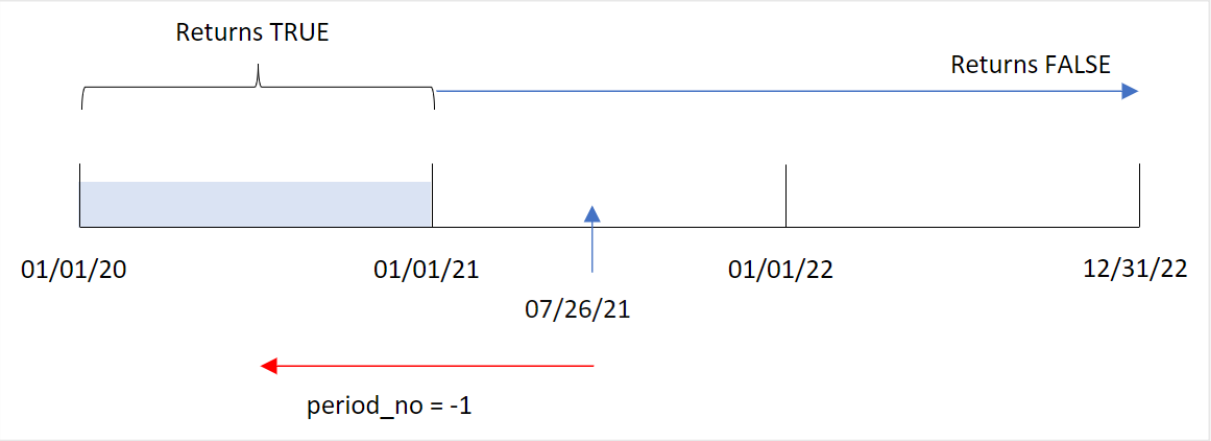
结果表

日期	previous_year
01/13/2020	-1
02/26/2020	-1
03/27/2020	-1
04/16/2020	-1
05/21/2020	-1
08/14/2020	-1
10/07/2020	-1
12/05/2020	-1

日期	previous_year
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

在 inyear() 函数中使用 -1 作为 period_no 参数，将比较年的边界向后移动一整年。2021 年最初被确定为比较年。period_no 将比较年偏移一年，使 2020 年成为比较年。

period_no 参数设置为 -1 的 inyear() 函数范围的图表



因此，2020 年发生的任何交易都会返回布尔值结果 TRUE。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 前置 Load，其中包含设置为 'in_year' 字段的 inyear() 函数，并确定哪些交易发生在 2021 年 7 月 26 日的同一年。

但是在本例中，组织策略是将三月作为财政年度的第一个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inyear(date,'07/26/2021', 0, 3) as in_year
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

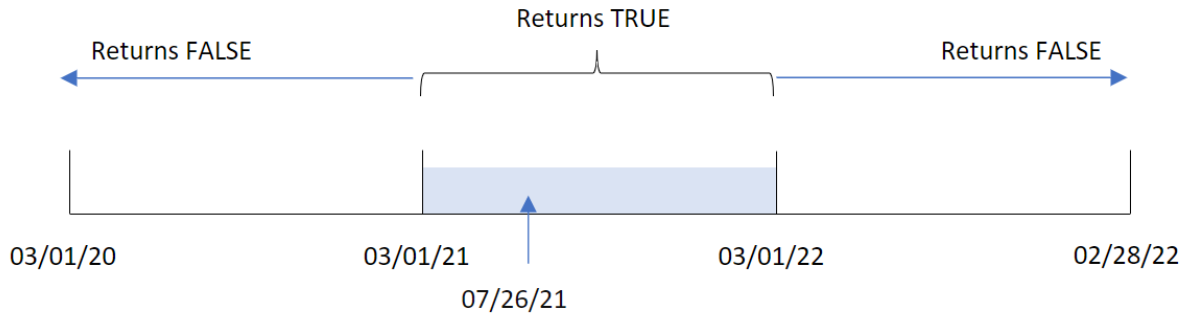
- date
- in_year

结果表

日期	in_year
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

在 `inyear()` 函数中使用 `3` 作为 `first_month_of_year` 参数，从 3 月 1 日开始，到 2 月底结束。

`inyear()` 函数范围的图表，3 月为一年中的第一个月



因此，2021 年 3 月 1 日至 2022 年 3 月 1 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。确定交易是否发生在 2021 年 7 月 26 日的同一年的计算是作为应用程序的图表对象中的度量值创建的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- date

为了计算交易是否发生在 2021 年 7 月 26 日的同一年，创建以下度量：

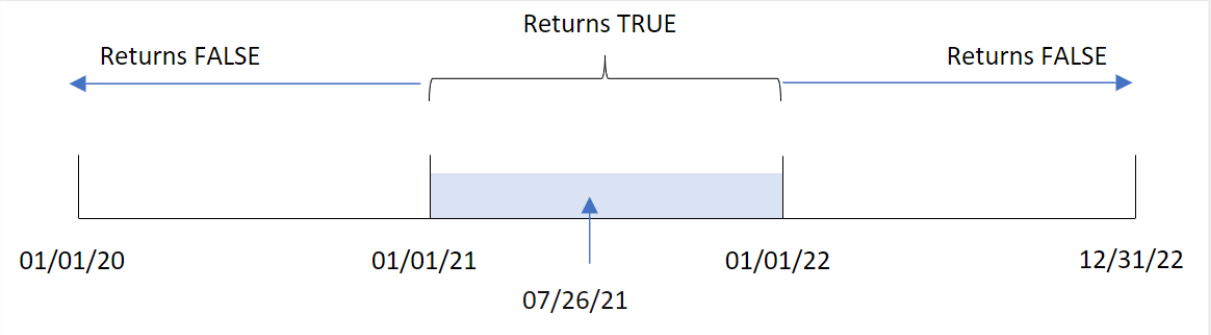
- =inyear(date,'07/26/2021', 0)

结果表

日期	=inyear(date,'07/26/2021',0)
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

使用 inyear() 函数在图表中创建 'in_year' 字段。第一个参数标识正在评估的字段。第二个参数是 2021 年 7 月 26 日的硬编码日期，这是确定比较年的日期的 base_date。为 0 的 period_no 是最后一个参数，这意味着 inyear() 函数不比较年之前或之后的年份。

以 7 月 27 日为基准日期的 inyear() 函数范围图



2021 年发生的任何交易都会返回布尔值结果 TRUE。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 'Products' 的表中的数据集。
- 表格包含以下字段：
  - 产品 ID
  - 产品类型
  - 制造日期
  - 成本价

最终用户想要一个图表对象，按产品类型显示 2021 年制造的产品的成本。

加载脚本

```
Products:
Load
*
Inline
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'01/13/2020',37.23
8189,product B,'02/26/2020',17.17
8190,product B,'03/27/2020',88.27
8191,product C,'04/16/2020',57.42
8192,product D,'05/21/2020',53.80
8193,product D,'08/14/2020',82.06
8194,product C,'10/07/2020',40.39
8195,product B,'12/05/2020',87.21
8196,product A,'01/22/2021',95.93
8197,product B,'02/03/2021',45.89
```

```
8198,product C,'03/17/2021',36.23
8199,product C,'04/23/2021',25.66
8200,product B,'05/04/2021',82.77
8201,product D,'06/30/2021',69.98
8202,product D,'07/26/2021',76.11
8203,product D,'12/27/2021',25.12
8204,product C,'06/06/2022',46.23
8205,product C,'07/18/2022',84.21
8206,product A,'11/14/2022',96.24
8207,product B,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

- product_type

要计算 2021 生产的每种产品的总和，请创建以下度量：

- =sum(if(InYear(manufacture_date,makedate(2021,01,01),0),cost_price,0))

执行以下操作：

1. 将度量的**数字格式**设置为**金额**。
2. 在**外观**下，关闭**总计**。

结果表

product_type	=sum(if(InYear(manufacture_date,makedate(2021,01,01),0),cost_price,0))
product A	$95.93
product B	$128.66
product C	$61.89
product D	$171.21

inyear() 函数在评估每个产品的制造日期时返回布尔值。对于 2021 年生产的任何产品，inyear() 函数返回布尔值 TRUE，并显示 cost_price 的总和。

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。

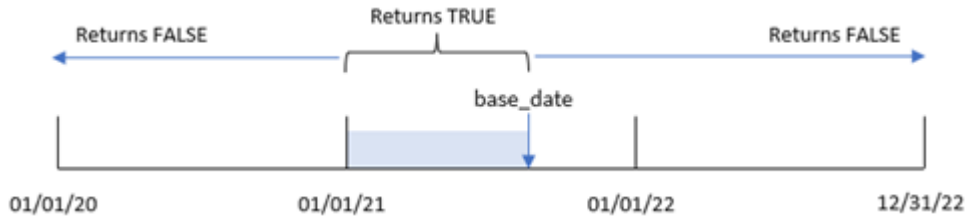
语法：

```
InYearToDate (timestamp, base_date, period_no[, first_month_of_year])
```

返回数据类型： 布尔值

> *在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。*

inyeartodate 函数图表



inyeartodate() 函数将使用 base_date 分割该年的特定部分，识别该年段的最大允许日期。然后，该函数计算日期字段或值是否属于该段，并返回布尔值结果。

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。

适用场景

inyeartodate() 函数返回布尔结果。通常，这种类型的函数将用作 if 表达式中的条件。这将返回一个聚合或计算，这取决于评估日期是否发生在该日期之前的年份。

例如，inyeartodate() 函数可用于识别截至特定日期的一个年内制造的所有设备。

以下示例使用日期格式 MM/DD/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。

函数示例

示例	结果
inyeartodate ('01/25/2013', '02/01/2013', 0)	返回 TRUE。
inyeartodate ('01/25/2012', '01/01/2013', 0)	返回 FALSE。

示例	结果
inyeartodate ('01/25/2012', '02/01/2013', -1)	返回 TRUE。
inyeartodate ('11/25/2012', '01/31/2013', 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 ('3/31/2013', '01/31/2013', 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。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 in_year_to_date，用于确定截至 2021 年 7 月 26 日的年内发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inyeartodate(date,'07/26/2021', 0) as in_year_to_date
        ;
Load
```

```
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- in_year_to_date

结果表

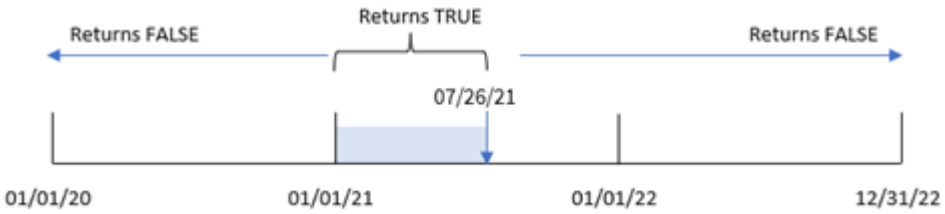
日期	in_year_to_date
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	-1
02/03/2021	-1

日期	in_year_to_date
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

in_year_to_date 字段是在前置 Load 语句中使用 inyeartodate() 函数创建的。提供的第一个参数标识正在评估的字段。

第二个参数是 2021年 7 月 26 日的硬编码日期，这是标识年份段结束边界的 base_date。为 0 的 period_no 是最后一个参数，这意味着该函数不比较分段年之前或之后的年。

inyeartodate 函数图表，没有额外参数



1 月 1 日至 7 月 26 日之间发生的任何交易都将返回布尔值结果 TRUE。2021 之前和 2021 年 7 月 26 日之后的交易日期返回 FALSE。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_year_to_date，确定在 2021 年 7 月 26 日结束的年段之前的整个年发生了哪些交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inyeartodate(date,'07/26/2021', -1) as previous_year_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_year_to_date

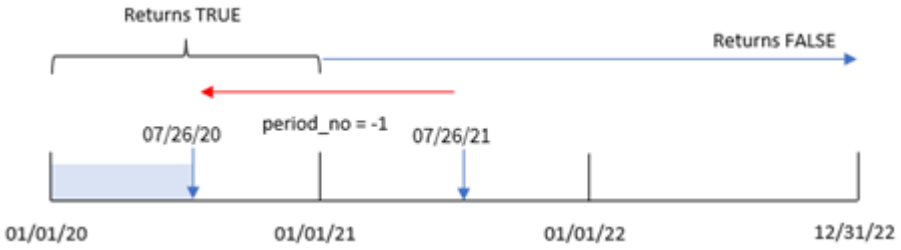
结果表

日期	previous_year_to_date
01/13/2020	-1

日期	previous_year_to_date
02/26/2020	-1
03/27/2020	-1
04/16/2020	-1
05/21/2020	-1
06/14/2020	-1
08/07/2020	0
09/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

值为 -1 的 period_no 指示 inyeartodate () 函数将输入季度段与前一年进行比较。输入日期为 2021 年 7 月 26 日，2021 年 1 月 1 日至 2021年 7 月 26 日段最初被确定为年初至今。然后 period_no 将该段提前整年偏移，导致日期边界变为 2020 年 1 月 1 日至 7 月 26 日。

inyeartodate 函数 period_no 示例的图表



因此，2020 年 1 月 1 日至 7 月 26 日之间发生的任何交易都将返回布尔值结果 TRUE。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 in_year_to_date，用于确定截至 2021 年 7 月 26 日的同一年内发生了哪些交易。

在本例中，我们将 3 月设置为会计年度的第一个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        inyeartodate(date,'07/26/2021', 0,3) as in_year_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

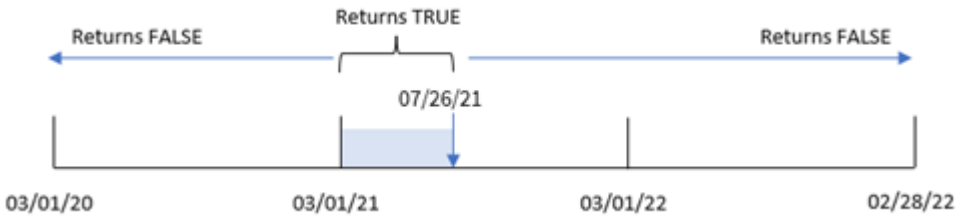
- date
- in_year_to_date

结果表

日期	in_year_to_date
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

通过在 inyeartodate() 函数中使用 3 作为 first_month_of_year 参数，函数从 3 月 1 日开始年份。然后，2021 年 7 月 26 日的 base_date 设置该年份段的结束日期。

inyeartodate 函数 first_month_of_year 示例的图表



因此，在 2021 年 3 月 1 日和 7 月 26 日之间发生的任何交易将返回布尔结果 TRUE，而日期在这些边界之外的交易将返回值 FALSE。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。确定截至 2021 年 7 月 26 日在同一年发生的交易的计算是作为应用程序中图表对象中的度量创建的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
```

```
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量：

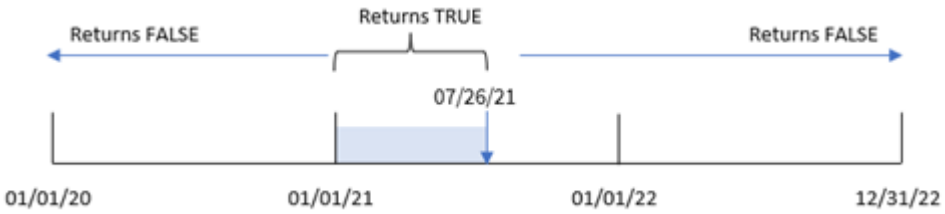
```
=inyeartodate(date,'07/26/2021', 0)
```

结果表

日期	=inyeartodate(date,'07/26/2021', 0)
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

通过使用 inyeartodate() 函数在图表对象中创建 in_year_to_date 度量。提供的第一个参数标识正在评估的字段。第二个参数是 2021年 7 月 26 日的硬编码日期，这是标识比较年份段结束边界的 base_date。为 0 的 period_no 是最后一个参数，这意味着该函数不比较分段年之前或之后的年。

inyeartodate 函数的图表，图表对象示例



2021 年 1 月 1 日至 7 月 26 日之间发生的任何交易都将返回布尔值结果 TRUE。2021 之前和 2021 年 7 月 26 日之后的交易日期返回 FALSE。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Products 的表中的数据集。
- 有关产品 ID、产品类型、制造日期和成本价格的信息。

最终用户想要一个图表对象，按产品类型显示截至 2021 年 7 月 26 日制造的产品的成本。

加载脚本

```
Products:
Load
*
Inline
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'01/13/2020',37.23
8189,product B,'02/26/2020',17.17
8190,product B,'03/27/2020',88.27
8191,product C,'04/16/2020',57.42
8192,product D,'05/21/2020',53.80
8193,product D,'08/14/2020',82.06
8194,product C,'10/07/2020',40.39
8195,product B,'12/05/2020',87.21
8196,product A,'01/22/2021',95.93
8197,product B,'02/03/2021',45.89
8198,product C,'03/17/2021',36.23
8199,product C,'04/23/2021',25.66
8200,product B,'05/04/2021',82.77
8201,product D,'06/30/2021',69.98
8202,product D,'07/26/2021',76.11
8203,product D,'12/27/2021',25.12
8204,product C,'06/06/2022',46.23
```

```
8205,product C,'07/18/2022',84.21
8206,product A,'11/14/2022',96.24
8207,product B,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：product_type。

创建一个度量值，计算 2021 年 7 月 27 日之前制造的每种产品的总和：

```
=sum(if(inyeartodate(manufacture_date,makedate(2021,07,26),0),cost_price,0))
```

将度量的**数字格式**设置为**金额**。

结果表

product_type	=sum(if(inyeartodate(manufacture_date,makedate (2021,07,26),0),cost_price,0))
product A	$95.93
product B	$128.66
product C	$61.89
product D	$146.09

inyeartodate() 函数在评估每个产品的制造日期时返回布尔值。对于 2021 年 7 月 27 日前生产的任何产品，inyeartodate() 函数返回布尔值 TRUE，并对 cost_price 求和。

产品 D 是 2021年 7 月 26 日之后生产的唯一产品。带有 product_ID 8203 的条目于 12 月 27 日制造，成本为 25.12 美元。因此，该成本不包括在图表对象中产品 D 的总成本中。

lastworkdate

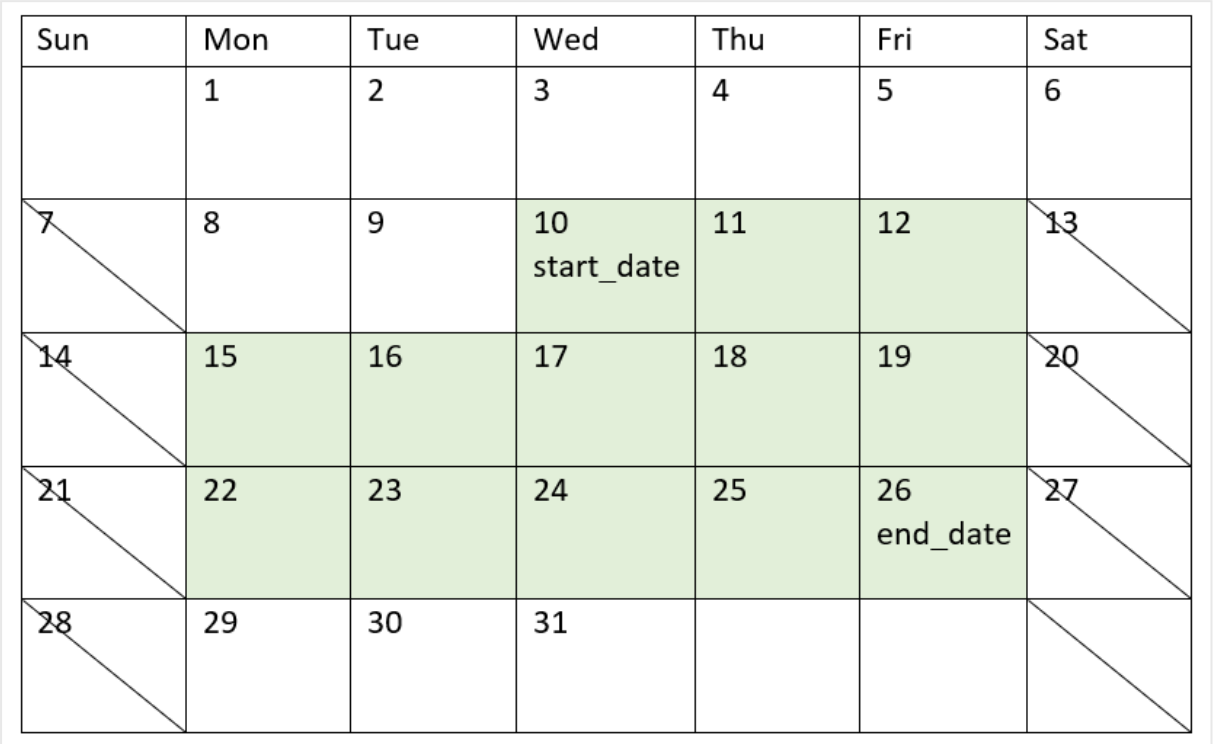
lastworkdate 函数用于返回最早的结束日以获得 **no_of_workdays**（周一至周五），如果在 **start_date** 开始考虑任何列出的可选 **holiday**。**start_date** 和 **holiday** 应是有效的日期或时间戳。

语法：

```
lastworkdate(start_date, no_of_workdays {, holiday})
```

返回数据类型： 整数

显示如何使用 lastworkdate() 函数的日历



Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10 start_date	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26 end_date	27
28	29	30	31			

限制

除了周一开始到周五结束的工作周之外，没有任何方法可以修改区域或场景的 lastworkdate() 函数。

假日参数必须是字符串常数。它不接受表达式。

适用场景

lastworkdate() 函数通常用作表达式的一部分，当用户希望根据项目开始的时间和该期间将发生的假期计算项目或作业的建议结束日期时。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

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。假日表示为字符串常量日期。您可以指定多个假期日期，以逗号分隔。 **示例：** `'12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'`

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含项目 ID、项目开始日期和项目所需的估计工作量（以天为单位）的数据集。数据集加载到名为 'Projects' 的表中。
- 包含设置为字段 'end_date' 并标识每个项目计划何时结束的 `lastworkdate()` 函数的前置 Load。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
    Load
       *,
       LastWorkDate(start_date,effort) as end_date
       ;
Load
id,
start_date,
effort
Inline
[
id,start_date,effort
1,01/01/2022,14
2,02/10/2022,17
```

```
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- effort
- end_date

结果表

id	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/23/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

由于没有计划的假日，该函数将定义的工作日数(星期一到星期五)添加到开始日期，以找到最早可能的结束日期。

以下日历显示项目 3 的开始和结束日期，工作日以绿色突出显示。

显示项目 3 开始和结束日期的日历

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18	19	20	21
22	23 End Date	24	25	26	27	28
29	30	31				

示例 2 - 单假期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含项目 ID、项目开始日期和项目所需的估计工作量（以天为单位）的数据集。数据集加载到名为 'Projects' 的表中。
- 包含设置为字段 'end_date' 并标识每个项目计划何时结束的 `lastworkdate()` 函数的前置 Load。

然而，2022 年 5 月 18 日有一个假期。前置 Load 中的 `lastworkdate()` 函数在其第三个参数中包含假日，以标识每个项目计划何时结束。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
    Load
       *,
       LastWorkDate(start_date,effort, '05/18/2022') as end_date
       ;
Load
id,
start_date,
effort
Inline
[
id,start_date,effort
1,01/01/2022,14
2,02/10/2022,17
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- effort
- end_date

结果表

id	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/24/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

单个计划假日作为 `lastworkdate()` 函数中的第三个参数输入。因此，项目 3 的结束日期推迟了一天，因为假期发生在结束日期之前的一个工作日。

下面的日历显示项目 3 的开始和结束日期，并显示假日将项目的结束日期更改一天。

显示项目 3 开始和结束日期的日历，5 月 18 日为假日

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18 Holiday	19	20	21
22	23	24 End Date	25	26	27	28
29	30	31				

示例 3 - 多个假期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含项目 ID、项目开始日期和项目所需的估计工作量（以天为单位）的数据集。数据集加载到名为 'Projects' 的表中。
- 包含设置为字段 'end_date' 并标识每个项目计划何时结束的 `lastworkdate()` 函数的前置 Load。

然而，5 月 19 日、20 日、21 日和 22 日有三个假期。前置 Load 中的 `lastworkdate()` 函数在其第三个参数中包含每个假日，以确定每个项目计划何时结束。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
    Load
       *,
       LastWorkDate(start_date,effort, '05/19/2022','05/20/2022','05/21/2022','05/22/2022') as
end_date
       ;
Load
id,
start_date,
effort
Inline
[
id,start_date,effort
1,01/01/2022,14
2,02/10/2022,17
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- effort
- end_date

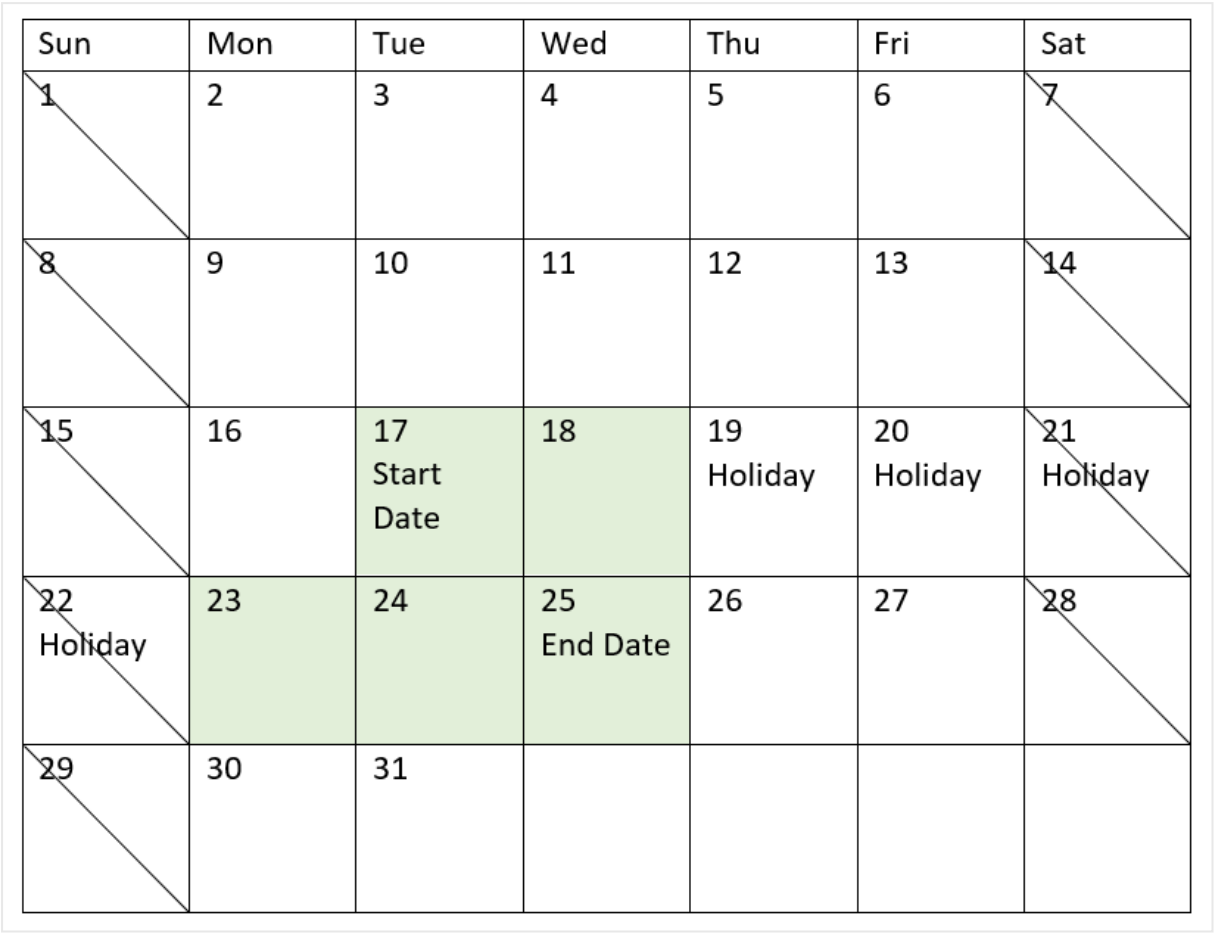
结果表

id	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/25/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

这四个假期在开始日期和工作日数之后作为参数列表输入 lastworkdate() 函数中。

以下日历显示项目 3 的开始和结束日期，并显示节假日将项目的结束日期更改三天。

显示项目 3 的开始和结束日期的日历，假期为 5 月 19 日至 22 日



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18	19 Holiday	20 Holiday	21 Holiday
22 Holiday	23	24	25 End Date	26	27	28
29	30	31				

示例 4 - 单假期(图表)

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。end_date 字段作为图表对象中的度量计算。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
Load
id,
start_date,
effort
Inline
[
```

```
id,start_date,effort
1,01/01/2022,14
2,02/10/2022,17
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- effort

要计算 end_date，创建以下度量：

- =LastWorkDate(start_date,effort,'05/18/2022')

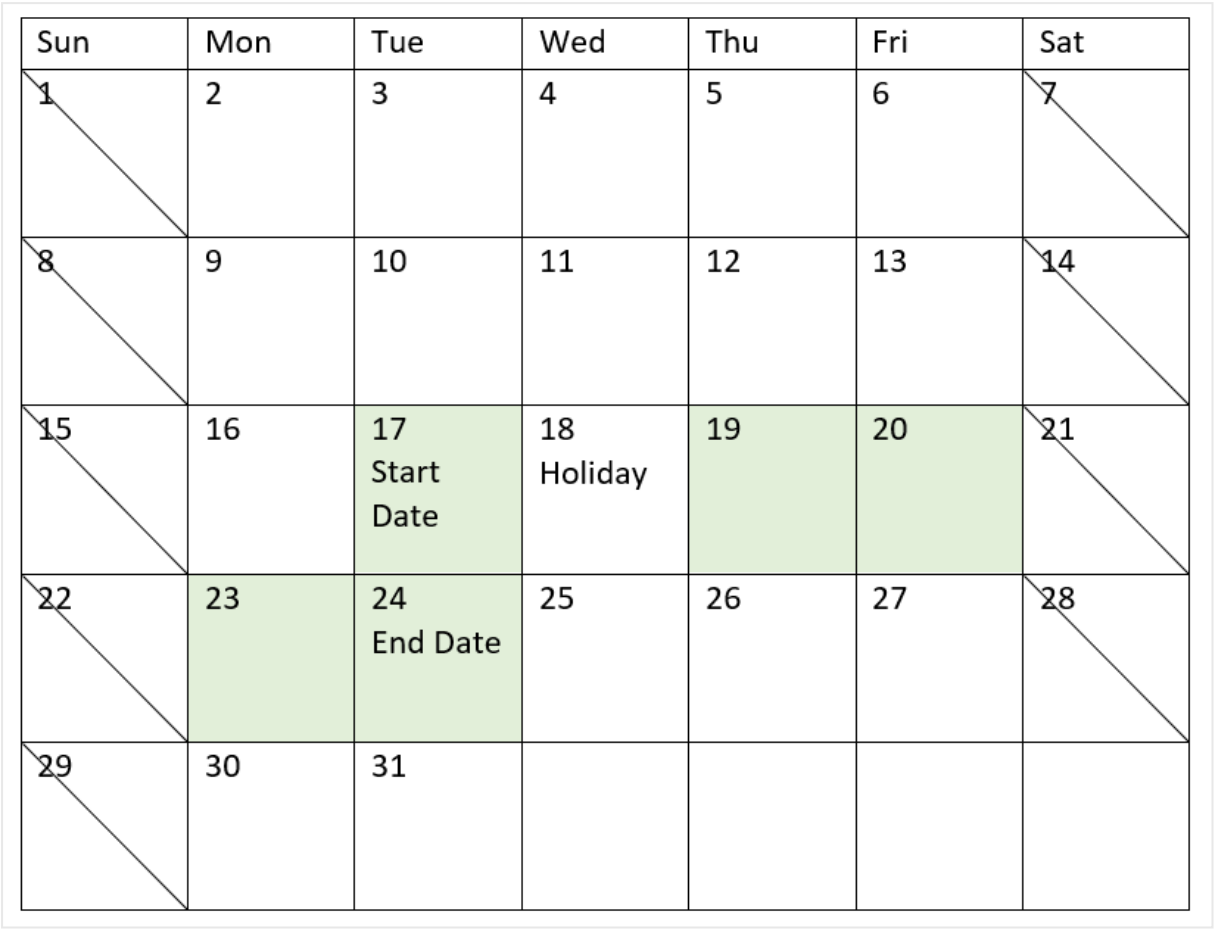
结果表

id	start_date	effort	=LastWorkDate(start_date,effort,'05/18/2022')
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/23/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

在图表中输入单个计划假日作为度量。因此，项目 3 的结束日期推迟了一天，因为假期发生在结束日期之前的一个工作日。

下面的日历显示项目 3 的开始和结束日期，并显示假日将项目的结束日期更改一天。

显示项目 3 开始和结束日期的日历，5 月 18 日为假日



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18 Holiday	19	20	21
22	23	24 End Date	25	26	27	28
29	30	31				

localtime

此函数用于返回指定时区的当前时间戳。

语法：

```
LocalTime([timezone [, ignoreDST ]])
```

返回数据类型：双

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。下表还列出了可接受的地点和时区列表。 如果未指定时区，则返回本地时间。 *如果使用 DST 偏移量(即，指定的 **ignoreDST** 参数值计算为 False)，则必须在 **place** 参数中指定一个位置，而不是 GMT 偏移量。这是因为调整夏令时除了需要 GMT 偏移量提供的纵向信息外，还需要纬度信息。有关更多信息，请参阅将 GMT 偏移量与夏令时结合使用 (page 807)。*
ignoreDST	如果此参数计算为 True，则忽略 DST(夏令时)。有效的参数值计算为 True，包括 -1 和 True()。 如果此参数计算为 False，则会根据夏令时调整时间戳。有效的参数值计算为 False，包括 0 和 False()。 如果 **ignoreDST** 参数值无效，则函数将计算表达式，如同 **ignore_dst** 值计算为 True 一样。如果未指定 **ignoreDST** 参数值，则函数将计算表达式，如同 **ignore_dst** 值计算为 False. 一样。

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

A-C	D-K	L-R	S-Z
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
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：

以下示例基于在当地时间 2023-08-14 08:39:47 调用的函数，其中服务器或桌面环境的本地时区为 GMT-05:00，并且在截至所列日期已实施夏令时的地区中。

脚本示例

示例	结果
`localtime ()`	返回本地时间 2023-08-14 08:39:47。
`localtime ('London')`	返回伦敦当地时间，2023-08-14 13:39:47.
`localtime ('GMT+02:00')`	返回 GMT+02:00 时区的本地时间 2023-08-14 14:39:47。由于指定了 GMT 偏移量，而不是位置，因此不会对夏令时进行调整。
`localtime ('Paris',-1)`	返回巴黎本地时间 2023-08-14 13:39:47，忽略夏令时。
`localtime ('Paris',True())`	返回巴黎本地时间 2023-08-14 13:39:47，忽略夏令时。
`localtime ('Paris',0)`	返回巴黎本地时间 2023-08-14 14:39:47，考虑夏令时。
`localtime ('Paris',False ())`	返回巴黎本地时间 2023-08-14 14:39:47，考虑夏令时。

将 GMT 偏移量与夏令时结合使用

在 Qlik Sense 实施 Unicode (ICU) 库的国际组件之后，将 GMT（格林尼治标准时间）偏移量与 DST（夏令时）结合使用需要额外的纬度信息。

GMT 是纵向（东西）偏移，而夏令时是横向（南北）偏移。例如，赫尔辛基（芬兰）和约翰内斯堡（南非）共享相同的 GMT+02:00 偏移量，但它们不共享相同的夏令时偏移量。这意味着，除了 GMT 偏移之外，任何夏令时偏移都需要关于当地时区的纬度位置的信息（地理时区输入），以便获得关于当地夏令时条件的完整信息。

lunarweekend

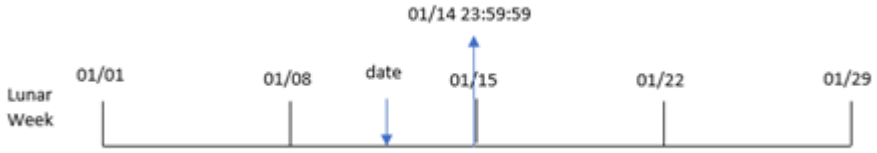
此函数用于返回与包含 **date** 的阴历周的最后毫秒的时间戳对应的值。Qlik Sense 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

语法：

```
LunarweekEnd(date[, period_no[, first_week_day]])
```

返回数据类型： 双

lunarweekend() 函数的示例图表



lunarweekend() 函数决定 date 属于哪个农历周。然后以日期格式返回该周最后一毫秒的时间戳。

参数

参数	说明
date	要评估的日期或时间戳。
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。

适用场景

当用户希望计算使用当周尚未发生的分数时，lunarweekend() 函数通常用作表达式的一部分。与 weekend() 函数不同，每个日历年的最后一个农历周将于 12 月 31 日结束。例如，lunarweekend() 函数可用于计算一周内尚未发生的利息。

函数示例

示例	结果
lunarweekend('01/12/2013')	返回 01/14/2013 23:59:59。
lunarweekend('01/12/2013', -1)	返回 01/07/2013 23:59:59。
lunarweekend('01/12/2013', 0, 1)	返回 01/15/2013 23:59:59。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建字段 end_of_week，返回交易发生的农历周结束的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekend(date) as end_of_week,
        timestamp(lunarweekend(date)) as end_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- end_of_week
- end_of_week_timestamp

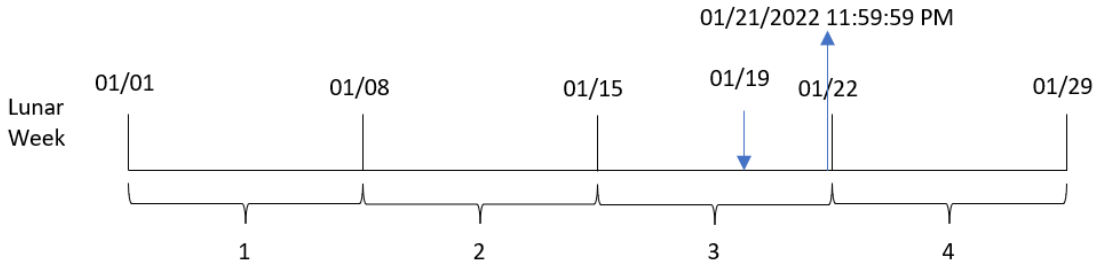
结果表

日期	end_of_week	end_of_week_timestamp
1/7/2022	01/07/2022	1/7/2022 11:59:59 PM
1/19/2022	01/21/2022	1/21/2022 11:59:59 PM
2/5/2022	02/11/2022	2/11/2022 11:59:59 PM
2/28/2022	03/04/2022	3/4/2022 11:59:59 PM
3/16/2022	03/18/2022	3/18/2022 11:59:59 PM
4/1/2022	04/01/2022	4/1/2022 11:59:59 PM
5/7/2022	05/13/2022	5/13/2022 11:59:59 PM
5/16/2022	05/20/2022	5/20/2022 11:59:59 PM
6/15/2022	06/17/2022	6/17/2022 11:59:59 PM
6/26/2022	07/01/2022	7/1/2022 11:59:59 PM
7/9/2022	07/15/2022	7/15/2022 11:59:59 PM
7/22/2022	07/22/2022	7/22/2022 11:59:59 PM
7/23/2022	07/29/2022	7/29/2022 11:59:59 PM
7/27/2022	07/29/2022	7/29/2022 11:59:59 PM
8/2/2022	08/05/2022	8/5/2022 11:59:59 PM
8/8/2022	08/12/2022	8/12/2022 11:59:59 PM
8/19/2022	08/19/2022	8/19/2022 11:59:59 PM
9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
10/14/2022	10/14/2022	10/14/2022 11:59:59 PM
10/29/2022	11/04/2022	11/4/2022 11:59:59 PM

通过使用 `lunarweekend()` 函数并将 `end_of_week` 字段作为函数的参数传递，在前置 Load 语句中创建了 `date` 字段。

`lunarweekend()` 函数标识日期值属于哪个农历周，并返回该周最后一毫秒的时间戳。

lunarweekend() 函数图表，示例没有额外参数



交易 8189 发生在 1 月 19 日。lunarweekend() 函数确定农历周从 1 月 15 日开始。因此，该交易 end_of_week 的值返回农历周的最后一毫秒，即 1 月 21 日下午 11:59:59。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_lunar_week_end，该字段返回事务发生前的农历周末的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekend(date,-1) as previous_lunar_week_end,
        timestamp(lunarweekend(date,-1)) as previous_lunar_week_end_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
```

```
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_lunar_week_end
- previous_lunar_week_end_timestamp

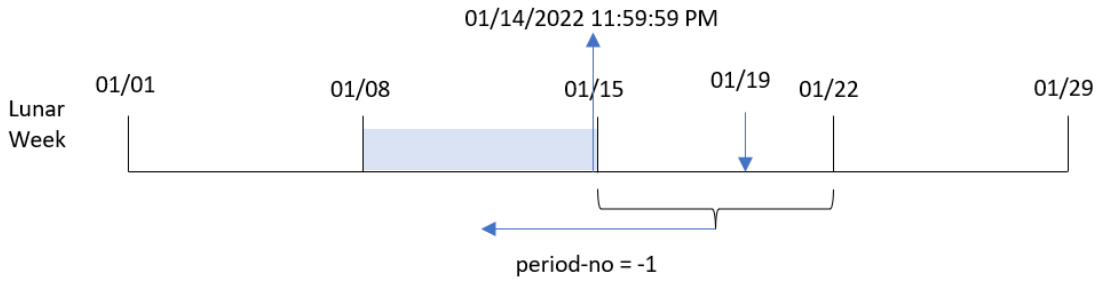
结果表

日期	previous_lunar_week_end	previous_lunar_week_end_timestamp
1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
1/19/2022	01/14/2022	1/14/2022 11:59:59 PM
2/5/2022	02/04/2022	2/4/2022 11:59:59 PM
2/28/2022	02/25/2022	2/25/2022 11:59:59 PM
3/16/2022	03/11/2022	3/18/2022 11:59:59 PM
4/1/2022	03/25/2022	3/25/2022 11:59:59 PM
5/7/2022	05/06/2022	5/6/2022 11:59:59 PM
5/16/2022	05/13/2022	5/13/2022 11:59:59 PM
6/15/2022	06/10/2022	6/10/2022 11:59:59 PM
6/26/2022	06/24/2022	6/24/2022 11:59:59 PM
7/9/2022	07/08/2022	7/8/2022 11:59:59 PM
7/22/2022	07/15/2022	7/15/2022 11:59:59 PM
7/23/2022	07/22/2022	7/22/2022 11:59:59 PM
7/27/2022	07/22/2022	7/22/2022 11:59:59 PM
8/2/2022	07/29/2022	7/29/2022 11:59:59 PM
8/8/2022	08/05/2022	8/5/2022 11:59:59 PM
8/19/2022	08/12/2022	8/12/2022 11:59:59 PM

日期	previous_lunar_week_end	previous_lunar_week_end_timestamp
9/26/2022	09/23/2022	9/23/2022 11:59:59 PM
10/14/2022	10/07/2022	10/7/2022 11:59:59 PM
10/29/2022	10/28/2022	10/28/2022 11:59:59 PM

在本例中，由于 lunarweekend() 函数中使用了为 -1 的 period_no 作为偏移量参数，因此该函数首先标识交易发生的农历周。然后，它在一周前移动，并确定农历周的最后一毫秒。

lunarweekend() 函数的图表，period_no 示例



交易 8189 发生在 1 月 19 日。lunarweekend() 函数确定农历周从 1 月 15 日开始。因此，上一个农历周开始于 1 月 8 日，结束于 1 月 14 日晚上 11:59:59；这是为 previous_lunar_week_end 字段返回的值。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在这个例子中，我们将农历周设置为 1 月 6 日开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekend(date,0,4) as end_of_week,
timestamp(lunarweekend(date,0,4)) as end_of_week_timestamp
;
Load
*
Inline
[
```

```
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- end_of_week
- end_of_week_timestamp

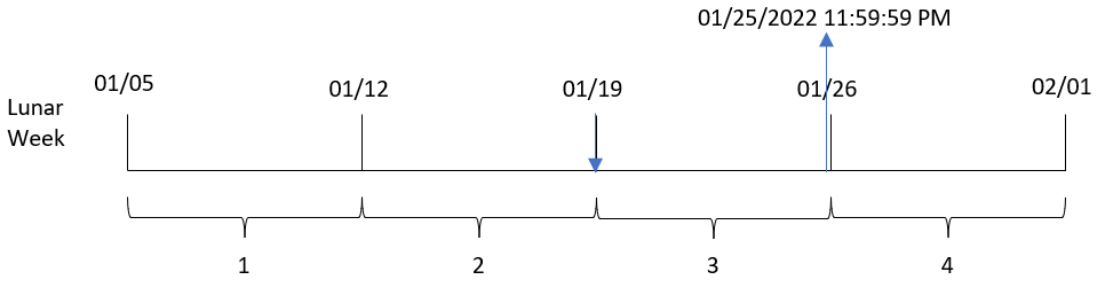
结果表

日期	end_of_week	end_of_week_timestamp
1/7/2022	01/11/2022	1/11/2022 11:59:59 PM
1/19/2022	01/25/2022	1/25/2022 11:59:59 PM
2/5/2022	02/08/2022	2/8/2022 11:59:59 PM
2/28/2022	03/01/2022	3/1/2022 11:59:59 PM
3/16/2022	03/22/2022	3/22/2022 11:59:59 PM
4/1/2022	04/05/2022	4/5/2022 11:59:59 PM
5/7/2022	05/10/2022	5/10/2022 11:59:59 PM
5/16/2022	05/17/2022	5/17/2022 11:59:59 PM
6/15/2022	06/21/2022	6/21/2022 11:59:59 PM
6/26/2022	06/28/2022	6/28/2022 11:59:59 PM
7/9/2022	07/12/2022	7/12/2022 11:59:59 PM

日期	end_of_week	end_of_week_timestamp
7/22/2022	07/26/2022	7/26/2022 11:59:59 PM
7/23/2022	07/26/2022	7/26/2022 11:59:59 PM
7/27/2022	08/02/2022	8/2/2022 11:59:59 PM
8/2/2022	08/02/2022	8/2/2022 11:59:59 PM
8/8/2022	08/09/2022	8/9/2022 11:59:59 PM
8/19/2022	08/23/2022	8/23/2022 11:59:59 PM
9/26/2022	09/27/2022	9/27/2022 11:59:59 PM
10/14/2022	10/18/2022	10/18/2022 11:59:59 PM
10/29/2022	11/01/2022	11/1/2022 11:59:59 PM

在本例中，由于 `lunarweekend()` 函数中使用了参数 `first_week_date`，因此它将年初从 1 月 1 日偏移到 1 月 5 日。

`lunarweekend()` 函数的图表，first_week_day 示例



交易 8189 发生在 1 月 19 日。由于农历周从 1 月 5 日开始，`lunarweekend()` 函数确定包含 1 月 19 日的农历周也从 1 月 19 号开始。因此，农历周的结束发生在 1 月 25 日晚上 11:59:59；这是为 `end_of_week` 字段返回的值。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生的农历周结束时间戳的计算在应用程序的图表对象中创建为度量。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

添加以下度量：

```
=lunarweekend(date)
```

```
=timestamp(lunarweekend(date))
```

结果表

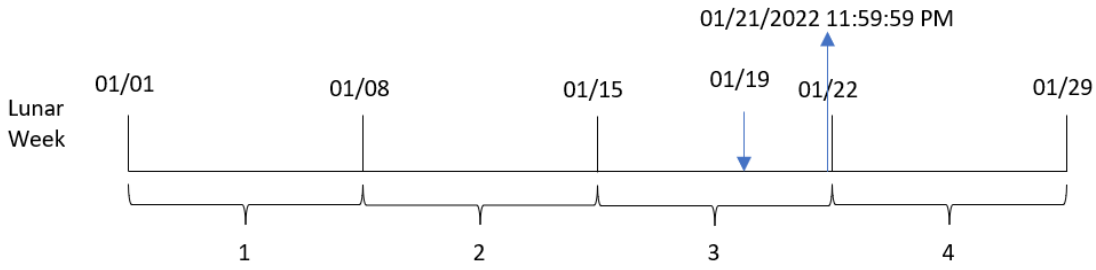
日期	=lunarweekend(date)	=timestamp(lunarweekend(date))
1/7/2022	01/07/2022	1/7/2022 11:59:59 PM
1/19/2022	01/21/2022	1/21/2022 11:59:59 PM
2/5/2022	02/11/2022	2/11/2022 11:59:59 PM
2/28/2022	03/04/2022	3/4/2022 11:59:59 PM
3/16/2022	03/18/2022	3/18/2022 11:59:59 PM
4/1/2022	04/01/2022	4/1/2022 11:59:59 PM

日期	=lunarweekend(date)	=timestamp(lunarweekend(date))
5/7/2022	05/13/2022	5/13/2022 11:59:59 PM
5/16/2022	05/20/2022	5/20/2022 11:59:59 PM
6/15/2022	06/17/2022	6/17/2022 11:59:59 PM
6/26/2022	07/01/2022	7/1/2022 11:59:59 PM
7/9/2022	07/15/2022	7/15/2022 11:59:59 PM
7/22/2022	07/22/2022	7/22/2022 11:59:59 PM
7/23/2022	07/29/2022	7/29/2022 11:59:59 PM
7/27/2022	07/29/2022	7/29/2022 11:59:59 PM
8/2/2022	08/05/2022	8/5/2022 11:59:59 PM
8/8/2022	08/12/2022	8/12/2022 11:59:59 PM
8/19/2022	08/19/2022	8/19/2022 11:59:59 PM
9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
10/14/2022	10/14/2022	10/14/2022 11:59:59 PM
10/29/2022	11/04/2022	11/4/2022 11:59:59 PM

通过使用 `lunarweekend()` 函数并将 `date` 字段作为函数的参数传递，在图表对象中创建 `end_of_week` 度量。

`lunarweekend()` 函数标识日期值属于哪个农历周，并返回该周最后一毫秒的时间戳。

`lunarweekend()` 函数的图表，图表对象示例



交易 8189 发生在 1 月 19 日。`lunarweekend()` 函数确定农历周从 1 月 15 日开始。因此，该交易 `end_of_week` 的值返回农历周的最后一毫秒，即 1 月 21 日下午 11:59:59。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Employee_Expenses 的表中的数据集。
- 每个员工的员工 ID、员工姓名和平均每日费用报销。

最终用户需要一个图表对象，该图表对象按员工 ID 和员工姓名显示农历周剩余时间内仍要发生的估计费用索赔。

加载脚本

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 添加以下字段作为维度：
    - employee_id
    - employee_name
3. 接下来，创建以下度量来计算累计利息：
   `=(lunarweekend(today(1))-today(1))*avg_daily_claim`
4. 将度量的**数字格式**设置为**金额**。

结果表

EmployeeID	employee_name	=(lunarweekend(today(1))-today(1))*avg_daily_claim
182	Mark	$75.00

EmployeeID	employee_name	=(lunarweekend(today(1))-today(1))*avg_daily_claim
183	Deryck	$62.50
184	Dexter	$62.50
185	Sydney	$135.00
186	Agatha	$90.00

lunarkweekend() 函数使用今天的日期作为唯一参数，返回当前农历周的结束日期。然后，通过从农历周结束日期中减去今天的日期，表达式返回本周剩余的天数。

然后将该值乘以每个员工的平均每日费用索赔，以计算每个员工在剩余农历周预计提出的索赔的估计值。

lunarweekname

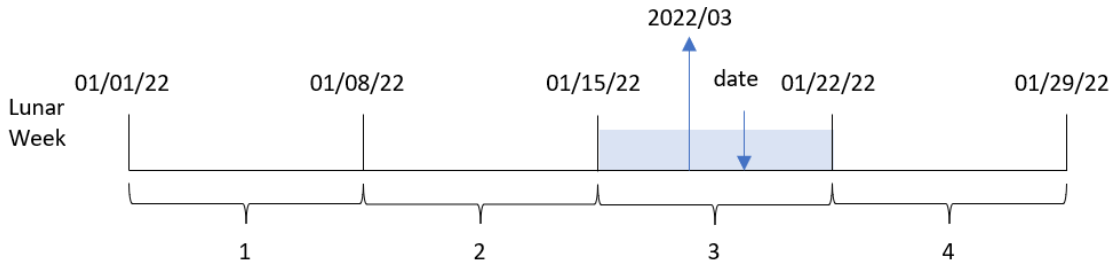
此函数用于返回一个显示值，显示与包含 **date** 的阴历周的第一天的第一毫秒时间戳对应的年份和阴历周数。Qlik Sense 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

语法：

```
LunarWeekName(date [, period_no[, first_week_day]])
```

返回数据类型：双

lunarweekname() 函数的示例图表



lunarweekname() 函数确定日期属于哪个农历周，从 1 月 1 日开始计算周数。然后返回一个包含 year/weekcount 的值。

参数

参数	说明
date	要评估的日期或时间戳。
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。

参数	说明
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。

适用场景

当您希望按农历周比较聚合时，lunarweekname() 函数非常有用。例如，该函数可用于确定农历周产品的总销售额。如果您希望确保一年中第一周包含的所有值最早只包含从 1 月 1 日开始的值，农历周非常有用。

通过使用函数在主日历表中创建字段，可以在 Load 脚本中创建这些维度。该函数也可以直接在图表中用作计算维度。

函数示例

示例	结果
`lunarweekname('01/12/2013')`	返回 2006/02。
`lunarweekname('01/12/2013', -1)`	返回 2006/01。
`lunarweekname('01/12/2013', 0, 1)`	返回 2006/02。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数的日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

- 创建字段 lunar_week_name，返回交易发生的农历周的年和周数字。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekname(date) as lunar_week_name
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- lunar_week_name

结果表

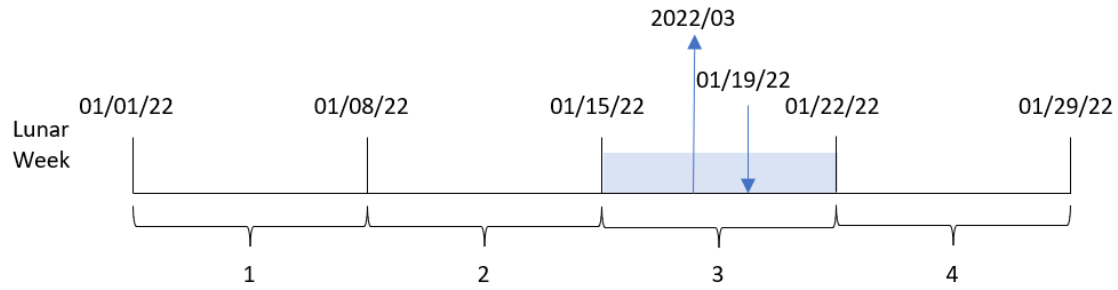
日期	lunar_week_name
1/7/2022	2022/01
1/19/2022	2022/03

日期	lunar_week_name
2/5/2022	2022/06
2/28/2022	2022/09
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/19
5/16/2022	2022/20
6/15/2022	2022/24
6/26/2022	2022/26
7/9/2022	2022/28
7/22/2022	2022/29
7/23/2022	2022/30
7/27/2022	2022/30
8/2/2022	2022/31
8/8/2022	2022/32
8/19/2022	2022/33
9/26/2022	2022/39
10/14/2022	2022/41
10/29/2022	2022/44

通过使用 `lunarweekname()` 函数并将 `lunar_week_name` 字段作为函数的参数传递，在前置 Load 语句中创建了 `date` 字段。

`lunarweekname()` 函数标识日期值属于哪个农历周，并返回该日期的年和周数字。

`lunarweekname()` 函数图表，示例没有额外参数



交易 8189 发生在 1 月 19 日。`lunarweekname()` 函数确定该日期属于 1 月 15 日开始的农历周；这是一年中的第三个农历周。因此，该交易返回的 `lunar_week_name` 值为 2022/03。

示例 2–带 period_no 参数的日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建字段 previous_lunar_week_name，返回交易发生前的农历周的年和周数字。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekname(date,-1) as previous_lunar_week_name
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

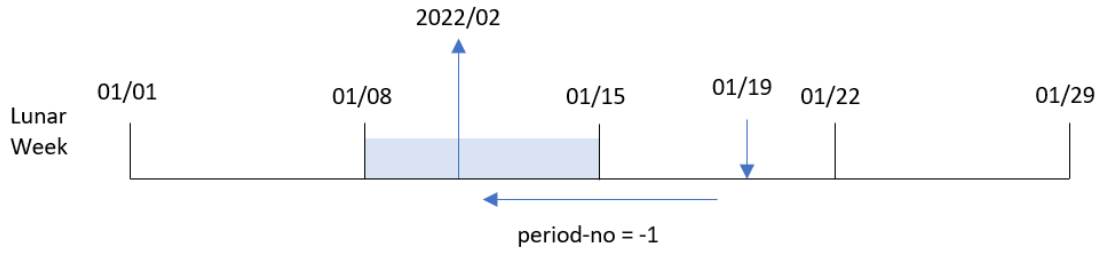
- date
- previous_lunar_week_name

结果表

日期	previous_lunar_week_name
1/7/2022	2021/52
1/19/2022	2022/02
2/5/2022	2022/05
2/28/2022	2022/08
3/16/2022	2022/10
4/1/2022	2022/12
5/7/2022	2022/18
5/16/2022	2022/19
6/15/2022	2022/23
6/26/2022	2022/25
7/9/2022	2022/27
7/22/2022	2022/28
7/23/2022	2022/29
7/27/2022	2022/29
8/2/2022	2022/30
8/8/2022	2022/31
8/19/2022	2022/32
9/26/2022	2022/38
10/14/2022	2022/40
10/29/2022	2022/43

在本例中，由于 lunarweekname() 函数中使用了为 -1 的 period_no 作为偏移量参数，因此该函数首先标识交易发生的农历周。然后返回年份和前一周的数字。

lunarweekname() 函数的图表，period_no 示例



交易 8189 发生在 1 月 19 日。lunarweekname() 函数确定该交易发生在一年中的第三个农历周，因此它返回 previous_lunar_week_name 字段在 2022/02 之前一周的年份和值。

示例 3 – 带有 first_week_day 参数的日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在这个例子中，我们将农历周设置为 1 月 6 日开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekname(date,0,4) as lunar_week_name
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
```

```
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

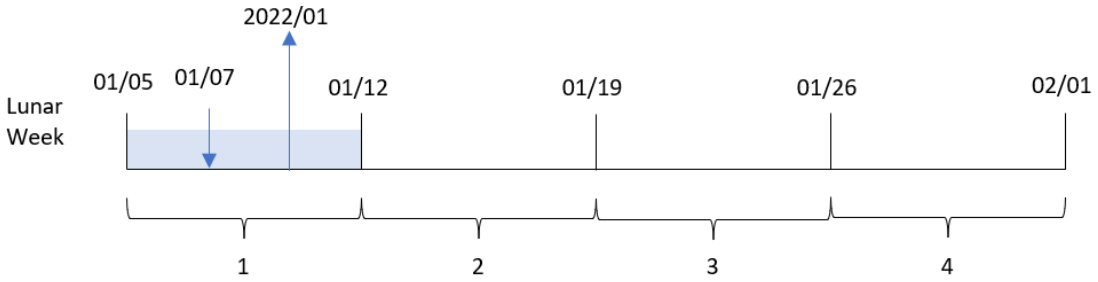
加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- lunar_week_name

结果表

日期	lunar_week_name
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/05
2/28/2022	2022/08
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/18
5/16/2022	2022/19
6/15/2022	2022/24
6/26/2022	2022/25
7/9/2022	2022/27
7/22/2022	2022/29
7/23/2022	2022/29
7/27/2022	2022/30
8/2/2022	2022/30
8/8/2022	2022/31
8/19/2022	2022/33
9/26/2022	2022/38
10/14/2022	2022/41
10/29/2022	2022/43

lunarweekname() 函数的图表，first_week_day 示例



在本例中，由于 lunarweekname() 函数中使用了参数 first_week_date，因此它将农历周初从 1 月 1 日偏移到 1 月 5 日。

交易 8188 发生在 1 月 7 日。由于农历周从 1 月 5 日开始，lunarweekname() 函数确定包含 1 月 7 号的农历周是一年中的第一个农历周。因此，该交易返回的 lunar_week_name 值为 2022/01。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生的农历周数和年份的计算在应用程序的图表对象中创建为度量。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
```

```
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

要计算交易发生的农历周的开始日期，请创建以下度量：

```
=lunarweekname(date)
```

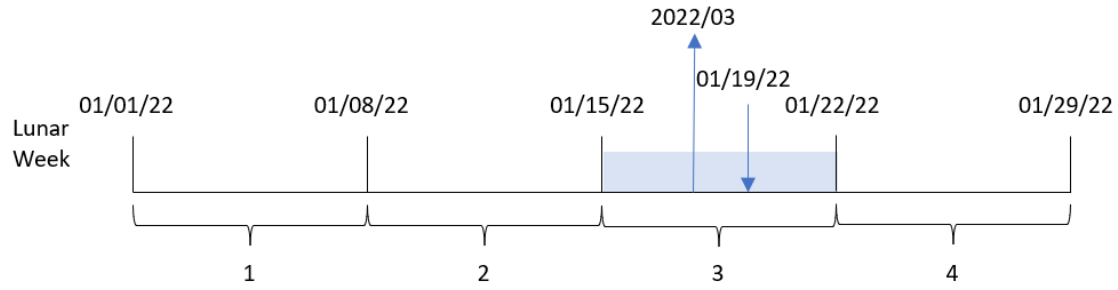
结果表

日期	=lunarweekname(date)
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/06
2/28/2022	2022/09
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/19
5/16/2022	2022/20
6/15/2022	2022/24
6/26/2022	2022/26
7/9/2022	2022/28
7/22/2022	2022/29
7/23/2022	2022/30
7/27/2022	2022/30
8/2/2022	2022/31
8/8/2022	2022/32
8/19/2022	2022/33
9/26/2022	2022/39
10/14/2022	2022/41
10/29/2022	2022/44

通过使用 lunarweekname() 函数并将 date 字段作为函数的参数传递，在图表对象中创建 lunar_week_name 度量。

lunarweekname() 函数标识日期值属于哪个农历周，并返回该日期的年和周数字。

lunarweekname() 函数的图表，图表对象示例



交易 8189 发生在 1 月 19 日。lunarweekname() 函数确定该日期属于 1 月 15 日开始的农历周；这是一年中的第三个农历周。因此，该交易的 lunar_week_name 值为 2022/03。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

最终用户需要一个图表对象，该对象按周显示当前年度的总销售额。第 1 周，长度为 7 天，应该从 1 月 1 日开始。即使数据模型中没有此维度，也可以通过将 lunarweekname() 函数用作图表中的计算维度来实现。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
```

```
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 使用以下表达式创建计算维度：
   `=lunarweekname(date)`
3. 使用以下聚合度量计算总销售额：
   `=sum(amount)`
4. 将度量的**数字格式**设置为**金额**。

结果表

=lunarweekname(date)	=sum(amount)
2022/01	$17.17
2022/03	$37.23
2022/06	$57.42
2022/09	$88.27
2022/11	$53.80
2022/13	$82.06
2022/19	$40.39
2022/20	$87.21
2022/24	$95.93
2022/26	$45.89
2022/28	$36.23
2022/29	$25.66

=lunarweekname(date)	=sum(amount)
2022/30	$152.75
2022/31	$76.11
2022/32	$25.12
2022/33	$46.23
2022/39	$84.21
2022/41	$96.24
2022/44	$67.67

lunarweekstart

此函数用于返回与包含 **date** 的阴历周第一天的第一毫秒的时间戳对应的值。Qlik Sense 中的农历周定义为将 1 月 1 日计算为一周的第一天，除一年的最后一周外，将正好包含七天。

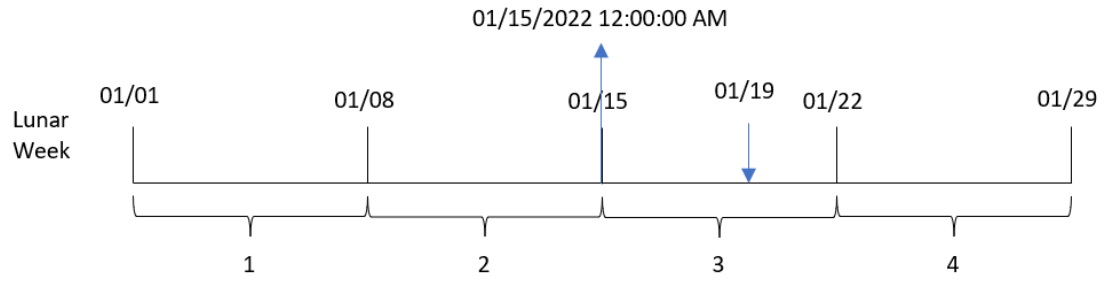
语法：

```
LunarweekStart(date[, period_no[, first_week_day]])
```

返回数据类型： 双

lunarweekstart() 函数决定 date 属于哪个农历周。然后以日期格式返回该周第一毫秒的时间戳。

lunarweekstart() 函数的示例图表



参数

参数	说明
date	要评估的日期或时间戳。
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。

适用场景

当用户希望计算使用到目前为止已过的一周的部分时，lunarweekstart() 函数通常用作表达式的一部分。与 weekstart() 函数不同，在每个新日历年开始时，一周从 1 月 1 日开始，随后的每一周都在 7 天后开始。lunarweekstart() 函数不受 FirstWeekDay 系统变量的影响。

例如，lunarweekstart() 可用于计算一周内累计的利息。

函数示例

示例	结果
lunarweekstart('01/12/2013')	返回 01/08/2013。
lunarweekstart('01/12/2013', -1)	返回 01/01/2013。
lunarweekstart('01/12/2013', 0, 1)	返回 01/09/2013，因为设置 first_week_day 为 1 表示年初已更改为 01/02/2013。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建字段 start_of_week，返回交易发生的农历周开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
```

```
    Load
        *,
        lunarweekstart(date) as start_of_week,
        timestamp(lunarweekstart(date)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- start_of_week
- start_of_week_timestamp

结果表

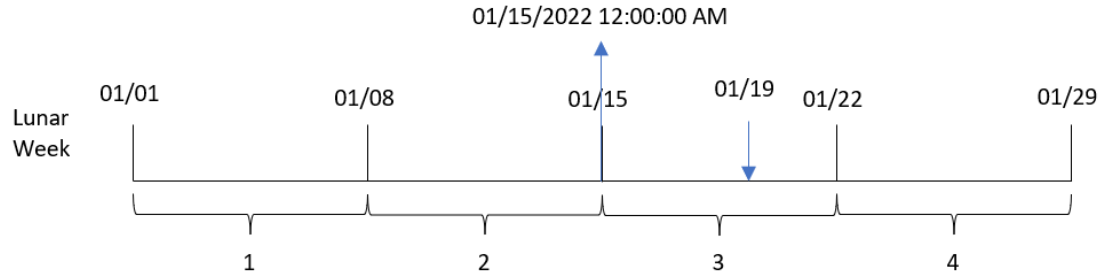
日期	start_of_week	start_of_week_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/15/2022	1/15/2022 12:00:00 AM
2/5/2022	02/05/2022	2/5/2022 12:00:00 AM
2/28/2022	02/26/2022	2/26/2022 12:00:00 AM
3/16/2022	03/12/2022	3/12/2022 12:00:00 AM

日期	start_of_week	start_of_week_timestamp
4/1/2022	03/26/2022	3/26/2022 12:00:00 AM
5/7/2022	05/07/2022	5/7/2022 12:00:00 AM
5/16/2022	05/14/2022	5/14/2022 12:00:00 AM
6/15/2022	06/11/2022	6/11/2022 12:00:00 AM
6/26/2022	06/25/2022	6/25/2022 12:00:00 AM
7/9/2022	07/09/2022	7/9/2022 12:00:00 AM
7/22/2022	07/16/2022	7/16/2022 12:00:00 AM
7/23/2022	07/23/2022	7/23/2022 12:00:00 AM
7/27/2022	07/23/2022	7/23/2022 12:00:00 AM
8/2/2022	07/30/2022	7/30/2022 12:00:00 AM
8/8/2022	08/06/2022	8/6/2022 12:00:00 AM
8/19/2022	08/13/2022	8/13/2022 12:00:00 AM
9/26/2022	09/24/2022	9/24/2022 12:00:00 AM
10/14/2022	10/08/2022	10/8/2022 12:00:00 AM
10/29/2022	10/29/2022	10/29/2022 12:00:00 AM

通过使用 lunarweekstart() 函数并将 `date` 字段作为函数的参数传递，在前置 Load 语句中创建了 `start_of_week` 字段。

`lunarweekstart()` 函数标识日期所属的农历周，并返回该周第一毫秒的时间戳。

`lunarweekstart()` 函数图表，示例没有额外参数



交易 8189 发生在 1 月 19 日。`lunarweekstart()` 函数确定农历周从 1 月 15 日开始。因此，该交易的 `start_of_week` 值返回当天的第一毫秒，即 1 月 15 日上午 12:00:00。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_lunar_week_start，该字段返回事务发生前的农历周开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekstart(date,-1) as previous_lunar_week_start,
        timestamp(lunarweekstart(date,-1)) as previous_lunar_week_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

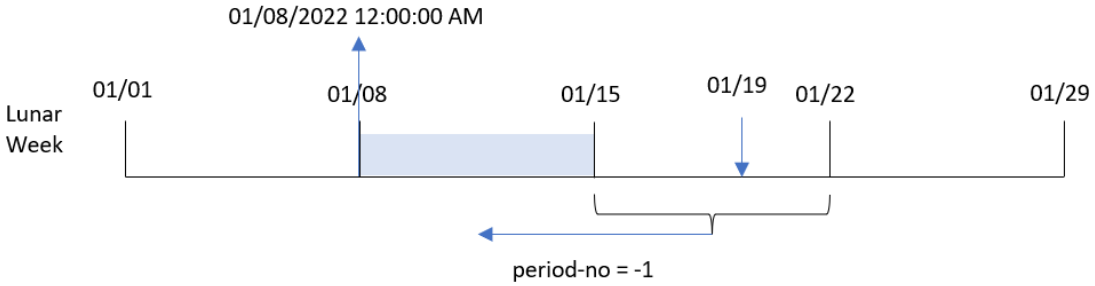
结果

结果表

日期	previous_lunar_week_start	previous_lunar_week_start_timestamp
1/7/2022	12/24/2021	12/24/2021 12:00:00 AM
1/19/2022	01/08/2022	1/8/2022 12:00:00 AM
2/5/2022	01/29/2022	1/29/2022 12:00:00 AM
2/28/2022	02/19/2022	2/19/2022 12:00:00 AM
3/16/2022	03/05/2022	3/5/2022 12:00:00 AM
4/1/2022	03/19/2022	3/19/2022 12:00:00 AM
5/7/2022	04/30/2022	4/30/2022 12:00:00 AM
5/16/2022	05/07/2022	5/7/2022 12:00:00 AM
6/15/2022	06/04/2022	6/4/2022 12:00:00 AM
6/26/2022	06/18/2022	6/18/2022 12:00:00 AM
7/9/2022	07/02/2022	7/2/2022 12:00:00 AM
7/22/2022	07/09/2022	7/9/2022 12:00:00 AM
7/23/2022	07/16/2022	7/16/2022 12:00:00 AM
7/27/2022	07/16/2022	7/16/2022 12:00:00 AM
8/2/2022	07/23/2022	7/23/2022 12:00:00 AM
8/8/2022	07/30/2022	7/30/2022 12:00:00 AM
8/19/2022	08/06/2022	8/6/2022 12:00:00 AM
9/26/2022	09/17/2022	9/17/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/22/2022	10/22/2022 12:00:00 AM

在本例中，由于 `lunarweekstart()` 函数中使用了为 -1 的 `period_no` 作为偏移参数，因此函数首先标识交易发生的农历周。然后，它在一周前移动，并确定农历周的第一毫秒。

lunarweekstart() 函数的图表，period_no 示例



交易 8189 发生在 1 月 19 日。lunarweekstart() 函数确定农历周从 1 月 15 日开始。因此，上一个农历周从 1 月 8 日上午 12:00:00 开始；这是为 previous_lunar_week_start 字段返回的值。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在这个例子中，我们将农历周设置为 1 月 6 日开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        lunarweekstart(date,0,4) as start_of_week,
        timestamp(lunarweekstart(date,0,4)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
```

```
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- start_of_week
- start_of_week_timestamp

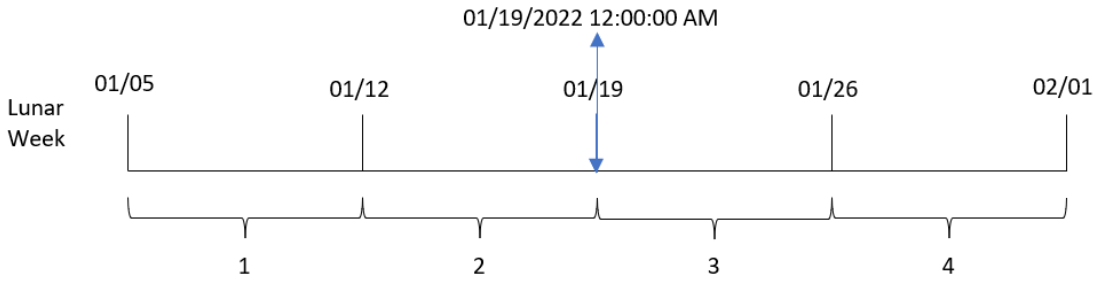
结果表

日期	start_of_week	start_of_week_timestamp
1/7/2022	01/05/2022	1/5/2022 12:00:00 AM
1/19/2022	01/19/2022	1/19/2022 12:00:00 AM
2/5/2022	02/02/2022	2/2/2022 12:00:00 AM
2/28/2022	02/23/2022	2/23/2022 12:00:00 AM
3/16/2022	03/16/2022	3/16/2022 12:00:00 AM
4/1/2022	03/30/2022	3/30/2022 12:00:00 AM
5/7/2022	05/04/2022	5/4/2022 12:00:00 AM
5/16/2022	05/11/2022	5/11/2022 12:00:00 AM
6/15/2022	06/15/2022	6/15/2022 12:00:00 AM
6/26/2022	06/22/2022	6/22/2022 12:00:00 AM
7/9/2022	07/06/2022	7/6/2022 12:00:00 AM
7/22/2022	07/20/2022	7/20/2022 12:00:00 AM
7/23/2022	07/20/2022	7/20/2022 12:00:00 AM
7/27/2022	07/27/2022	7/27/2022 12:00:00 AM
8/2/2022	07/27/2022	7/27/2022 12:00:00 AM
8/8/2022	08/03/2022	8/3/2022 12:00:00 AM
8/19/2022	08/17/2022	8/17/2022 12:00:00 AM
9/26/2022	09/21/2022	9/21/2022 12:00:00 AM

日期	start_of_week	start_of_week_timestamp
10/14/2022	10/12/2022	10/12/2022 12:00:00 AM
10/29/2022	10/26/2022	10/26/2022 12:00:00 AM

在本例中，由于 lunarweekstart() 函数中使用了参数 first_week_date，因此它将年初从 1 月 1 日偏移到 1 月 5 日。

lunarweekstart() 函数的图表，first_week_day 示例



交易 8189 发生在 1 月 19 日。由于农历周从 1 月 5 日开始，lunarweekstart() 函数确定包含 1 月 19 日的农历周也从 1 月 19 号上午 12:00:00 开始。因此，这是 start_of_week 字段返回的值。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生的农历周开始时间戳的计算在应用程序的图表对象中创建为度量。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
```

```
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

添加以下度量：

```
=lunarweekstart(date)
```

```
=timestamp(lunarweekstart(date))
```

结果表

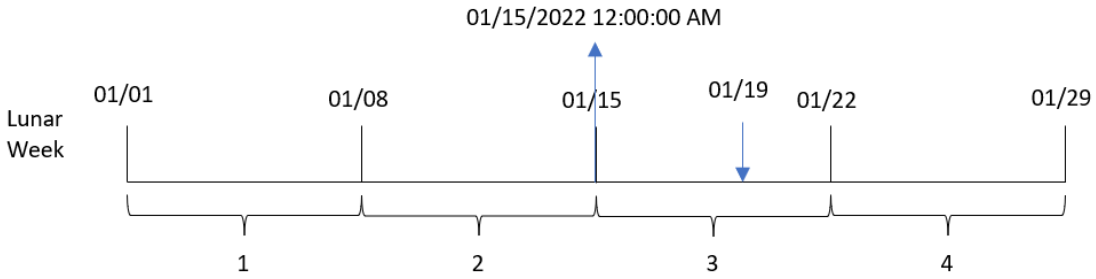
日期	=lunarweekstart(date)	=timestamp(lunarweekstart(date))
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/15/2022	1/15/2022 12:00:00 AM
2/5/2022	02/05/2022	2/5/2022 12:00:00 AM
2/28/2022	02/26/2022	2/26/2022 12:00:00 AM
3/16/2022	03/12/2022	3/12/2022 12:00:00 AM
4/1/2022	03/26/2022	3/26/2022 12:00:00 AM
5/7/2022	05/07/2022	5/7/2022 12:00:00 AM
5/16/2022	05/14/2022	5/14/2022 12:00:00 AM
6/15/2022	06/11/2022	6/11/2022 12:00:00 AM
6/26/2022	06/25/2022	6/25/2022 12:00:00 AM
7/9/2022	07/09/2022	7/9/2022 12:00:00 AM
7/22/2022	07/16/2022	7/16/2022 12:00:00 AM
7/23/2022	07/23/2022	7/23/2022 12:00:00 AM
7/27/2022	07/23/2022	7/23/2022 12:00:00 AM
8/2/2022	07/30/2022	7/30/2022 12:00:00 AM

日期	=lunarweekstart(date)	=timestamp(lunarweekstart(date))
8/8/2022	08/06/2022	8/6/2022 12:00:00 AM
8/19/2022	08/13/2022	8/13/2022 12:00:00 AM
9/26/2022	09/24/2022	9/24/2022 12:00:00 AM
10/14/2022	10/08/2022	10/8/2022 12:00:00 AM
10/29/2022	10/29/2022	10/29/2022 12:00:00 AM

通过使用 `lunarweekstart()`函数并将日期字段作为函数的参数传递，在图表对象中创建了 `start_of_week` 度量。

`lunarweekstart()` 函数标识日期值属于哪个农历周，并返回该周最后一毫秒的时间戳。

`lunarweekstart()` 函数的图表，图表对象示例



交易 8189 发生在 1 月 19 日。`lunarweekstart()` 函数确定农历周从 1 月 15 日开始。因此，该交易的 `start_of_week` 值是当天的第一毫秒，即 1 月 15 日上午 12:00:00。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含一组贷款余额的数据集，该数据集加载到名为 `Loans` 的表中。
- 数据包括贷款 ID、本周初余额和每年每笔贷款收取的简单利率。

最终用户希望有一个图表对象，该对象按贷款 ID 显示周初至今每笔贷款的应计利息。

加载脚本

```
Loans:
Load
*
Inline
```

```
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 添加以下字段作为维度：
   - loan_id
   - start_balance
3. 接下来，创建以下度量来计算累计利息：
   `=start_balance*(rate*(today(1)-lunarweekstart(today(1)))/365)`
4. 将度量的**数字格式**设置为**金额**。

结果表

loan_id	start_balance	=start_balance*(rate*(today(1)-lunarweekstart (today(1)))/365)
8188	$10000.00	$15.07
8189	$15000.00	$128.84
8190	$17500.00	$63.29
8191	$21000.00	$107.59
8192	$90000.00	$1139.18

`lunarweekstart()` 函数使用今天的日期作为唯一参数，返回当前年份的开始日期。通过从当前日期中减去该结果，表达式将返回本周迄今为止经过的天数。

然后将该值乘以利率并除以 365，以返回该期间产生的实际利率。然后将结果乘以贷款的起始余额，以返回本周迄今为止累计的利息。

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	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)。

适用场景

makedate() 函数通常在脚本中用于生成数据以生成日历。当日期字段不能直接用作日期，但需要进行一些转换以提取年、月和日组件时，也可以使用此方法。

以下示例使用日期格式 MM/DD/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。

函数示例

示例	结果
makedate(2012)	返回 01/01/2012。
makedate(12)	返回 01/01/2012。
makedate(2012,12)	返回 12/01/2012。
makedate(2012,2,14)	返回 02/14/2012。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2018 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 transaction_date，返回格式为 MM/DD/YYYY 的日期。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

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, customer_id
3750, 2018, 08, 30, 12423.56, 23, 2038593
3751, 2018, 09, 07, 5356.31, 6, 203521
3752, 2018, 09, 16, 15.75, 1, 5646471
3753, 2018, 09, 22, 1251, 7, 3036491
3754, 2018, 09, 22, 21484.21, 1356, 049681
3756, 2018, 09, 22, -59.18, 2, 2038593
3757, 2018, 09, 23, 3177.4, 21, 203521
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- transaction_year
- transaction_month
- transaction_day
- transaction_date

结果表

transaction_year	transaction_month	transaction_day	transaction_date
2018	08	30	08/30/2018
2018	09	07	09/07/2018
2018	09	16	09/16/2018
2018	09	22	09/22/2018
2018	09	23	09/23/2018

transaction_date 字段是在前置 Load 语句中创建的，方法是使用 makedate() 函数并将年、月、日字段作为函数参数传递。

然后，该函数将这些值组合并转换为日期字段，以 DateFormat 系统变量的格式返回结果。

示例 2 – 修改的日期格式

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 在不修改 DateFormat 系统变量的情况下，以 DD/MM/YYYY 格式创建字段 transaction_date。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        date(makedate(transaction_year, transaction_month, transaction_day), 'DD/MM/YYYY') as
transaction_date
        ;
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_day, transaction_amount,
transaction_quantity, customer_id
3750, 2018, 08, 30, 12423.56, 23, 2038593
3751, 2018, 09, 07, 5356.31, 6, 203521
3752, 2018, 09, 16, 15.75, 1, 5646471
3753, 2018, 09, 22, 1251, 7, 3036491
3754, 2018, 09, 22, 21484.21, 1356, 049681
3756, 2018, 09, 22, -59.18, 2, 2038593
3757, 2018, 09, 23, 3177.4, 21, 203521
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- transaction_year
- transaction_month
- transaction_day
- transaction_date

结果表

transaction_year	transaction_month	transaction_day	transaction_date
2018	08	30	30/08/2018
2018	09	07	07/09/2018

transaction_year	transaction_month	transaction_day	transaction_date
2018	09	16	16/09/2018
2018	09	22	22/09/2018
2018	09	23	23/09/2018

在本例中，makedate() 函数嵌套在 date() 函数内部。date() 函数的第二个参数将 makedate() 函数结果的格式设置为所需的 DD/MM/YYYY。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2018 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 跨以下两个字段提供的交易日期：year 和 month。

创建一个图表对象度量 transaction_date，返回格式为 MM/DD/YYYY 的日期。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_amount, transaction_quantity,
customer_id
3750, 2018, 08, 12423.56, 23, 2038593
3751, 2018, 09, 5356.31, 6, 203521
3752, 2018, 09, 15.75, 1, 5646471
3753, 2018, 09, 1251, 7, 3036491
3754, 2018, 09, 21484.21, 1356, 049681
3756, 2018, 09, -59.18, 2, 2038593
3757, 2018, 09, 3177.4, 21, 203521
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- year
- month

要确定 transaction_date，创建该度量：

```
=makedate(transaction_year,transaction_month)
```

结果表

transaction_year	transaction_month	transaction_date
2018	08	08/01/2018
2018	09	09/01/2018

通过使用 makedate() 函数并将年份和月份字段作为函数的参数传递，在图表对象中创建 transaction_date 度量。

然后，该函数将这些值与假定的日值 01 进行组合。然后将这些值转换为日期字段，以 DateFormat 系统变量的格式返回结果。

示例 4 – 场景

加载脚本和图表表达式

概述

创建 2022 日历年的日历数据集。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Calendar:
       load
               *
    where year(date)=2022;
load
       date(recno()+makedate(2021,12,31)) as date
AutoGenerate 400;
```

结果

结果表

日期
01/01/2022
01/02/2022
01/03/2022
01/04/2022
01/05/2022
01/06/2022
01/07/2022

日期
01/08/2022
01/09/2022
01/10/2022
01/11/2022
01/12/2022
01/13/2022
01/14/2022
01/15/2022
01/16/2022
01/17/2022
01/18/2022
01/19/2022
01/20/2022
01/21/2022
01/22/2022
01/23/2022
01/24/2022
01/25/2022
+ 340 更多行

makedate() 函数为 2021 年 12 月 31 日创建日期值recno() 函数提供加载到表中的当前记录的记录号，从 1 开始。因此，第一条记录的日期为 2022 年 1 月 1 日。然后，每个连续的 recno() 都将这个日期递增 1。这个表达式被包装在一个 date() 函数中，以将值转换为日期。此过程由 autogenerate 函数重复 400 次。最后，通过使用前置 Load，where 条件可以用于仅加载 2022 年的日期。此脚本生成包含 2022 年的每个日期的日历。

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	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。

适用场景

maketime() 函数通常在脚本中用于生成数据以生成时间字段。有时，当时间字段是从输入文本派生的时，可以使用此函数使用其组件来构造时间。

以下示例使用时间格式 h:mm:ss。时间格式已经在数据加载脚本顶部的 SET TimeFormat 语句中指定。可以根据要求更改示例中的格式。

函数示例

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。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – maketime()

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 跨三个字段提供的交易时间：hours、minutes 和 seconds。
- 创建字段 transaction_time，以系统变量 TimeFormat 的格式返回时间。

加载脚本

```
SET TimeFormat='h:mm:ss TT';

Transactions:
    Load
        *,
        maketime(transaction_hour, transaction_minute, transaction_second) as transaction_time
        ;
Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_second, transaction_amount,
transaction_quantity, customer_id
3750, 18, 43, 30, 12423.56, 23, 2038593
3751, 6, 32, 07, 5356.31, 6, 203521
3752, 12, 09, 16, 15.75, 1, 5646471
3753, 21, 43, 41, 7, 3036491
3754, 17, 55, 22, 21484.21, 1356, 049681
3756, 2, 52, 22, -59.18, 2, 2038593
3757, 9, 25, 23, 3177.4, 21, 203521
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- transaction_hour
- transaction_minute
- transaction_second
- transaction_time

结果表

transaction_hour	transaction_minute	transaction_second	transaction_time
2	52	22	2:52:22 AM

transaction_hour	transaction_minute	transaction_second	transaction_time
6	32	07	6:32:07 AM
9	25	23	9:25:23 AM
12	09	16	12:09:16 PM
17	55	22	5:55:22 PM
18	43	30	6:43:30 PM
21	43	41	9:43:41 PM

transaction_time 字段是在前置 Load 语句中创建的，方法是使用 maketime() 函数并将小时、分钟和秒字段作为函数参数传递。

然后，该函数将这些值组合并转换为时间字段，以 TimeFormat 系统变量的时间格式返回结果。

示例 2 – time() 函数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 transaction_time，这将允许我们在不修改 TimeFormat 系统变量的情况下以 24 小时时间格式显示结果。

加载脚本

```
SET TimeFormat='h:mm:ss TT';

Transactions:
    Load
        *,
        time(maketime(transaction_hour, transaction_minute, transaction_second),'h:mm:ss') as
transaction_time
        ;
Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_second, transaction_amount,
transaction_quantity, customer_id
3750, 18, 43, 30, 12423.56, 23, 2038593
3751, 6, 32, 07, 5356.31, 6, 203521
3752, 12, 09, 16, 15.75, 1, 5646471
3753, 21, 43, 41, 7, 3036491
3754, 17, 55, 22, 21484.21, 1356, 049681
3756, 2, 52, 22, -59.18, 2, 2038593
3757, 9, 25, 23, 3177.4, 21, 203521
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- transaction_hour
- transaction_minute
- transaction_second
- transaction_time

结果表

transaction_hour	transaction_minute	transaction_second	transaction_time
2	52	22	2:52:22
6	32	07	6:32:07
9	25	23	9:25:23
12	09	16	12:09:16
17	55	22	17:55:22
18	43	30	18:43:30
21	43	41	21:43:41

在本例中，maketime() 函数嵌套在 time() 函数内部。time() 函数的第二个参数将 maketime() 函数结果的格式设置为所需的 h:mm:ss。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 跨两个字段提供的交易时间：hours 和 minutes。
- 创建字段 transaction_time，以系统变量 TimeFormat 的格式返回时间。

创建一个图表对象度量 transaction_time，返回格式为 h:mm:ss TT 的日期。

加载脚本

```
SET TimeFormat='h:mm:ss TT';

Transactions:
Load * Inline [
```

```
transaction_id, transaction_hour, transaction_minute, transaction_amount, transaction_
quantity, customer_id
3750, 18, 43, 12423.56, 23, 2038593
3751, 6, 32, 5356.31, 6, 203521
3752, 12, 09, 15.75, 1, 5646471
3753, 21, 43, 7, 3036491
3754, 17, 55, 21484.21, 1356, 049681
3756, 2, 52, -59.18, 2, 2038593
3757, 9, 25, 3177.4, 21, 203521
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- transaction_hour
- transaction_minute

要计算 transaction_time，请创建此度量：

```
=maketime(transaction_hour,transaction_minute)
```

结果表

transaction_hour	transaction_minute	=maketime(transaction_hour, transaction_minute)
2	52	2:52:00 AM
6	32	6:32:00 AM
9	25	9:25:00 AM
12	09	12:09:00 PM
17	55	5:55:00 PM
18	43	6:43:00 PM
21	43	9:43:00 PM

通过使用 maketime() 函数并将小时和分钟字段作为函数参数传递，在图表对象中创建 transaction_time 度量。

然后，函数将这些值组合起来，并假定秒为 00。然后将这些值转换为时间字段，以 TimeFormat 系统变量的格式返回结果。

示例 4 – 场景

加载脚本和图表表达式

概述

创建 2022 年 1 月的日历数据集，分为 8 小时增量。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

tmpCalendar:
       load
                *
    where year(date)=2022;
load
       date(recno()+makedate(2021,12,31)) as date
AutoGenerate 31;

Left join(tmpCalendar)
load
       maketime((recno()-1)*8,00,00) as time
autogenerate 3;

Calendar:
load
       timestamp(date + time) as timestamp
resident tmpCalendar;

drop table tmpCalendar;
```

结果

结果表

时间戳
1/1/2022 12:00:00 AM
1/1/2022 8:00:00 AM
1/1/2022 4:00:00 PM
1/2/2022 12:00:00 AM
1/2/2022 8:00:00 AM
1/2/2022 4:00:00 PM
1/3/2022 12:00:00 AM
1/3/2022 8:00:00 AM
1/3/2022 4:00:00 PM
1/4/2022 12:00:00 AM
1/4/2022 8:00:00 AM
1/4/2022 4:00:00 PM
1/5/2022 12:00:00 AM
1/5/2022 8:00:00 AM

时间戳
1/5/2022 4:00:00 PM
1/6/2022 12:00:00 AM
1/6/2022 8:00:00 AM
1/6/2022 4:00:00 PM
1/7/2022 12:00:00 AM
1/7/2022 8:00:00 AM
1/7/2022 4:00:00 PM
1/8/2022 12:00:00 AM
1/8/2022 8:00:00 AM
1/8/2022 4:00:00 PM
1/9/2022 12:00:00 AM
+ 68 更多行

初始 autogenerate 函数在名为 tmpCalendar 的表中创建一个日历，其中包含 1 月份的所有日期。

创建了包含三条记录的第二个表。对于每个记录，recno() - 1 取（值 0、1、2），并将结果乘以 8。因此，生成值 0、8 16。这些值用作 maketime() 函数中的小时参数，分钟和秒值为 0。因此，该表包含三个时间字段：12:00:00 AM、8:00:00 AM 和 4:00:00 PM。

这张表与 tmpCalendar 表相联接。因为联接的两个表之间没有匹配的字段，所以时间行被添加到每个日期行。因此，每个日期行现在用每个时间值重复三次。

最后，日历表是从 tmpCalendar 表的常驻负载创建的。日期和时间字段在 timestamp() 函数中连接和包装，以创建时间戳字段。

然后将 tmpCalendar 表放下。

makeweekdate

此函数返回根据年、周数和星期几计算的日期。

语法：

```
MakeWeekDate(weekyear [, week [, weekday [, first_week_day [, broken_weeks [, reference_day]]]]])
```

返回数据类型：双

makeweekdate() 函数可作为脚本和图表函数使用。函数将根据传递到函数中的参数计算日期。

参数

参数	说明
weekyear	WeekYear() 函数为特定日期定义的年份，即周数所属的年份。 *在某些情况下，周年份可能与日历年不同，例如，如果第 1 周已经在上一年的 12 月开始。*
week	Week() 函数为特定日期定义的周数。 如果未注明周数，则假定为 1。
weekday	WeekDay() 函数为相关日期定义的星期几。0 是一周中的第一天，6 是一周的最后一天。 如果未指定星期几，则假定为 0。 *尽管 0 总是表示一周的第一天，6 总是最后一天，但对应的工作日由* ***first_week_day*** *参数确定。如果忽略，使用* ***FirstWeekDay*** *变量的值。* 如果使用中断周，以及不可能的参数组合，这可能会导致不属于所选年份的结果。 **示例：** MakeWeekDate(2021,1,0,6,1) 返回 'Dec 27 2020'，因为这一天是指定周的第一天（星期日）。2021 年 1 月 1 日是星期五。
first_week_day	指定一周的开始日期。如果忽略，使用 **FirstWeekDay** 变量的值。 可能的值 **first_week_day** 为：周一为 0，周二为 1，周三为 2，周四为3，周五为 4，周六为 5，星期日为 6。 有关系统变量的详细信息，请参见*FirstWeekDay (page 218)*。
broken_weeks	如果不指定 **broken_weeks**，则变量 **BrokenWeeks** 的值将用于定义周是否中断。
reference_day	如果不指定 **reference_day**，则变量 **ReferenceDay** 的值用于定义将一月的哪一天设置为定义第 1 周的参考日。

适用场景

makeweekdate() 函数通常用于脚本中的数据生成，以生成日期列表，或在输入数据中提供年、周和日时构造日期。

以下示例假设：

```
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;
```

函数示例

示例	结果
makeweekdate(2014,6,6)	返回 02/09/2014
makeweekdate(2014,6,1)	返回 02/04/2014
makeweekdate(2014,6)	返回 02/03/2014(假定普通日为 0)

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 包括日

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 在名为 Sales 的表中包含 2022 年每周销售总额的数据集。
- 跨三个字段提供的交易日期：year、week 和 sales。
- 前置 Load，用于创建度量 end_of_week，使用 makeweekdate() 函数以 MM/DD/YYYY 格式返回该周星期五的日期。

为了证明返回的日期是星期五，end_of_week 表达式也被包装在 weekday() 函数中以显示星期几。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Transactions:
    Load
        *,
        makeweekdate(transaction_year, transaction_week,4) as end_of_week,
```

```
        weekday(makeweekdate(transaction_year, transaction_week,4)) as week_day
        ;
Load * Inline [
transaction_year, transaction_week, sales
2022, 01, 10000
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- transaction_year
- transaction_week
- end_of_week
- week_day

结果表

transaction_year	transaction_week	end_of_week	week_day
2022	01	01/07/2022	Fri
2022	02	01/14/2022	Fri
2022	03	01/21/2022	Fri
2022	04	01/28/2022	Fri
2022	05	02/04/2022	Fri
2022	06	02/11/2022	Fri
2022	07	02/18/2022	Fri

end_of_week 字段是在前置 Load 语句中使用 makeweekdate() 函数创建的。transaction_year、transaction_week 字段作为年和周参数通过函数传递。值 4 用于日参数。

然后,该函数将这些值组合并转换为日期字段,以 DateFormat 系统变量的格式返回结果。

makeweekdate() 函数及其参数也封装在一个 weekday() 函数中以返回 week_day 字段;如上表所示,week_day 字段显示这些日期确实发生在星期五。

示例 2 – 排除日

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 在名为 Sales 的表中包含 2022 年每周销售总额的数据集。
- 跨三个字段提供的交易日期：year、week 和 sales。
- 前置 Load，用于使用 makeweekdate() 函数创建度量 first_day_of_week。这将返回星期一的日期，格式为 MM/DD/YYYY。

为了证明返回的日期是星期一，first_day_of_week 表达式也被包装在 weekday() 函数中以显示星期几。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Transactions:
    Load
        *,
        makeweekdate(transaction_year, transaction_week) as first_day_of_week,
        weekday(makeweekdate(transaction_year, transaction_week)) as week_day
        ;
Load * Inline [
transaction_year, transaction_week, sales
2022, 01, 10000
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- transaction_year
- transaction_week
- first_day_of_week
- week_day

结果表

transaction_year	transaction_week	first_day_of_week	week_day
2022	01	01/03/2022	Mon
2022	02	01/10/2022	Mon
2022	03	01/17/2022	Mon
2022	04	01/24/2022	Mon
2022	05	01/31/2022	Mon
2022	06	02/07/2022	Mon
2022	07	02/14/2022	Mon

first_day_of_week 字段是在前置 Load 语句中使用 makeweekdate() 函数创建的。transaction_year 和 transaction_week 参数作为函数参数传递，day 参数为空。

然后，该函数将这些值组合并转换为日期字段，以 DateFormat 系统变量的格式返回结果。

makeweekdate() 函数及其参数也封装在 weekday() 函数中以返回 week_day 字段。从上表中可以看出，week_day 字段在所有情况下都返回星期一，因为该参数在 makeweekdate() 函数中为空，默认为 0(一周的第一天)，并且 FirstWeekDay 系统变量将一周的第 1 天设置为星期一。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 在名为 Sales 的表中包含 2022 年每周销售总额的数据集。
- 跨三个字段提供的交易日期：year、week 和 sales。

在本例中，将使用图表对象创建与第一个示例中的 end_of_week 计算等效的度量。此度量值将使用 makeweekdate() 函数以 MM/DD/YYYY 格式返回该星期五的日期。

为了证明返回的日期是星期五，创建了第二个度量来返回星期几。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Master_Calendar:
Load * Inline [
transaction_year, transaction_week, sales
```

```
2022, 01, 10000
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将这些字段添加为维度：
   - transaction_year
   - transaction_week
2. 要执行与第一个示例中的 end_of_week 字段等效的计算，请创建以下度量：
   `=makeweekdate(transaction_year,transaction_week,4)`
3. 要计算每个事务的星期几，请创建以下度量：
   `=weekday(makeweekdate(transaction_year,transaction_week,4))`

结果表

transaction_year	transaction_week	=makeweekdate (transaction_year,transaction_week,4)	=weekday(makeweekdate (transaction_year,transaction_week,4))
2022	01	01/07/2022	Fri
2022	02	01/14/2022	Fri
2022	03	01/21/2022	Fri
2022	04	01/28/2022	Fri
2022	05	02/04/2022	Fri
2022	06	02/11/2022	Fri
2022	07	02/18/2022	Fri

使用 makeweekdate() 函数在图表对象中创建与 end_of_week 的等效字段作为度量。transaction_year 和 transaction_week 字段作为年和周参数传递。值 4 用于日参数。

然后，该函数将这些值组合并转换为日期字段，以 DateFormat 系统变量的格式返回结果。

makeweekdate() 函数及其参数也封装在一个 weekday() 函数中，以返回与第一个示例中的 week_day 字段等效的计算。如上表所示，右侧最后一列显示这些日期确实发生在星期五。

示例 4 – 场景

加载脚本和图表表达式

概述

在本例中，创建一个包含 2022 年所有周五的日期列表。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Calendar:
    load
           *,
        weekday(date) as weekday
    where year(date)=2022;
load
    makeweekdate(2022,recno()-2,4) as date
AutoGenerate 60;
```

结果

结果表

日期	weekday
01/07/2022	Fri
01/14/2022	Fri
01/21/2022	Fri
01/28/2022	Fri
02/04/2022	Fri
02/11/2022	Fri
02/18/2022	Fri
02/25/2022	Fri
03/04/2022	Fri
03/11/2022	Fri
03/18/2022	Fri
03/25/2022	Fri

日期	weekday
04/01/2022	Fri
04/08/2022	Fri
04/15/2022	Fri
04/22/2022	Fri
04/29/2022	Fri
05/06/2022	Fri
05/13/2022	Fri
05/20/2022	Fri
05/27/2022	Fri
06/03/2022	Fri
06/10/2022	Fri
06/17/2022	Fri
+ 27 更多行	

makeweekdate() 函数查找 2022 年的每个星期五。使用为 -2 的 week 参数可确保不会错过任何日期。最后，为了清晰起见，前置 Load 创建了一个额外的 weekday 字段，以显示每个 date 值都是星期五。

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

适用场景

当您希望按分钟比较聚合时，minute() 函数非常有用。例如，如果希望按分钟查看活动计数分布，可以使用该函数。

通过使用函数在主日历表中创建字段，可以在 Load 脚本中创建这些维度。或者，它们可以直接在图表中用作计算维度。

函数示例

示例	结果
minute ('09:14:36')	返回 14。
minute ('0.5555')	返回 19(因为 0.5555 = 13:19:55)

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 变量(脚本)

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 按时间戳包含交易的数据集，加载到名为 Transactions 的表格中。
- 使用默认 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT)。
- 创建字段 minute，以计算交易发生的时间。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        minute(timestamp) as minute
        ;
Load
*
Inline
[
id,timestamp,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
```

```
9506,'2022-01-06 08:49:09',74.23
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- timestamp
- minute

结果表

时间戳	minute
2022-01-01 22:10:22	10
2022-01-02 08:35:54	35
2022-01-03 05:40:49	40
2022-01-03 14:21:53	21
2022-01-04 18:49:38	49
2022-01-04 22:58:34	58
2022-01-05 19:04:57	4
2022-01-05 19:34:46	34
2022-01-06 08:49:09	49
2022-01-06 11:29:38	29

minute 字段中的值是通过使用 minute() 函数并将 timestamp 作为前置 Load 语句中的表达式传递来创建的。

示例 2–图表对象(图表)

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 使用默认 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT)。

然而，在本例中，未更改的数据集被加载到应用程序中。minute 通过图表对象中的度量计算值。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';
```

```
Transactions:
Load
*
Inline
[
id,timestamp,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：timestamp。

创建以下度量：

```
=minute(timestamp)
```

结果表

时间戳	minute
2022-01-01 22:10:22	10
2022-01-02 08:35:54	35
2022-01-03 05:40:49	40
2022-01-03 14:21:53	21
2022-01-04 18:49:38	49
2022-01-04 22:58:34	58
2022-01-05 19:04:57	4
2022-01-05 19:34:46	34
2022-01-06 08:49:09	49
2022-01-06 11:29:38	29

minute 的值是通过使用 minute() 函数并将 timestamp 作为表达式传递给图表对象的度量来创建的。

示例 3 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 时间戳的数据集，生成该数据集以表示检票口处的条目。
- 每个 timestamp 的信息及其对应的 id，加载到名为 Ticket_Barrier_Tracker 的表中。
- 使用默认 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT)。

用户需要一个图表对象，以分钟为单位显示关口条目的计数。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

tmpTimeStampCreator:
       load
                *
    where year(date)=2022;
load
       date(recno()+makedate(2021,12,31)) as date
AutoGenerate 1;

join load
       maketime(floor(rand()*24),floor(rand()*59),floor(rand()*59)) as time
autogenerate 10000;

Ticket_Barrier_Tracker:
load
       recno() as id,
       timestamp(date + time) as timestamp
resident tmpTimeStampCreator;

drop table tmpTimeStampCreator;
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 使用以下表达式创建计算维度：
   `=minute(timestamp)`
3. 添加以下聚合度量以计算条目总数：
   `=count(id)`
4. 将度量的**数字格式**设置为**金额**。

结果表

minute(timestamp)	=count(id)
0	174
1	171
2	175
3	165
4	188
5	176
6	158
7	187
8	178
9	178
10	197
11	161
12	166
13	184
14	159
15	161
16	152
17	160
18	176
19	164
20	170
21	170
22	142
23	145
24	155
+ 35 更多行	

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

示例 1 – DateFormat 数据集 (脚本)

加载脚本和结果

概述

打开 数据加载编辑器 , 并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名称为 Master_Calendar 的日期数据集。DateFormat 系统变量设置为 DD/MM/YYYY。
- 使用 month() 函数创建另一个名为 month_name 的字段的前置 Load。
- 另一个名为 long_date 的字段 , 使用 date() 功能表示完整的日期。

加载脚本

```
SET DateFormat='DD/MM/YYYY';

Master_Calendar:
Load
        date,
        date(date,'dd-MMMM-YYYY') as long_date,
        month(date) as month_name
Inline
[
date
03/01/2022
03/02/2022
03/03/2022
03/04/2022
03/05/2022
03/06/2022
03/07/2022
03/08/2022
03/09/2022
```

```
03/10/2022
03/11/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- long_date
- month_name

结果表

日期	long_date	month_name
03/01/2022	2022 年 1 月 3 日	一月
03/02/2022	2022 年 2 月 3 日	二月
03/03/2022	2022 年 3 月 3 日	三月
03/04/2022	2022 年 4 月 3 日	四月
03/05/2022	2023 年 5 月 3 日	五月
03/06/2022	2022 年 6 月 3 日	六月
03/07/2022	2022 年 7 月 3 日	7 月
03/08/2022	2022 年 8 月 3 日	8 月
03/09/2022	2022 年 9 月 3 日	9 月
03/10/2022	2022 年 10 月 3 日	10 月
03/11/2022	2022 年 11 月 3 日	11 月

脚本中的 month() 函数可以正确计算月份的名称。

示例 2–ANSI 日期(脚本)

加载脚本和结果

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名称为 Master_Calendar 的日期数据集。使用 DateFormat 系统变量 DD/MM/YYYY。但是，数据集中包含的日期采用 ANSI 标准日期格式。
- 使用 month() 功能创建另一个名为 month_name 的字段的前置 Load。
- 另一个名为 long_date 的字段，使用 date() 函数表示完整的日期。

加载脚本

```
SET DateFormat='DD/MM/YYYY';
Master_Calendar:
Load
       date,
       date(date,'dd-MMMM-YYYY') as long_date,
       month(date) as month_name

Inline
[
date
2022-01-11
2022-02-12
2022-03-13
2022-04-14
2022-05-15
2022-06-16
2022-07-17
2022-08-18
2022-09-19
2022-10-20
2022-11-21
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- long_date
- month_name

结果表

日期	long_date	month_name
03/11/2022	2022 年 3 月 11 日	11
03/12/2022	2022 年 3 月 12 日	12
03/13/2022	2022 年 3 月 13 日	13
03/14/2022	2022 年 3 月 14 日	14
03/15/2022	2022 年 3 月 15 日	15
03/16/2022	2022 年 3 月 16 日	16
03/17/2022	2022 年 3 月 17 日	17
03/18/2022	2022 年 3 月 18 日	18
03/19/2022	2022 年 3 月 19 日	19

日期	long_date	month_name
03/20/2022	2022 年 3 月 20 日	20
03/21/2022	2022 年 3 月 21 日	21

脚本中的 month() 函数可以正确计算月份的名称。

示例 3 – 未格式化日期(脚本)

加载脚本和结果

概述

打开 数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 名称为 Master_Calendar 的日期数据集。使用 DateFormat 系统变量 DD/MM/YYYY。
- 使用 month() 功能创建另一个名为 month_name 的字段的前置 Load。
- 原始未格式化日期, 命名为 unformatted_date。
- 另一个名为 long_date 的字段, 使用 date() 功能表示完整的日期。

加载脚本

```
SET DateFormat='DD/MM/YYYY';

Master_Calendar:
Load
      unformatted_date,
      date(unformatted_date,'dd-MMMM-YYYY') as long_date,
      month(unformatted_date) as month_name

Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- unformatted_date
- long_date
- month_name

结果表

unformatted_date	long_date	month_name
44868	2022 年 1 月 3 日	一月
44898	2022 年 2 月 3 日	二月
44928	2022 年 3 月 3 日	三月
44958	2022 年 4 月 3 日	四月
44988	2022 年 5 月 3 日	五月
45018	2022 年 6 月 3 日	六月
45048	2022 年 7 月 3 日	7 月
45078	2022 年 8 月 3 日	8 月
45008	2022 年 9 月 3 日	9 月
45038	2022 年 10 月 3 日	10 月
45068	2022 年 11 月 3 日	11 月

脚本中的 month() 函数可以正确计算月份的名称。

示例 4 – 计算到期月

加载脚本和图表表达式

概述

打开 数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名为 Subscriptions 的 3 月订单数据集。表格包含三个字段：
  - id
  - order_date
  - 金额

加载脚本

```
Subscriptions:
Load
        id,
        order_date,
```

```
        amount
Inline
[
id,order_date,amount
1,03/01/2022,231.24
2,03/02/2022,567.28
3,03/03/2022,364.28
4,03/04/2022,575.76
5,03/05/2022,638.68
6,03/06/2022,785.38
7,03/07/2022,967.46
8,03/08/2022,287.67
9,03/09/2022,764.45
10,03/10/2022,875.43
11,03/11/2022,957.35
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：order_date。

要计算订单的到期月份，请创建以下度量：=month(order_date+180)。

结果表

order_date	=month(order_date+180)
03/01/2022	7 月
03/02/2022	8 月
03/03/2022	8 月
03/04/2022	9 月
03/05/2022	10 月
03/06/2022	11 月
03/07/2022	12 月
03/08/2022	一月
03/09/2022	三月
03/10/2022	四月
03/11/2022	五月

month() 函数正确地确定了3月11日下达的订单将于7月到期。

monthend

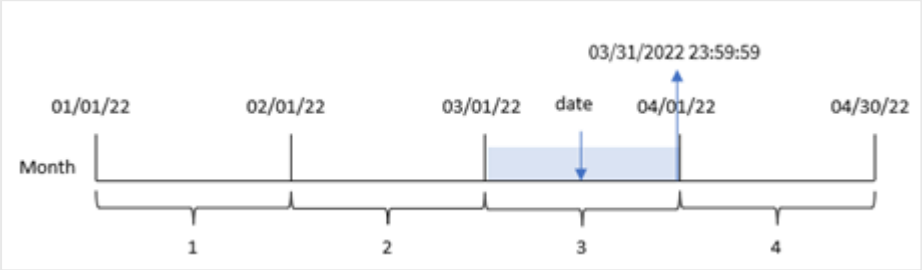
此函数用于返回与包含 date 的月份的最后一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 DateFormat。

语法：

```
MonthEnd(date[, period_no])
```

换句话说，monthend() 函数确定日期属于哪一月。然后以日期格式返回该月最后一毫秒的时间戳。

monthend 函数的图表。



适用场景

当您希望计算使用尚未发生的月份分数时，monthend() 函数用作表达式的一部分。例如，您想计算一月中尚未发生的利息总额。

返回数据类型： 双

参数

参数	说明
date	要评估的日期或时间戳。
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。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
monthend('02/19/2012')	返回 02/29/2012 23:59:59。
monthend('02/19/2001', -1)	返回 01/31/2001 23:59:59。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 采用 DateFormat 系统变量 (MM/DD/YYYY) 格式的日期字段。
- 前置 Load 语句包含：
    - 设置为 'end_of_month' 字段的 monthend() 函数。
    - 设置为 'end_of_month_timestamp' 字段的 timestamp 函数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    monthend(date) as end_of_month,
    timestamp(monthend(date)) as end_of_month_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- end_of_month
- end_of_month_timestamp

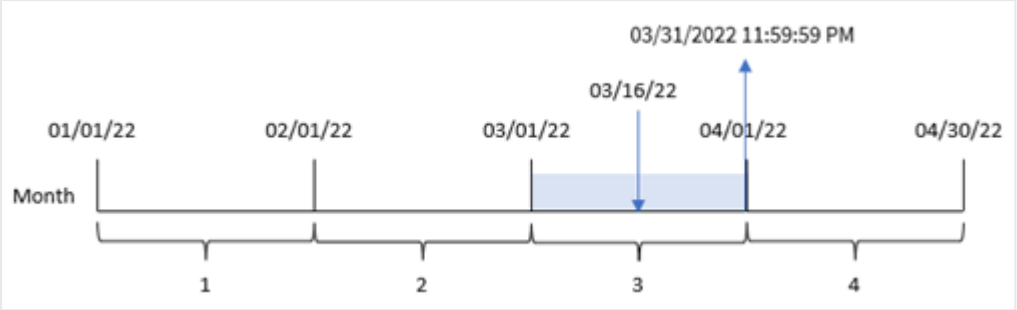
结果表

id	日期	end_of_month	end_of_month_timestamp
8188	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8189	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8199	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8200	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8201	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

通过使用 monthend() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'end_of_month' 字段。

monthend() 函数标识日期值落入哪个月份，返回该月份最后一毫秒的时间戳。

以三月为选定月份的 monthend 函数图。



交易 8192 发生在 3 月 16 日。monthend() 函数返回该月的最后一毫秒，即 3 月 31 日晚上 11:59:59。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

在该示例中，该任务创建一个字段 previous_month_end，该字段返回交易发生前的月末的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    monthend(date,-1) as previous_month_end,
    timestamp(monthend(date,-1)) as previous_month_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
```

```
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

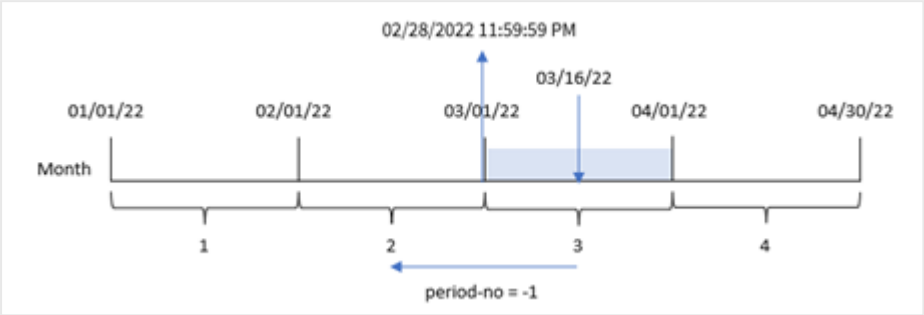
- id
- date
- previous_month_end
- previous_month_end_timestamp

结果表

id	日期	previous_month_end	previous_month_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	01/31/2022	1/31/2022 11:59:59 PM
8191	2/28/2022	01/31/2022	1/31/2022 11:59:59 PM
8192	3/16/2022	02/28/2022	2/28/2022 11:59:59 PM
8193	4/1/2022	03/31/2022	3/31/2022 11:59:59 PM
8194	5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
8195	5/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8196	6/15/2022	05/31/2022	5/31/2022 11:59:59 PM
8197	6/26/2022	05/31/2022	5/31/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	07/31/2022	7/31/2022 11:59:59 PM
8203	8/8/2022	07/31/2022	7/31/2022 11:59:59 PM
8204	8/19/2022	07/31/2022	7/31/2022 11:59:59 PM
8205	9/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8206	10/14/2022	09/30/2022	9/30/2022 11:59:59 PM
8207	10/29/2022	09/30/2022	9/30/2022 11:59:59 PM

monthend() 函数首先标识交易发生的月份，并且将为 -1 的 period_no 作为偏移参数。然后，它在一月前移动，并确定月的最后一毫秒。

monthend 函数的图表，其中具有 period_no 变量.



交易 8192 发生在 3 月 16 日。monthend() 函数标识交易发生的前一个月是二月。然后返回该月的最后一毫秒，即 2 月 28 日晚上 11:59:59。

示例 3 – 图表示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

在本例中，未更改的数据集被加载到应用程序中。任务是创建一个计算，返回交易发生时月末的时间戳，作为应用程序图表中的一个度量。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
```

```
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `id`

要计算交易发生月份的结束日期，请创建以下度量：

- `=monthend(date)`
- `=timestamp(monthend(date))`

结果表

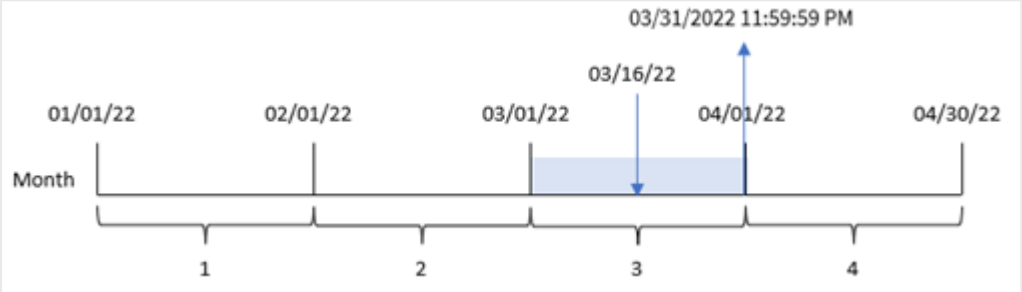
id	日期	=monthend(date)	=timestamp(monthend(date))
8188	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8189	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM
8190	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8191	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8192	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8193	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8194	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8195	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8196	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8197	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8198	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8201	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8202	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8203	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8204	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8205	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM

id	日期	=monthend(date)	=timestamp(monthend(date))
8206	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8207	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM

通过使用 monthend() 函数并将日期字段作为函数的参数传递，在图表中创建了 'end_of_month' 度量。

monthend() 函数标识日期值落入哪个月份，返回该月份最后一毫秒的时间戳。

monthend 函数的图表，其中具有 period_no 变量.



交易 8192 发生在 3 月 16 日。monthend() 函数返回该月的最后一毫秒，即 3 月 31 日晚上 11:59:59。

示例 4 – 场景

加载脚本和结果

概述

在本例中，将数据集加载到名为 'Employee_Expenses' 的表中。表格包含以下字段：

- 员工 ID
- 员工姓名
- 每位员工的平均每日费用报销。

最终用户需要一个图表，该图表按员工 id 和员工姓名显示本月剩余时间的预计费用报销。

加载脚本

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- employee_id
- employee_name

要计算累计利息，请创建该度量：

```
=floor(monthend(today(1),0)-today(1))*avg_daily_claim
```

此度量是动态的，将根据加载数据的日期产生不同的表结果。

将度量的**数字格式**设置为**金额**。

结果表

EmployeeID	employee_name	=floor(monthend(today(1),0)-today(1))*avg_daily_claim
182	Mark	$30.00
183	Deryck	$25.00
184	Dexter	$25.00
185	Sydney	$54.00
186	Agatha	$36.00

monthend() 函数通过使用今天的日期作为唯一参数返回当前月份的结束日期。该表达式通过从月末日期减去今天的日期来返回本月剩余的天数。

然后将该值乘以每个员工的平均每日费用索赔，以计算每个员工在剩余月份预计提出的索赔的估计值。

monthname

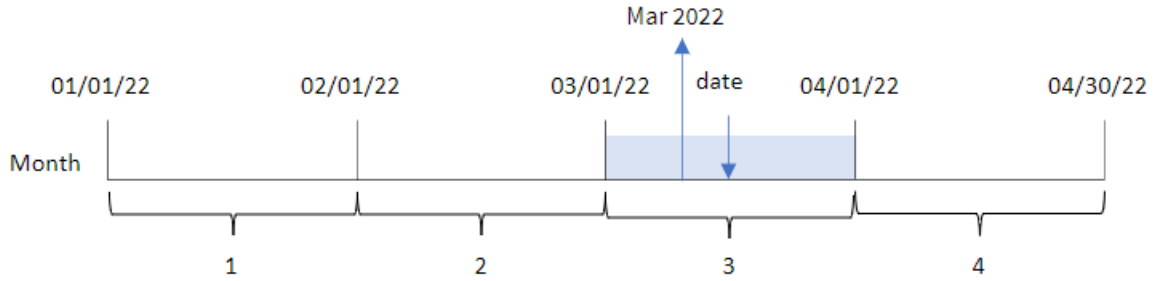
此函数用于返回一个显示值，该值显示该月(根据 **MonthNames** 脚本变量的格式)以及年，伴随一个与该月第一天第一毫秒的时间戳对应的基础数值。

语法：

```
MonthName(date[, period_no])
```

返回数据类型：双

monthname 函数的图表



参数

参数	说明
date	要评估的日期或时间戳。
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。

函数示例

示例	结果
`monthname('10/19/2013')`	返回 Oct 2013
`monthname('10/19/2013', -1)`	返回 Sep 2013

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 transaction_month，该字段返回事务发生前的月末的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        monthname(date) as transaction_month
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

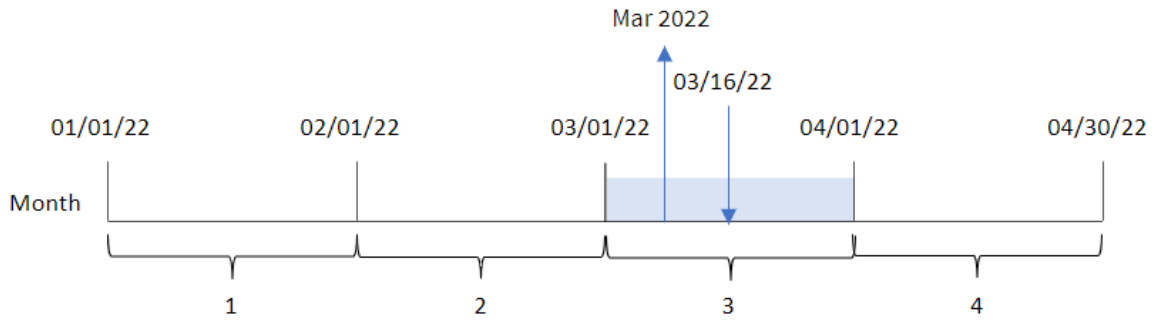
- `date`
- `transaction_month`

结果表

日期	transaction_month
1/7/2022	Jan 2022
1/19/2022	Jan 2022
2/5/2022	Feb 2022
2/28/2022	Feb 2022
3/16/2022	Mar 2022
4/1/2022	Apr 2022
5/7/2022	May 2022
5/16/2022	May 2022
6/15/2022	Jun 2022
6/26/2022	Jun 2022
7/9/2022	Jul 2022
7/22/2022	Jul 2022
7/23/2022	Jul 2022
7/27/2022	Jul 2022
8/2/2022	Aug 2022
8/8/2022	Aug 2022
8/19/2022	Aug 2022
9/26/2022	Sep 2022
10/14/2022	Oct 2022
10/29/2022	Oct 2022

通过使用 monthname() 函数并将 `date` 字段作为函数的参数传递，在前置 Load 语句中创建了 `transaction_month` 字段。

monthname 函数的图表，基本示例



monthname() 函数识别交易 8192 发生在 2022 年 3 月，并使用 MonthNames 系统变量返回该值。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的内联数据集和场景。
- 创建一个字段 transaction_previous_month，该字段返回事务发生前的月末的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        monthname(date,-1) as transaction_previous_month
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
```

```
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `transaction_previous_month`

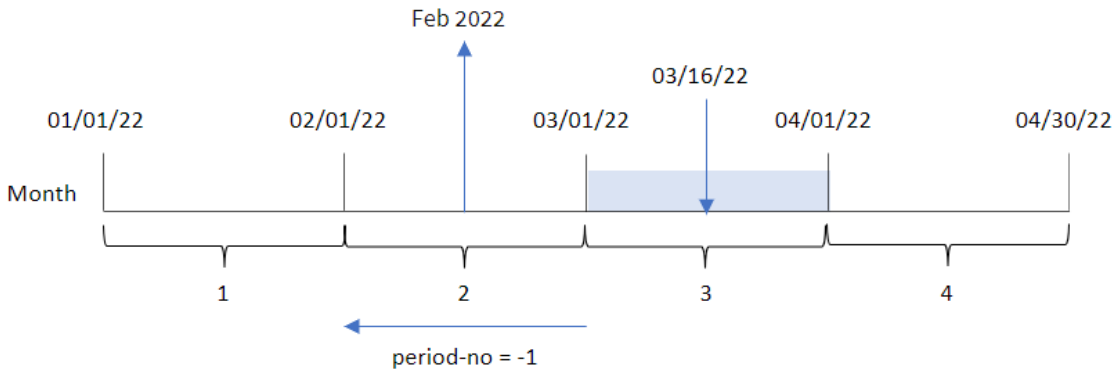
结果表

日期	transaction_previous_month
1/7/2022	Dec 2021
1/19/2022	Dec 2021
2/5/2022	Jan 2022
2/28/2022	Jan 2022
3/16/2022	Feb 2022
4/1/2022	Mar 2022
5/7/2022	Apr 2022
5/16/2022	Apr 2022
6/15/2022	May 2022
6/26/2022	May 2022
7/9/2022	Jun 2022
7/22/2022	Jun 2022
7/23/2022	Jun 2022
7/27/2022	Jun 2022
8/2/2022	Jul 2022
8/8/2022	Jul 2022
8/19/2022	Jul 2022

日期	transaction_previous_month
9/26/2022	Aug 2022
10/14/2022	Sep 2022
10/29/2022	Sep 2022

在本例中，由于 monthname() 函数中使用了为 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的月份。然后转移到一个月前，并返回月份名称和年份。

monthname 函数的图表，period_no 示例



交易 8192 发生在 3 月 16 日。monthname() 函数确定交易发生前的月份是二月，并以 MonthNames 系统变量格式返回该月份以及 2022 年。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。返回事务发生时月末的时间戳的计算作为应用程序的图表对象中的度量创建。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
```

```
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：`date`。

创建以下度量：

```
=monthname(date)
```

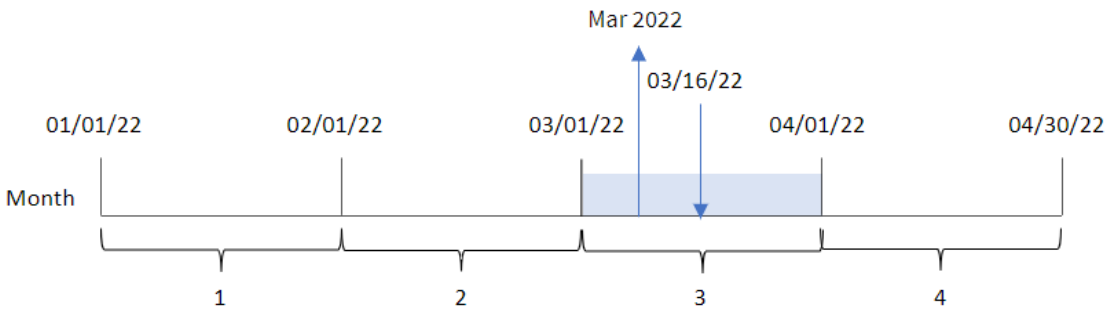
结果表

日期	=monthname(date)
1/7/2022	Jan 2022
1/19/2022	Jan 2022
2/5/2022	Feb 2022
2/28/2022	Feb 2022
3/16/2022	Mar 2022
4/1/2022	Apr 2022
5/7/2022	May 2022
5/16/2022	May 2022
6/15/2022	Jun 2022
6/26/2022	Jun 2022
7/9/2022	Jul 2022
7/22/2022	Jul 2022

日期	=monthname(date)
7/23/2022	Jul 2022
7/27/2022	Jul 2022
8/2/2022	Aug 2022
8/8/2022	Aug 2022
8/19/2022	Aug 2022
9/26/2022	Sep 2022
10/14/2022	Oct 2022
10/29/2022	Oct 2022

通过使用 monthname() 函数并将 date 字段作为函数的参数传递，在图表对象中创建 month_name 度量。

monthname 函数的图表，图表对象示例



monthname() 函数识别交易 8192 发生在 2022 年 3 月，并使用 MonthNames 系统变量返回该值。

monthsend

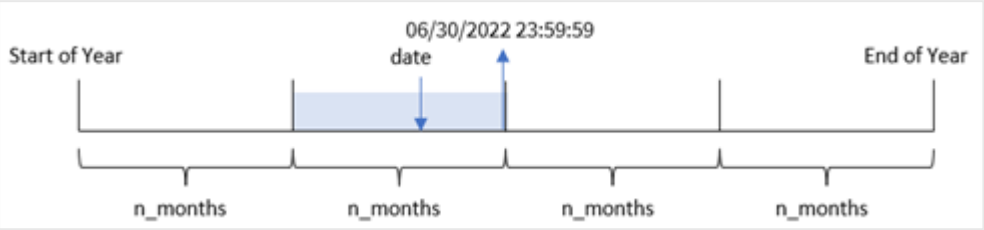
此函数用于返回与包含基准日期的一个月、两个月、季度、四个月期间或半年的最后毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期末的时间戳。默认输出格式是在脚本中设置的 DateFormat。

语法：

```
MonthsEnd(n_months, date[, period_no [, first_month_of_year]])
```

返回数据类型：双

monthsend 函数的图表。



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(四个月期间)或 6(半年)。
date	要评估的日期或时间戳。
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。

monthsend() 函数根据提供的 n_months 参数将一年划分为若干段。然后，它评估所提供的每个日期属于哪个段，并以日期格式返回该段的最后一毫秒。该函数可以从前面或后面的段返回结束时间戳，并重新定义一年的第一个月。

函数中提供了一年中的以下时段作为 n_month 参数。

n_month 参数

期间	月数
月	1
双月	2
季	3
四个月	4
半年	6

适用场景

当用户希望计算使用到目前为止已过的一月的部分时，monthsend() 函数用作表达式的一部分。用户有机会使用变量选择自己选择的时段。例如，monthsend() 可以提供一个输入变量，让用户计算月份、季度或半年内尚未发生的总利息。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
monthsend(4, '07/19/2013')	返回 08/31/2013。
monthsend(4, '10/19/2013', -1)	返回 08/31/2013。
monthsend(4, '10/19/2013', 0, 2)	返回 01/31/2014。 因为该年度的开始变成 2 月。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供的日期字段。
- 前置 Load 语句包含：
  - 设置为 'bi_monthly_end' 字段的 monthsend 函数。这将交易分为两个月的部分。
  - timestamp 函数，返回每个交易的时段的开始时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
```

```
    Load
    *,
    monthsend(2,date) as bi_monthly_end,
    timestamp(monthsend(2,date)) as bi_monthly_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- bi_monthly_end
- bi_monthly_end_timestamp

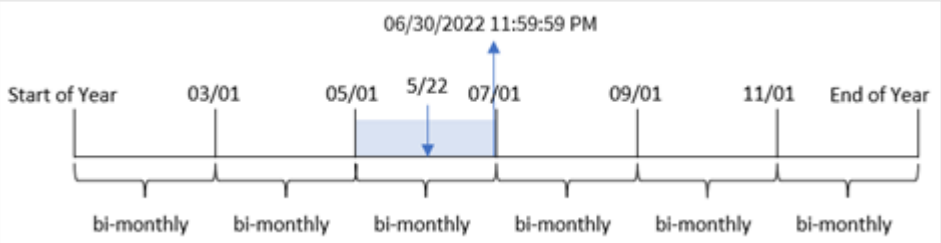
结果表

id	日期	bi_monthly_end	bi_monthly_end_timestamp
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM

id	日期	bi_monthly_end	bi_monthly_end_timestamp
8192	3/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	10/31/2022	10/31/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

`bi_monthly_end` 字段是在前置 Load 语句中使用 `monthsend()` 函数创建的。提供的第一个参数是 `2`，将一年分成两个月的段。第二个参数标识要计算的字段。

带双月段的 `monthsend` 函数的图表。



交易 8195 发生在 5 月 22 日。`monthsend()` 函数最初将一年划分为两个月的段。交易 8195 发生在 5 月至 6 月期间。因此，该函数返回此段的最后一毫秒，即 06/30/2022 11:59:59 PM。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

在本例中，任务是创建一个字段 'prev_bi_monthly_end'，该字段返回交易发生前的双月段的第一毫秒。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    monthsend(2,date,-1) as prev_bi_monthly_end,
    timestamp(monthsend(2,date,-1)) as prev_bi_monthly_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

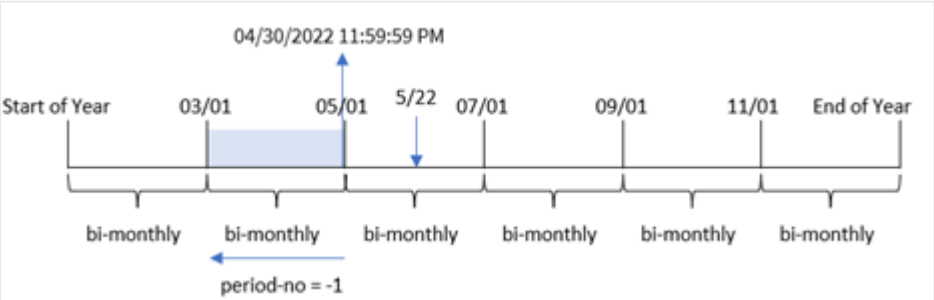
- id
- date
- prev_bi_monthly_end
- prev_bi_monthly_end_timestamp

结果表

id	日期	prev_bi_monthly_end	prev_bi_monthly_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	12/31/2021	12/31/2021 11:59:59 PM
8191	2/28/2022	12/31/2021	12/31/2021 11:59:59 PM
8192	3/16/2022	02/28/2022	2/28/2022 11:59:59 PM
8193	4/1/2022	02/28/2022	2/28/2022 11:59:59 PM
8194	5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
8195	5/22/2022	04/30/2022	4/30/2022 11:59:59 PM
8196	6/15/2022	04/30/2022	4/30/2022 11:59:59 PM
8197	6/26/2022	04/30/2022	4/30/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	06/30/2022	6/30/2022 11:59:59 PM
8203	8/8/2022	06/30/2022	6/30/2022 11:59:59 PM
8204	8/19/2022	06/30/2022	6/30/2022 11:59:59 PM
8205	9/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8206	10/14/2022	08/31/2022	8/31/2022 11:59:59 PM
8207	10/29/2022	08/31/2022	8/31/2022 11:59:59 PM

通过在 `monthsend()` 函数中使用 `-1` 作为 `period_no` 参数，在最初将一年划分为双月段之后，该函数返回上一个双月段的最后一毫秒，直到交易发生。

返回上一个双月段的 `monthsend` 函数图。



交易 8195 发生在 5 月至 6 月期间。因此，上一个双月段在 3 月 1 日至 4 月 30 日之间，因此函数返回该段的最后一毫秒，即 2022 年 4 月 30 号晚上 11:59:59。

示例 3 – first_month_of_year

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

在本例中，组织策略是将四月作为财政年度的第一个月。

创建一个字段 'bi_monthly_end'，该字段将交易分组为两个月一次的段，并返回每个交易段的最后毫秒时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
    *,
    monthsend(2,date,0,4) as bi_monthly_end,
    timestamp(monthsend(2,date,0,4)) as bi_monthly_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

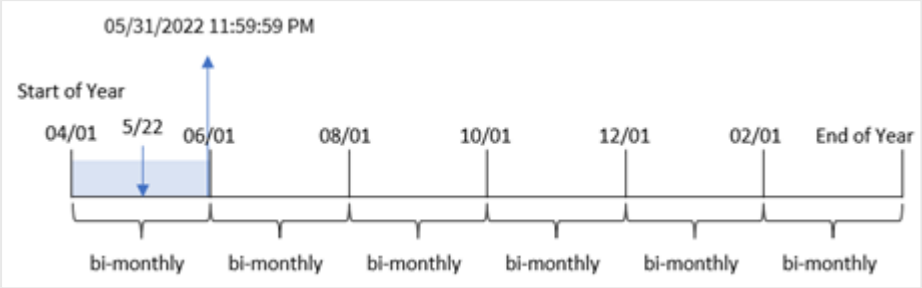
- id
- date
- bi_monthly_end
- bi_monthly_end_timestamp

结果表

id	日期	bi_monthly_end	bi_monthly_end_timestamp
8188	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8189	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	05/31/2022	5/31/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/22/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	07/31/2022	7/31/2022 11:59:59 PM
8197	6/26/2022	07/31/2022	7/31/2022 11:59:59 PM
8198	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8199	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8200	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8201	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	11/30/2022	11/30/2022 11:59:59 PM
8207	10/29/2022	11/30/2022	11/30/2022 11:59:59 PM

通过在 monthsend() 函数中使用 4 作为 first_month_of_year 参数，函数从 4 月 1 日开始一年。然后它将一年分成四个季度。4 月至 5 月、6 月至 7 月、8 月至 9 月、10 月至 11 月、12 月至 1 月、2 月至 3 月。

一年中第一个月设置为 4 月的 monthsend 函数图



交易 8195 发生在 5 月 22 日，属于 4 月 1 日至 5 月 31 日期间。因此，该函数返回此段的最后一毫秒，即 05/31/2022 11:59:59 PM。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。

在本例中，任务是创建一个计算，将交易分组为两个月的交易段，并返回每个交易段的最后一毫秒时间戳，作为应用程序的图表对象中的度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
```

```
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
date
```

要获取交易发生时双月段的最后一毫秒时间戳，请创建以下度量：

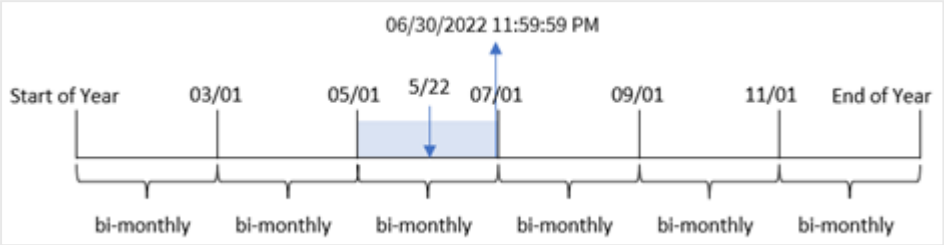
- `=monthsEnd(2,date)`
- `=timestamp(monthsend(2,date))`

结果表

id	日期	=monthsend(2,date)	=timestamp(monthsend(2,date))
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	10/31/2022	10/31/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

通过使用 monthsend() 函数在图表对象中创建 'bi_monthly_end' 字段作为度量。提供的第一个参数是 2，将一年分成两个月的段。第二个参数标识要计算的字段。

带双月段的 monthsend 函数的图表。



交易 8195 发生在 5 月 22 日。monthsend() 函数最初将一年划分为两个月的段。交易 8195 发生在 5 月至 6 月期间。因此，该函数返回此段的第一毫秒，即 06/30/2022 11:59:59 PM。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

在本例中，将数据集加载到名为 'Employee_Expenses' 的表中。表格包含以下字段：

- 员工 ID
- 员工姓名
- 每位员工的平均每日费用报销。

最终用户想要一个图表，该图表按员工 id 和员工姓名显示他们自己选择的剩余时间段的估计费用索赔。财政年度从一月份开始。

加载脚本

```
SET vPeriod = 1;

Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

结果

加载数据并打开新工作表。

在加载脚本开始时，创建了一个变量 `vPeriod`，该变量将绑定到变量输入控件。

进行以下操作：

1. 在资产面板中，单击**自定义对象**。
2. 选择 **Qlik 仪表板捆绑**，并创建**变量输入**对象。
3. 输入图表对象的标题。
4. 在**变量**下，选择 **vPeriod** 作为名称，并将对象设置为显示为**下拉列表**。
5. 在**值**下，单击**动态**值。输入以下内容：
   `='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.`

创建新表并将这些字段创建为维度：

- `employee_id`
- `employee_name`

要计算累计利息，请创建该度量：

```
=floor(monthsend($(vPeriod),today(1))-today(1))*avg_daily_claim
```

此度量是动态的，将根据加载数据的日期产生不同的表结果。

将度量的**数字格式**设置为**金额**。

结果表

EmployeeID	employee_name	=floor(monthsend($(vPeriod),today(1))-today(1))*avg_daily_claim
182	Mark	$1410.00
183	Deryck	$1175.00
184	Dexter	$1175.00
185	Sydney	$2538.00
186	Agatha	$1692.00

`monthsend()` 函数使用用户输入作为其第一个参数，使用今天的日期作为其第二个参数。这将返回用户所选时间段的结束日期。然后，表达式通过从该结束日期减去今天的日期，返回所选时间段剩余的天数。

然后将该值乘以每个员工的平均每日费用索赔，以计算每个员工在该期间剩余天预计提出的索赔的估计值。

monthsname

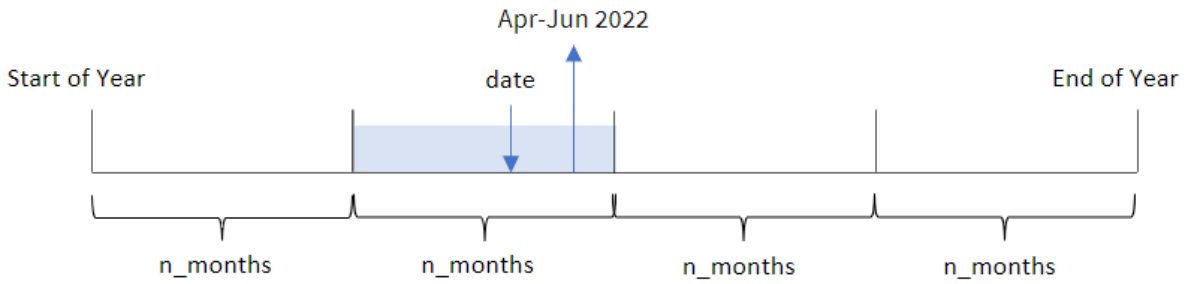
此函数用于返回一个显示值，表示时段各月份（根据 **MonthNames** 脚本变量的格式）和年的范围。基础数值与包含基准日期的一个月、两个月、季度、四个月期间或半年的第一毫秒的时间戳对应。

语法：

```
MonthsName(n_months, date[, period_no[, first_month_of_year]])
```

返回数据类型： 双

monthsname 函数的图表



monthsname() 函数根据提供的 n_months 参数将一年划分为若干段。然后，它评估每个提供的 date 所属的段，并返回该段的开始和结束月份名称以及年份。该函数还提供了从前面或后面的段返回这些边界的能力，以及重新定义一年中的第一个月的能力。

函数中提供了一年中的以下时段作为 n_month 参数：

可能的 n_month 参数

期间	月数
月	1
双月	2
季	3
四个月	4
半年	6

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（四个月期间）或 6（半年）。

参数	说明
date	要评估的日期或时间戳。
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。

适用场景

当您希望向用户提供按其选择的时间段比较聚合的功能时，monthsname() 函数非常有用。例如，您可以提供一个输入变量，让用户查看每月、季度或半年的产品总销售额。

可以在加载脚本中创建这些维度，方法是将函数作为主日历表中的字段添加，或者直接在图表中创建维度作为计算维度。

函数示例

示例	结果
`monthsname(4, '10/19/2013')`	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, '10/19/2013', -1)`	返回 'May-Aug 2013'。
`monthsname(4, '10/19/2013', 0, 2)`	返回 'Oct-Jan 2014'，因为指定年从月份 2 开始。因此，四个月期在次年的第一个月结束。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 bi_monthly_range，将交易分组为双月段，并为每个交易返回该段的边界名称。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        monthsname(2,date) as bi_monthly_range
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

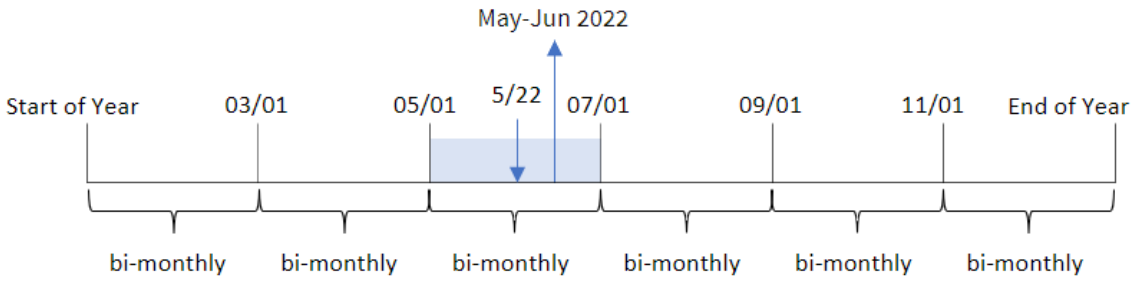
- date
- bi_monthly_range

结果表

日期	bi_monthly_range
2/19/2022	Jan-Feb 2022
3/7/2022	Mar-Apr 2022
3/30/2022	Mar-Apr 2022
4/5/2022	Mar-Apr 2022
4/16/2022	Mar-Apr 2022
5/1/2022	May-Jun 2022
5/7/2022	May-Jun 2022
5/22/2022	May-Jun 2022
6/15/2022	May-Jun 2022
6/26/2022	May-Jun 2022
7/9/2022	Jul-Aug 2022
7/22/2022	Jul-Aug 2022
7/23/2022	Jul-Aug 2022
7/27/2022	Jul-Aug 2022
8/2/2022	Jul-Aug 2022
8/8/2022	Jul-Aug 2022
8/19/2022	Jul-Aug 2022
9/26/2022	Sep-Oct 2022
10/14/2022	Sep-Oct 2022
10/29/2022	Sep-Oct 2022

bi_monthly_range 字段是在前置 Load 语句中使用 monthsname() 函数创建的。提供的第一个参数是 2，将一年分成两个月的段。第二个参数标识要计算的字段。

monthsname 函数的图表，基本示例



交易 8195 发生在 5 月 22 日。monthsname() 函数最初将一年划分为两个月的段。交易 8195 发生在 5 月至 6 月期间。因此，函数以 MonthNames 系统变量格式返回这些月份，以及 2022 年 5 月至 6 月的年份。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的内联数据集和场景。
- 创建一个字段 prev_bi_monthly_range，将交易分组为两个月一次的段，并返回每个交易的先前段边界名称。

根据需要在此处添加其他文本，包括列表等。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        MonthsName(2,date,-1) as prev_bi_monthly_range
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
```

```
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `prev_bi_monthly_range`

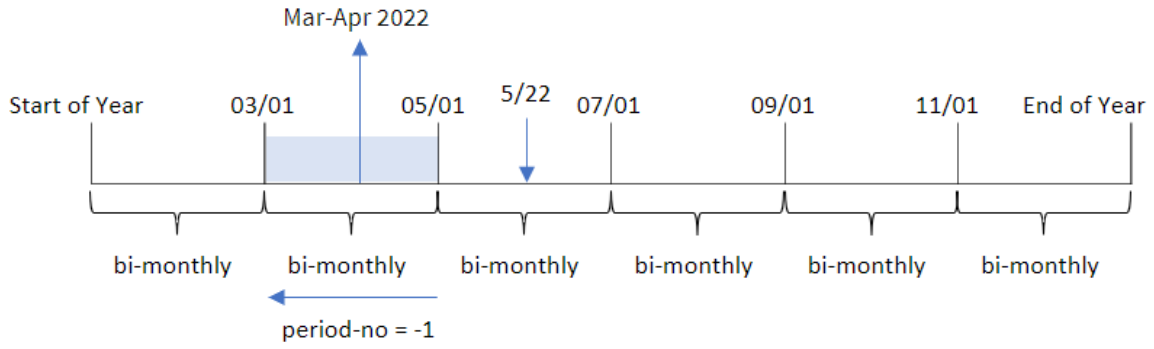
结果表

日期	prev_bi_monthly_range
2/19/2022	Nov-Dec 2021
3/7/2022	Jan-Feb 2022
3/30/2022	Jan-Feb 2022
4/5/2022	Jan-Feb 2022
4/16/2022	Jan-Feb 2022
5/1/2022	Mar-Apr 2022
5/7/2022	Mar-Apr 2022
5/22/2022	Mar-Apr 2022
6/15/2022	Mar-Apr 2022
6/26/2022	Mar-Apr 2022
7/9/2022	May-Jun 2022
7/22/2022	May-Jun 2022
7/23/2022	May-Jun 2022
7/27/2022	May-Jun 2022
8/2/2022	May-Jun 2022

日期	prev_bi_monthly_range
8/8/2022	May-Jun 2022
8/19/2022	May-Jun 2022
9/26/2022	Jul-Aug 2022
10/14/2022	Jul-Aug 2022
10/29/2022	Jul-Aug 2022

在本例中，-1 用作 monthsname() 函数中的 period_no参数。最初将一年划分为两个月段后，该函数返回交易发生时的前一段边界。

monthsname 函数的图表，period_no 示例



交易 8195 发生在 5 月至 6 月期间。因此，上一个双月段在 3 月 1 日至 4 月 30 日之间，因此函数返回 Mar-Apr 2022。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的内联数据集和场景。
- 创建一个不同的字段 bi_monthly_range，将交易分为两个月的段，并返回每个交易的段边界。

然而，在本例中，我们还需要将 4 月设置为财政年度的第一个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
    Load
        *,
        MonthsName(2,date,0,4) as bi_monthly_range
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- bi_monthly_range

结果表

日期	bi_monthly_range
2/19/2022	Feb-Mar 2021
3/7/2022	Feb-Mar 2021
3/30/2022	Feb-Mar 2021
4/5/2022	Apr-May 2022
4/16/2022	Apr-May 2022
5/1/2022	Apr-May 2022

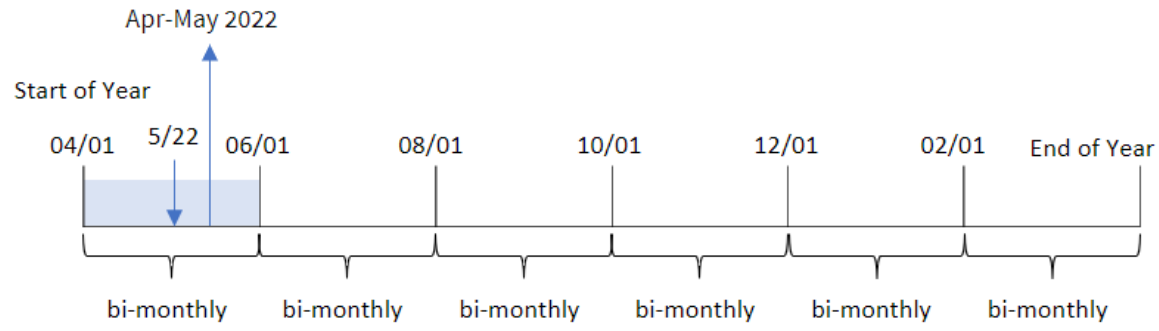
日期	bi_monthly_range
5/7/2022	Apr-May 2022
5/22/2022	Apr-May 2022
6/15/2022	Jun-Jul 2022
6/26/2022	Jun-Jul 2022
7/9/2022	Jun-Jul 2022
7/22/2022	Jun-Jul 2022
7/23/2022	Jun-Jul 2022
7/27/2022	Jun-Jul 2022
8/2/2022	Aug-Sep 2022
8/8/2022	Aug-Sep 2022
8/19/2022	Aug-Sep 2022
9/26/2022	Aug-Sep 2022
10/14/2022	Oct-Nov 2022
10/29/2022	Oct-Nov 2022

通过在 `monthsname()` 函数中使用 4 作为 `first_month_of_year` 参数，函数从 4 月 1 日开始一年。然后它将一年分成四个季度。Apr-May,Jun-Jul,Aug-Sep,Oct-Nov,Dec-Jan,Feb-Mar。

结果的段落文本。

交易 8195 发生在 5 月 22 日，属于 4 月 1 日至 5 月 31 日期间。因此，函数返回 Apr-May 2022。

monthsname 函数 first_month_of_year 示例的图表



示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。将交易分组为两个月段并返回每个交易的段边界的计算是作为应用程序的图表对象中的度量创建的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：`date`。

创建以下度量：

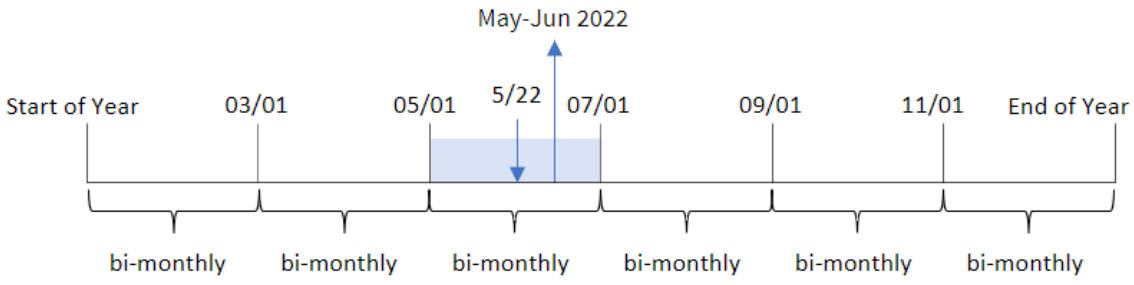
```
=monthsname(2,date)
```

结果表

日期	=monthsname(2,date)
2/19/2022	Jan-Feb 2022
3/7/2022	Mar-Apr 2022
3/30/2022	Mar-Apr 2022
4/5/2022	Mar-Apr 2022
4/16/2022	Mar-Apr 2022
5/1/2022	May-Jun 2022
5/7/2022	May-Jun 2022
5/22/2022	May-Jun 2022
6/15/2022	May-Jun 2022
6/26/2022	May-Jun 2022
7/9/2022	Jul-Aug 2022
7/22/2022	Jul-Aug 2022
7/23/2022	Jul-Aug 2022
7/27/2022	Jul-Aug 2022
8/2/2022	Jul-Aug 2022
8/8/2022	Jul-Aug 2022
8/19/2022	Jul-Aug 2022
9/26/2022	Sep-Oct 2022
10/14/2022	Sep-Oct 2022
10/29/2022	Sep-Oct 2022

通过使用 `monthsname()` 函数在图表对象中创建 `bi_monthly_range` 字段作为度量。提供的第一个参数是 2，将一年分成两个月的段。第二个参数标识要计算的字段。

monthsname 函数的图表，图表对象示例



交易 8195 发生在 5 月 22 日。monthsname() 函数最初将一年划分为两个月的段。交易 8195 发生在 5 月至 6 月期间。因此，函数以 MonthNames 系统变量格式返回这些月份，以及 2022 年 5 月至 6 月的年份。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

最终用户想要一个图表对象，该对象按他们自己选择的时间段显示总销售额。即使该维度在数据模型中不可用，也可以使用 monthsname() 函数作为由变量输入控件动态修改的计算维度来实现。

加载脚本

```
SET vPeriod = 1;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/7/2022',17.17
8189,'1/19/2022',37.23
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

加载数据并打开工作表。

在加载脚本开始时，创建了一个变量 (vPeriod)，该变量将绑定到变量输入控件。接下来，将变量配置为工作表中的自定义对象。

执行以下操作：

1. 在资产面板中，单击**自定义对象**。
2. 选择 **Qlik 仪表板捆绑**，并创建**变量输入**对象。
3. 输入图表对象的标题。
4. 在**变量**下，选择 **vPeriod** 作为名称，并将对象设置为显示为**下拉列表**。
5. 在**值**下，将对象配置为使用动态值。输入以下内容：
   ```
   ='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'
   ```

接下来，创建结果表。

执行以下操作：

1. 创建新表并添加以下计算维度：
   ```
   =monthsname($(vPeriod),date)
   ```
2. 添加此度量以计算总销售额：
   ```
   =sum(amount)
   ```
3. 将度量的**数字格式**设置为**金额**。单击 ✔ **完成编辑**。现在可以通过调整变量对象中的时间段来修改表中显示的数据。

这是选择 tertial 选项时结果表的外观：

结果表

monthsname($(vPeriod),date)	=sum(amount)
Jan-Apr 2022	253.89
May-Aug 2022	713.58
Sep-Dec 2022	248.12

monthsstart

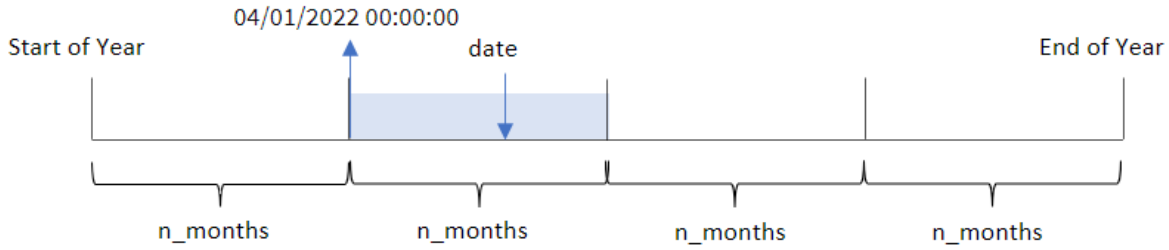
此函数用于返回与包含基准日期的一个月、两个月、季度、四个月期间或半年的第一毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。默认输出格式是在脚本中设置的 **DateFormat**。

语法：

```
MonthsStart(n_months, date[, period_no [, first_month_of_year]])
```

返回数据类型：双

monthsstart() 函数的图表



monthsstart() 函数根据提供的 n_months 参数将一年划分为若干段。然后，它评估所提供的每个日期属于哪个段，并以日期格式返回该段的第一毫秒。该函数还提供了从前面或后面的段返回开始时间戳的功能，以及重新定义一年中的第一个月。

函数中提供了一年中的以下时段作为 n_month 参数：

可能的 n_month 参数

期间	月数
月	1
双月	2
季	3
四个月	4
半年	6

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(四个月期间)或 6(半年)。
date	要评估的日期或时间戳。
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。

适用场景

当用户希望计算使用当前期间尚未发生的分数时，monthsstart() 函数通常用作表达式的一部分。例如，这可以用来提供一个输入变量，让用户计算到目前为止该月、季度或半年累计的总利息。

函数示例

示例	结果
monthsstart(4, '10/19/2013')	返回 09/01/2013。
monthsstart(4, '10/19/2013, -1)	返回 05/01/2013。
monthsstart(4, '10/19/2013', 0, 2)	返回 10/01/2013，因为该年度的开始时间变成 2 月。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 bi_monthly_start，将交易分组为两个月一次的时段，并返回每个交易时段的开始时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        monthsstart(2,date) as bi_monthly_start,
        timestamp(monthsstart(2,date)) as bi_monthly_start_timestamp
```

```
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- bi_monthly_start
- bi_monthly_start_timestamp

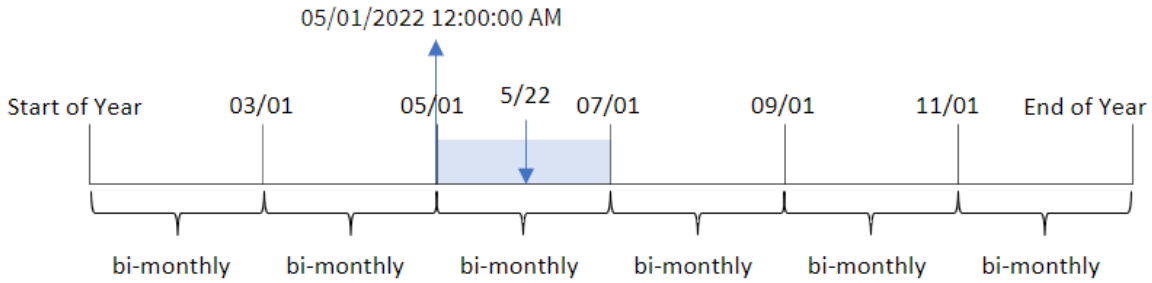
结果表

日期	bi_monthly_start	bi_monthly_start_timestamp
2/19/2022	01/01/2022	1/1/2022 12:00:00 AM
3/7/2022	03/01/2022	3/1/2022 12:00:00 AM
3/30/2022	03/01/2022	3/1/2022 12:00:00 AM
4/5/2022	03/01/2022	3/1/2022 12:00:00 AM
4/16/2022	03/01/2022	3/1/2022 12:00:00 AM
5/1/2022	05/01/2022	5/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/22/2022	05/01/2022	5/1/2022 12:00:00 AM

日期	bi_monthly_start	bi_monthly_start_timestamp
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

`bi_monthly_start` 字段是在前置 Load 语句中使用 `monthsstart()` 函数创建的。提供的第一个参数是 2，将一年分成两个月的段。第二个参数标识要计算的字段。

`monthsstart()` 函数图表，示例没有额外参数



交易 8195 发生在 5 月 22 日。`monthsstart()` 函数最初将一年划分为两个月的段。交易 8195 发生在 5 月至 6 月期间。因此，该函数返回此段的第一毫秒，即 2022 年 5 月 1 日中午 12:00:00。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 prev_bi_monthly_start，该字段返回交易发生前的双月段的第一毫秒。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        monthsstart(2,date,-1) as prev_bi_monthly_start,
           timestamp(monthsstart(2,date,-1)) as prev_bi_monthly_start_timestamp
           ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

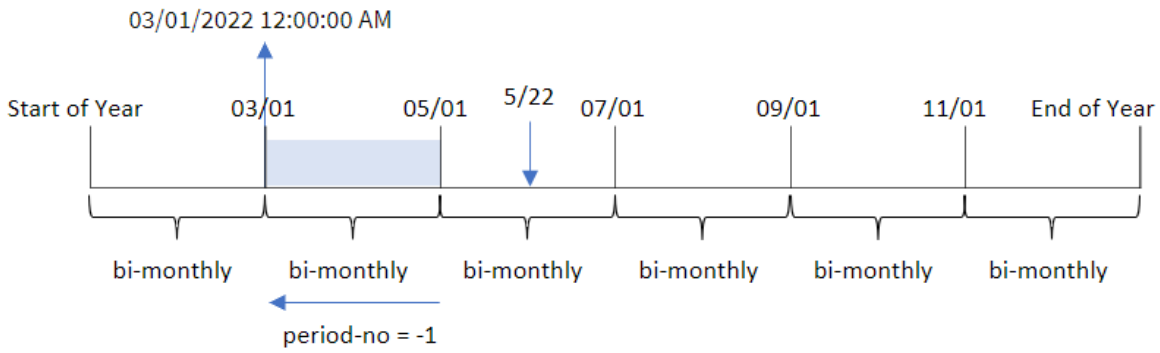
- date
- prev_bi_monthly_start
- prev_bi_monthly_start_timestamp

结果表

日期	prev_bi_monthly_start	prev_bi_monthly_start_timestamp
2/19/2022	11/01/2021	11/1/2021 12:00:00 AM
3/7/2022	01/01/2022	1/1/2022 12:00:00 AM
3/30/2022	01/01/2022	1/1/2022 12:00:00 AM
4/5/2022	01/01/2022	1/1/2022 12:00:00 AM
4/16/2022	01/01/2022	1/1/2022 12:00:00 AM
5/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	03/01/2022	3/1/2022 12:00:00 AM
5/22/2022	03/01/2022	3/1/2022 12:00:00 AM
6/15/2022	03/01/2022	3/1/2022 12:00:00 AM
6/26/2022	03/01/2022	3/1/2022 12:00:00 AM
7/9/2022	05/01/2022	5/1/2022 12:00:00 AM
7/22/2022	05/01/2022	5/1/2022 12:00:00 AM
7/23/2022	05/01/2022	5/1/2022 12:00:00 AM
7/27/2022	05/01/2022	5/1/2022 12:00:00 AM
8/2/2022	05/01/2022	5/1/2022 12:00:00 AM
8/8/2022	05/01/2022	5/1/2022 12:00:00 AM
8/19/2022	05/01/2022	5/1/2022 12:00:00 AM
9/26/2022	07/01/2022	7/1/2022 12:00:00 AM
10/14/2022	07/01/2022	7/1/2022 12:00:00 AM
10/29/2022	07/01/2022	7/1/2022 12:00:00 AM

通过在 `monthsstart()` 函数中使用 `-1` 作为 `period_no` 参数，在最初将一年划分为双月段之后，该函数返回第一个双月段的最后一毫秒，直到交易发生。

monthsstart() 函数的图表，period_no 示例



交易 8195 发生在 5 月至 6 月期间。因此，上一个双月段是在 3 月 1 日至 4 月 30 日之间，因此函数返回该段的第一毫秒，即 2022 年 3 月 1 号中午 12:00:00。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 bi_monthly_start，将交易分组为两个月一次的时段，并返回每个交易集的开始时间戳。

然而，在本例中，我们还需要将 4 月设置为财政年度的第一个月。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        monthsstart(2,date,0,4) as bi_monthly_start,
        timestamp(monthsstart(2,date,0,4)) as bi_monthly_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
```

```
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- bi_monthly_start
- bi_monthly_start_timestamp

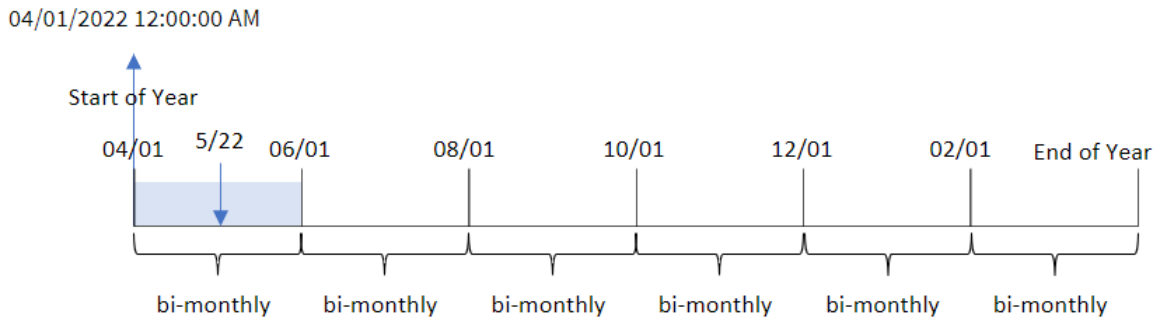
结果表

日期	bi_monthly_start	bi_monthly_start_timestamp
2/19/2022	02/01/2022	2/1/2022 12:00:00 AM
3/7/2022	02/01/2022	2/1/2022 12:00:00 AM
3/30/2022	02/01/2022	2/1/2022 12:00:00 AM
4/5/2022	04/01/2022	4/1/2022 12:00:00 AM
4/16/2022	04/01/2022	4/1/2022 12:00:00 AM
5/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/22/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM

日期	bi_monthly_start	bi_monthly_start_timestamp
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
9/26/2022	08/01/2022	8/1/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

通过在 `monthsstart()` 函数中使用 4 作为 `first_month_of_year` 参数，函数从 4 月 1 日开始一年。然后它将一年分成四个季度。Apr-May,Jun-Jul,Aug-Sep,Oct-Nov,Dec-Jan,Feb-Mar。

`monthsstart()` 函数 first_month_of_year 示例的图表



交易 8195 发生在 5 月 22 日，属于 4 月 1 日至 5 月 31 日期间。因此，该函数返回此段的第一毫秒，即 2022 年 4 月 1 日中午 12:00:00。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。将季度分组为两个月一次的段并返回每个季度的开始时间戳的计算是作为应用程序的图表对象中的度量创建的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
```

```
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量：

```
=monthsstart(2,date)
```

```
=timestamp(monthsstart(2,date))
```

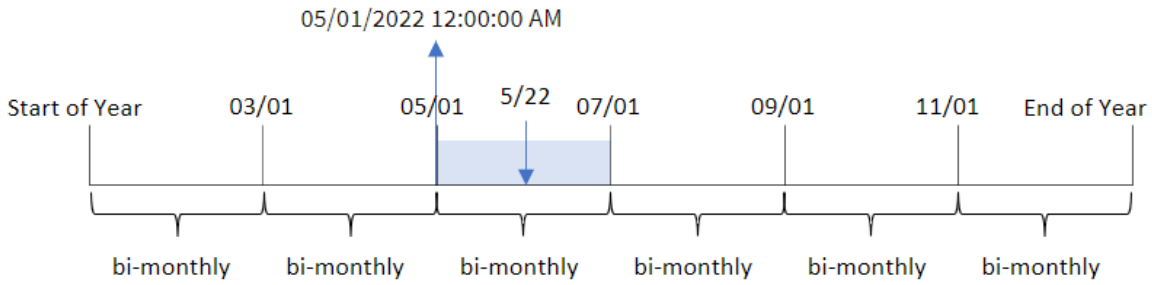
这些计算将检索每个交易发生的双月段的开始时间戳。

结果表

日期	=monthsstart(2,date)	=timestamp(monthsstart(2,date))
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM

日期	=monthsstart(2,date)	=timestamp(monthsstart(2,date))
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
5/1/2022	05/01/2022	5/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/22/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
3/7/2022	03/01/2022	3/1/2022 12:00:00 AM
3/30/2022	03/01/2022	3/1/2022 12:00:00 AM
4/5/2022	03/01/2022	3/1/2022 12:00:00 AM
4/16/2022	03/01/2022	3/1/2022 12:00:00 AM
2/19/2022	01/01/2022	1/1/2021 12:00:00 AM

monthsstart() 函数的图表，图表对象示例



交易 8195 发生在 5 月 22 日。monthsstart() 函数最初将一年划分为两个月的段。交易 8195 发生在 5 月至 6 月期间。因此，该函数返回此段的第一毫秒，即 2022 年 5 月 1 日中午 12:00:00。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含一组贷款余额的数据集，该数据集加载到名为 Loans 的表中。
- 数据包括贷款 ID、月初余额和每年每笔贷款的简单利率。

最终用户想要一个图表对象，该对象按贷款 ID 显示在他们选择的时期内每个贷款的当前利息。财政年度从一月份开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

结果

加载数据并打开工作表。

在加载脚本开始时，创建了一个变量 (vPeriod)，该变量将绑定到变量输入控件。接下来，将变量配置为工作表中的自定义对象。

执行以下操作：

1. 在资产面板中，单击**自定义对象**。
2. 选择 **Qlik 仪表板捆绑**，并创建**变量输入**对象。
3. 输入图表对象的标题。
4. 在**变量**下，选择 **vPeriod** 作为名称，并将对象设置为显示为**下拉列表**。
5. 在**值**下，将对象配置为使用动态值。输入以下内容：
   `='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'`

接下来，创建结果表。

执行以下操作：

1. 新建表格。添加以下字段作为维度：
   - employee_id
   - employee_name
2. 创建一个度量来计算累计利息：
   `=start_balance*(rate*(today(1)-monthsstart($(vPeriod),today(1)))/365)`

3. 将度量的**数字格式**设置为**金额**。单击 ✔ **完成编辑**。现在可以通过调整变量对象中的时间段来修改表中显示的数据。

这是选择 month 期间选项时结果表的外观：

结果表

loan_id	start_ balance	=start_balance*(rate*(today(1)-monthsstart($(vPeriod),today (1)))/365)
8188	$10000.00	$7.95
8189	$15000.00	$67.93
8190	$17500.00	$33.37
8191	$21000.00	$56.73
8192	$90000.00	$600.66

monthsstart() 函数，使用用户的输入作为其第一个参数，使用今天的日期作为其第二个参数，返回用户选择的期间的开始日期。通过从当前日期中减去该结果，表达式将返回本期间迄今为止经过的天数。

然后将该值乘以利率并除以 365，以返回该期间产生的实际利率。然后将结果乘以贷款的起始余额，以返回本期间迄今为止累计的利息。

monthstart

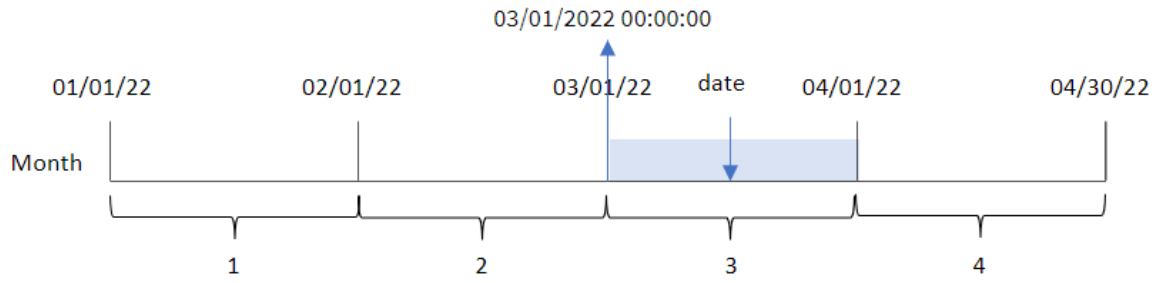
此函数用于返回与包含 **date** 的月份的第一天的第一毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
MonthStart(date[, period_no])
```

返回数据类型：双

monthstart() 函数的图表



monthstart() 函数确定日期属于哪个月。然后以日期格式返回该月第一毫秒的时间戳。

参数

参数	说明
date	要评估的日期或时间戳。
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。

适用场景

当用户希望计算使用到目前为止已过的一月的部分时，monthstart() 函数通常用作表达式的一部分。例如，它可用于计算截至某一日期的一个月内累计的利息。

函数示例

示例	结果
monthstart('10/19/2001')	返回 10/01/2001。
monthstart('10/19/2001', -1)	返回 09/01/2001。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建字段 start_of_month，其返回交易发生的月份开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
    Load
        *,
        monthstart(date) as start_of_month,
        timestamp(monthstart(date)) as start_of_month_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- start_of_month
- start_of_month_timestamp

结果表

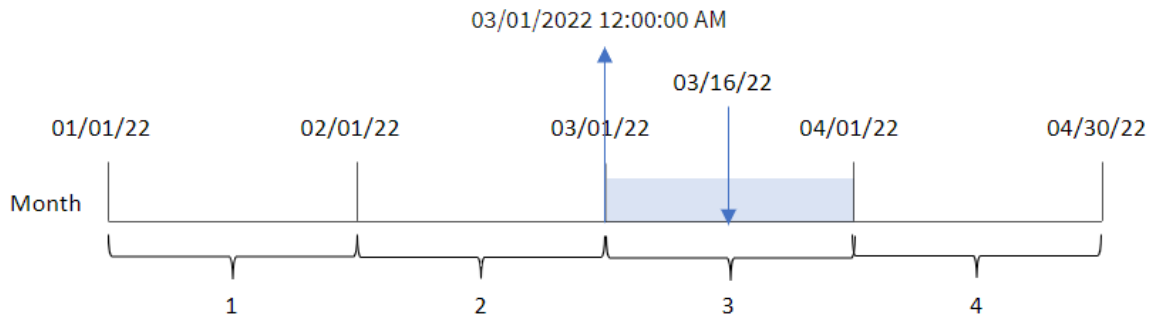
日期	start_of_month	start_of_month_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/01/2022	2/1/2022 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM

日期	start_of_month	start_of_month_timestamp
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	07/01/2022	6/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

通过使用 `monthstart()` 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 `start_of_month` 字段。

`monthstart()` 函数标识日期值落入哪个月份，返回该月份第一毫秒的时间戳。

`monthstart()` 函数图表，示例没有额外参数



交易 8192 发生在 3 月 16 日。`monthstart()` 函数返回当月的第一毫秒，即 3 月 1 日中午 12:00:00。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建字段 previous_month_start，它返回交易发生前一个月开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        monthstart(date,-1) as previous_month_start,
        timestamp(monthstart(date,-1)) as previous_month_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

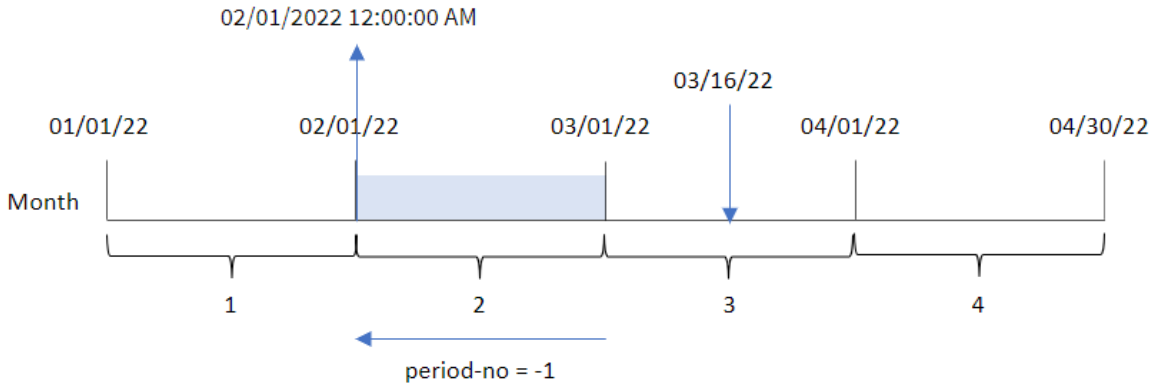
- date
- previous_month_start
- previous_month_start_timestamp

结果表

日期	previous_month_start	previous_month_start_timestamp
1/7/2022	12/01/2021	12/1/2021 12:00:00 AM
1/19/2022	12/01/2021	12/1/2021 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	02/01/2022	2/1/2022 12:00:00 AM
4/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/16/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	08/01/2022	8/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

在本例中，由于 monthstart() 函数中使用了为 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的月份。然后，它在一月前移动，并确定月的第一毫秒。

monthstart() 函数的图表，period_no 示例



交易 8192 发生在 3 月 16 日。monthstart() 函数标识交易发生的前一个月是二月。然后返回当月的第一毫秒，即 2 月 1 日中午 12:00:00。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生时月初的时间戳的计算作为应用程序的图表对象中的度量创建。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

要计算交易发生月份的开始日期，请创建以下度量：

- =monthstart(date)
- =timestamp(monthstart(date))

结果表

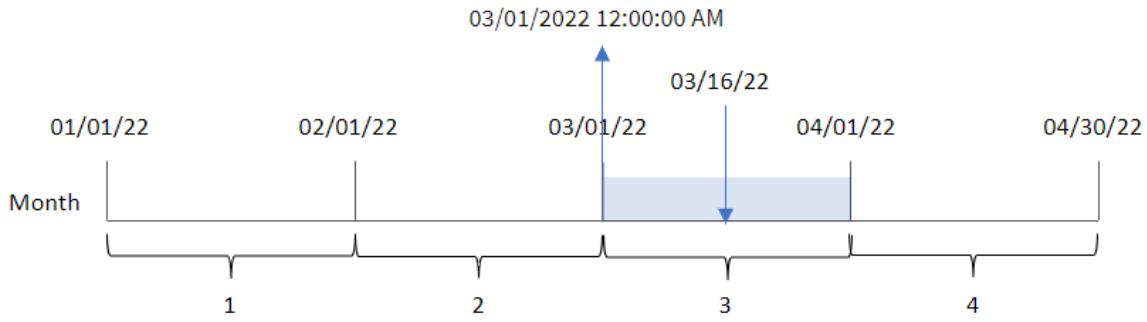
日期	=monthstart(date)	=timestamp(monthstart(date))
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/01/2022	5/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM

日期	=monthstart(date)	=timestamp(monthstart(date))
2/28/2022	02/01/2022	2/1/2022 12:00:00 AM
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM

通过使用 monthstart()函数并将日期字段作为函数的参数传递，在图表对象中创建了 start_of_month 度量。

monthstart() 函数标识日期值落入哪个月份，返回该月份第一毫秒的时间戳。

monthstart() 函数的图表，图表对象示例



交易 8192 发生在 3 月 16 日。monthstart() 函数标识交易发生在 3 月份，并返回该月份的第一毫秒，即 3 月 1 日中午 12:00:00。

示例 4 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含一组贷款余额的数据集，该数据集加载到名为 Loans 的表中。
- 数据包括贷款 ID、月初余额和每年每笔贷款的简单利率。

最终用户希望有一个图表对象，该对象按贷款 ID 显示月初至今每笔贷款的应计利息。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Loans:
Load
*
```

```
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将这些字段添加为维度：
   - loan_id
   - start_balance
2. 接下来，创建一个度量来计算累计利息：
   `=start_balance*(rate*(today(1)-monthstart(today(1)))/365)`
3. 将度量的**数字格式**设置为**金额**。

结果表

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8188	$10000.00	$16.44
8189	$15000.00	$58.56
8190	$17500.00	$28.77
8191	$21000.00	$48.90
8192	$90000.00	$517.81

monthstart() 函数使用今天的日期作为唯一参数，返回当前月份的开始日期。通过从当前日期中减去该结果，表达式将返回本月迄今为止经过的天数。

然后将该值乘以利率并除以 365，以返回该期间产生的实际利率。然后将结果乘以贷款的起始余额，以返回本月迄今为止累计的利息。

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

日历图表显示 networkdays 函数返回的日期范围

networkdays 函数具有以下限制：

- 没有修改工作日的方法。换言之，除了周一至周五的工作之外，没有办法修改区域或情况的函数。
- holiday 参数必须是字符串常数。不接受表达式。

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。假日表示为字符串常量日期。您可以指定多个假期日期，以逗号分隔。 **示例：** '12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'

适用场景

当用户希望计算使用两个日期之间的工作周天数时，networkdays() 函数通常用作表达式的一部分。例如，如果用户希望计算员工在 PAYE(即收即付)合同中的总工资。

函数示例

示例	结果
networkdays ('12/19/2013', '01/07/2014')	返回 14。以下示例没有将假期考虑在内。
networkdays ('12/19/2013', '01/07/2014', '12/25/2013', '12/26/2013')	返回 12。以下示例将 12/25/2013 至 12/26/2013 的假期考虑在内。
networkdays ('12/19/2013', '01/07/2014', '12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014')	返回 10。以下示例将两个假期考虑在内。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含项目 ID、开始日期和结束日期的数据集。该信息加载到名为 Projects 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个额外的字段 net_work_days，以计算每个项目所涉及的工作日数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
    Load
        *,
        networkdays(start_date,end_date) as net_work_days
        ;
Load
id,
start_date,
```

```
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- end_date
- net_work_days

结果表

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	13

由于没有计划的假日(这将出现在 networkdays() 函数的第三个参数中)，函数从 end_date 中以及所有周末中减去 start_date，以计算两个日期之间的工作日数。

突出显示项目 5 工作日的日历图(无节假日)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

上面的日历以为 5 的 id 从视觉上勾勒出项目的轮廓。项目 5 于 2022 年 8 月 10 日星期三开始，2022 年 9 月 26 日结束。由于忽略了所有周六和周日，这两个日期之间(包括在内)有 13 个工作日。

示例 2 - 单假期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与上一个示例相同的数据集和场景。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个额外的字段 net_work_days，以计算每个项目所涉及的工作日数。

在这个例子中，2022 年 8 月 19 日有一天的假期。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
    Load
        *,
```

```
        networkdays(start_date,end_date,'08/19/2022') as net_work_days
        ;
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- end_date
- net_work_days

结果表

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	12

单个计划假日作为 networkdays() 函数中的第三个参数输入。

突出显示项目 5 工作日的日历图(单个假日)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

上面的日历直观地勾勒了项目 5, 展示了将假日包括在内的调整。这个假期发生在 2022 年 8 月 19 日星期五的项目 5 期间。因此, 项目 5 的总 net_work_days 值减少了一天, 从 13 天减少到 12 天。

示例 3 - 多个假期

加载脚本和结果

概述

打开数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 与第一个示例相同的数据集和场景。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个额外的字段 net_work_days, 以计算每个项目所涉及的工作日数。

然而, 在本例中, 有四个假期计划于 2022 年 8 月 18 日至 8 月 21 日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
    Load
        *,
        networkdays(start_date,end_date,'08/18/2022','08/19/2022','08/20/2022','08/21/2022')
as net_work_days
        ;
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- end_date
- net_work_days

结果表

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	11

从 networkdays() 函数的第三个参数开始，以逗号分隔的列表形式输入四个计划假日。

突出显示项目 5 工作日的日历图(多个节假日)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18 Holiday	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

上面的日历直观地勾勒了项目 5，展示了将这些假日包括在内的调整。这段计划假期发生在项目 5 期间，其中两天发生在星期四和星期五。因此，项目 5 的总 net_work_days 值从 13 天减少到 11 天。

示例 4 - 单假期

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

2022 年 8 月 19 日有一天的假期。

然而，在本例中，未更改的数据集被加载到应用程序中。net_work_days 字段作为图表对象中的度量计算。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Projects:
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- start_date
- end_date

创建以下度量：

```
= networkdays(start_date,end_date,'08/19/2022')
```

结果表

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	12

单个计划假日作为 networkdays() 函数中的第三个参数输入。

日历图，显示带单个假日的净工作日(图表对象)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

上面的日历直观地勾勒了项目 5，展示了将假日包括在内的调整。这个假期发生在 2022 年 8 月 19 日星期五的项目 5 期间。因此，项目 5 的总 net_work_days 值减少了一天，从 13 天减少到 12 天。

now

此函数用于返回当前时间的时间戳。函数以 **TimeStamp** 系统变量格式返回值。默认 **timer_mode** 值为 1。

语法：

```
now([ timer_mode])
```

返回数据类型： 双

now() 函数可以在加载脚本或图表对象中使用。

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。*

now() 函数具有高性能影响，如果在表的表达式中使用该函数，则可能会导致滚动问题。当不是绝对必要使用它的时候，我们建议使用 today() 函数。如果布局中需要使用 now()，我们建议在可能的情况下使用非默认设置 now(0) 或 now(2)，因为它们不需要不断重新计算

适用场景

now() 函数通常用作表达式中的组件。例如，它可以用于计算产品生命周期中剩余的时间。当表达式需要使用一天的一小部分时，将使用 now() 函数代替 today() 函数。

下表提供了 now() 函数返回的结果的解释，给出了 timer_mode 参数的不同值：

函数示例

timer_mode 值	结果(如果在加载脚本中使用)	在图表对象中使用时的结果
0	以 TimeStamp 系统变量格式返回最近一次数据重新加载之前最后一次成功数据重新加载的时间戳。	以 TimeStamp 系统变量格式返回最新数据重新加载的时间戳。
1	以 TimeStamp 系统变量格式返回最新数据重新加载的时间戳。	以 TimeStamp 系统变量格式返回函数调用的时间戳。
2	以 TimeStamp 系统变量格式返回应用程序中用户会话开始的时间戳。除非用户重新加载脚本，否则不会更新该脚本。	以 TimeStamp 系统变量格式返回应用程序中用户会话开始的时间戳。一旦新会话开始或重新加载应用程序中的数据，将刷新此属性。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 使用加载脚本生成对象

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

本例使用 now() 函数创建三个变量。每个变量都使用其中一个 timer_mode 选项来演示其效果。

为了演示变量的用途，请重新加载脚本，然后在一段短时间后再次重新加载脚本。这将导致 now(0) 和 now(1) 变量显示不同的值，从而正确地显示它们的用途。

加载脚本

```
LET vPreviousDataLoad = now(0);
LET vCurrentDataLoad = now(1);
LET vApplicationOpened = now(2);
```

结果

第二次加载数据后，使用下面的说明创建三个文本框。

首先，为先前加载的数据创建一个文本框。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 将以下度量值添加到对象：
   =vPreviousDataLoad
3. 在**外观**下，选择 **Show titles** 并向对象添加标题“上次重新加载时间”。

接下来，为当前正在加载的数据创建一个文本框。

执行以下操作：

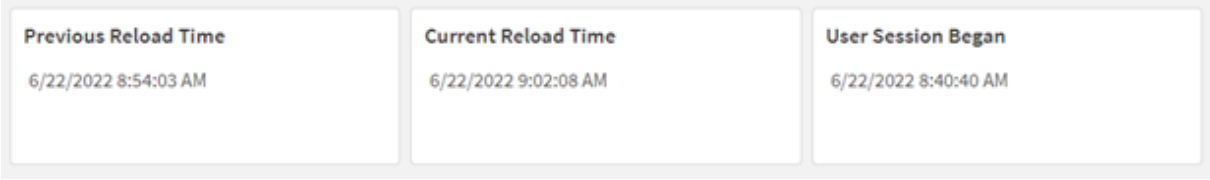
1. 使用**文本和图像**图表对象，创建一个文本框。
2. 将以下度量值添加到对象：
   =vCurrentDataLoad
3. 在**外观**下，选择 **Show titles** 并向对象添加标题“当前重新加载时间”。

创建最后一个文本框，以显示用户在应用程序中的会话何时启动。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 将以下度量值添加到对象：
   `=vApplicationOpened`
3. 在**外观**下，选择 **Show titles** 并向对象添加标题“用户会话已开始”。

now() 加载脚本变量



上图显示了每个已创建变量的示例值。例如，这些值可以如下所示：

- 上次重新加载时间：6/22/2022 8:54:03 AM
- 当前重新加载时间：6/22/2022 9:02:08 AM
- 用户会话开始时间：6/22/2022 8:40:40 AM

示例 2 –不使用加载脚本生成对象

加载脚本和图表表达式

概述

在本例中，您将使用 `now()` 函数创建三个图表对象，而无需将任何变量或数据加载到应用程序中。每个图表对象都使用其中一个 `timer_mode` 选项来演示其效果。

此示例没有加载脚本。

执行以下操作：

1. 打开数据加载编辑器。
2. 在不更改现有加载脚本的情况下，单击**加载数据**。
3. 短时间后，再次加载脚本。

结果

第二次加载数据后，创建三个文本框。

首先，为最新数据重新加载创建一个文本框。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 添加以下度量：
   `=now(0)`
3. 在**外观**下，选择**显示标题**并将标题“最新数据重新加载”添加到对象。

接下来，创建一个文本框以显示当前时间。

执行以下操作：

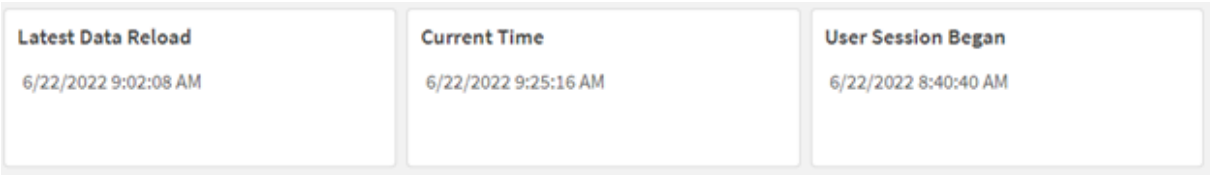
1. 使用**文本和图像**图表对象，创建一个文本框。
2. 添加以下度量：
   `=now(1)`
3. 在**外观**下，选择**显示标题**并向对象添加标题“当前时间”。

创建最后一个文本框，以显示用户在应用程序中的会话何时启动。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 添加以下度量：
   `=now(2)`
3. 在**外观**下，选择**显示标题**并向对象添加标题“用户会话开始”。

now() 图表对象示例



上图显示了每个已创建对象的示例值。例如，这些值可以如下所示：

- 最后重新加载数据时间：6/22/2022 9:02:08 AM
- 当前时间：6/22/2022 9:25:16 AM
- 用户会话开始时间：6/22/2022 8:40:40 AM

“最新数据重新加载”图表对象使用值为 0 的 `timer_mode`。这将返回上次成功重新加载数据的时间戳。

“当前时间”图表对象使用为 1 的 `timer_mode` 值。这将根据系统时钟返回当前时间。如果刷新了工作表或对象，则将更新此值。

“用户会话开始”图表对象使用值为 2 的 `timer_mode`。这将返回应用程序打开和用户会话开始的时间戳。

示例 3 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 由加密货币挖掘操作的库存组成的数据集，该数据集加载到名为 Inventory 的表中。
- 具有以下字段的数据：id、purchase_date 和 wph（瓦特/小时）。

用户想要一个表格，按 id 显示每个采矿平台在该月迄今为止的总成本（以功耗为单位）。

每当刷新图表对象时，此值应更新。目前的电力成本为每千瓦时 0.0678 美元。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Inventory:
Load
*
Inline
[
id,purchase_date,wph
8188,1/7/2022,1123
8189,1/19/2022,1432
8190,2/28/2022,1227
8191,2/5/2022,1322
8192,3/16/2022,1273
8193,4/1/2022,1123
8194,5/7/2022,1342
8195,5/16/2022,2342
8196,6/15/2022,1231
8197,6/26/2022,1231
8198,7/9/2022,1123
8199,7/22/2022,1212
8200,7/23/2022,1223
8201,7/27/2022,1232
8202,8/2/2022,1232
8203,8/8/2022,1211
8204,8/19/2022,1243
8205,9/26/2022,1322
8206,10/14/2022,1133
8207,10/29/2022,1231
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：id。

创建以下度量：

```
=(now(1)-monthstart(now(1)))*24*wph/1000*0.0678
```

如果图表对象在 2022 年 6 月 22 日上午 10:39:05 刷新，它将返回以下结果：

结果表

id	=(now(1)-monthstart(now(1)))*24*wph/1000*0.0678
8188	$39.18
8189	$49.97
8190	$42.81
8191	$46.13
8192	$44.42
8193	$39.18
8194	$46.83
8195	$81.72
8196	$42.95
8197	$42.95
8198	$39.18
8199	$42.29
8200	$42.67
8201	$42.99
8202	$42.99
8203	$42.25
8204	$43.37
8205	$46.13
8206	$39.53

用户希望每次刷新对象时都刷新对象结果。因此，为表达式中的 `now()` 函数实例提供 `timer_mode` 参数。通过将 `now()` 函数用作 `monthstart()` 函数中的时间戳参数，从 `now()` 函数标识的当前时间中减去月份开始的时间戳。这提供了本月迄今为止的总时间，以天为单位。

该值乘以 24(一天中的小时数)，然后乘以 `wph` 字段中的值。

将每小时瓦特转换为千瓦时，结果除以 1000，最后乘以所提供的千瓦时费率。

quarterend

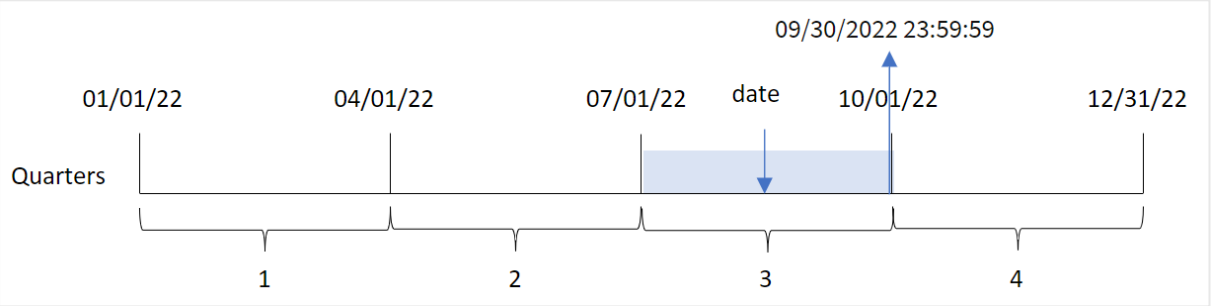
此函数用于返回与包含 **date** 的季度的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
QuarterEnd(date[, period_no[, first_month_of_year]])
```

返回数据类型：双

quarterend() 函数的图表



quarterend() 函数确定日期属于哪个季度。然后，它以日期格式返回该季度最后一个月的最后一毫秒的时间戳。默认情况下，一年的第一个月是一月。但是，您可以使用 quarterend() 函数中的 first_month_of_year 参数更改将哪个月设置为第一个月。

> *quarterend() 函数不考虑 FirstMonthOfYear 系统变量。这一年从 1 月 1 日开始，除非用 first_month_of_year 参数来改变它。*

适用场景

当您希望计算使用尚未发生的月份分数时，quarterend() 函数通常用作表达式的一部分。例如，您想计算一季度中尚未发生的利息总额。

参数

参数	说明
date	要评估的日期或时间戳。
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。

可以使用以下值在 first_month_of_year 参数中设置一年中的第一个月：

first_month_of_year 值

月	值
二月	2
三月	3
四月	4
五月	5
六月	6
七月	7
八月	8
九月	9
十月	10
十一月	11
十二月	12

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
`quarterend('10/29/2005')`	Returns 12/31/2005 23:59:59.
`quarterend('10/29/2005', -1)`	Returns 09/30/2005 23:59:59.
`quarterend('10/29/2005', 0, 3)`	Returns 11/30/2005 23:59:59.

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 包含以下内容的前置 Load：
  - 设置为 'end_of_quarter' 字段并返回交易发生时季度末的时间戳的 quarterend() 函数。
  - 设置为 'end_of_quarter_timestamp' 字段并返回所选季度结束的确切时间戳的 timestamp() 函数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       quarterend(date) as end_of_quarter,
       timestamp(quarterend(date)) as end_of_quarter_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- end_of_quarter
- end_of_quarter_timestamp

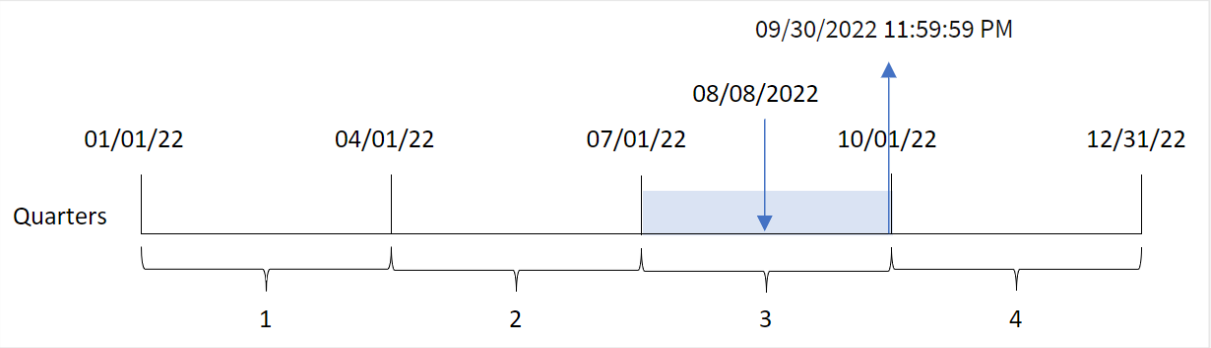
结果表

id	日期	end_of_quarter	end_of_quarter_timestamp
8188	1/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8189	1/19/2022	03/31/2022	3/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	06/30/2022	6/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/16/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	09/30/2022	9/30/2022 11:59:59 PM
8199	7/22/2022	09/30/2022	9/30/2022 11:59:59 PM
8200	7/23/2022	09/30/2022	9/30/2022 11:59:59 PM
8201	7/27/2022	09/30/2022	9/30/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	10/29/2022	12/31/2022	12/31/2022 11:59:59 PM

通过使用 quarterend() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'end_of_quarter' 字段。

quarterend() 函数最初确定日期值属于哪一季度，然后返回该季度最后一毫秒的时间戳。

已确定交易 8203 季度末的 quarterend() 函数图



交易 8203 发生在 8 月 8 日。quarterend() 函数标识交易发生在第三季度，并返回该季度的最后一毫秒，即 9 月 30 日下午 11:59:59。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 包含以下内容的前置 Load：
  - 设置为 'previous_quarter_end' 字段并返回交易发生前季度末的时间戳的 quarterend() 函数。
  - 设置为 'previous_end_of_quarter_timestamp' 字段并返回交易发生前季度末的确切时间戳的 timestamp() 函数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       quarterend(date, -1) as previous_quarter_end,
       timestamp(quarterend(date, -1)) as previous_quarter_end_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
```

```
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- previous_quarter_end
- previous_quarter_end_timestamp

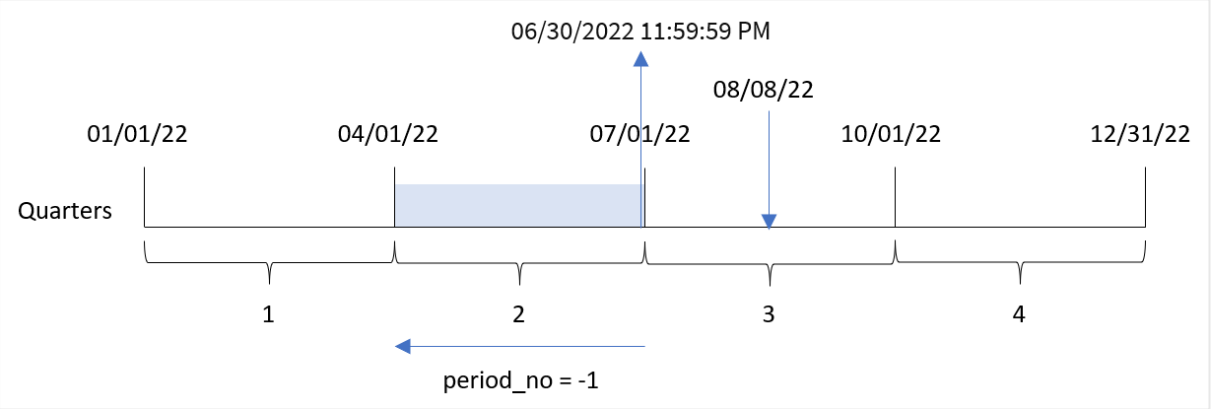
结果表

id	日期	previous_quarter_end	previous_quarter_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	12/31/2021	12/31/2021 11:59:59 PM
8191	2/28/2022	12/31/2021	12/31/2021 11:59:59 PM
8192	3/16/2022	12/31/2021	12/31/2021 11:59:59 PM
8193	4/1/2022	03/31/2022	3/31/2022 11:59:59 PM
8194	5/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8195	5/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8196	6/15/2022	03/31/2022	3/31/2022 11:59:59 PM
8197	6/26/2022	03/31/2022	3/31/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM

id	日期	previous_quarter_end	previous_quarter_end_timestamp
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	06/30/2022	6/30/2022 11:59:59 PM
8203	8/8/2022	06/30/2022	6/30/2022 11:59:59 PM
8204	8/19/2022	06/30/2022	6/30/2022 11:59:59 PM
8205	9/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8206	10/14/2022	09/30/2022	9/30/2022 11:59:59 PM
8207	10/29/2022	09/30/2022	9/30/2022 11:59:59 PM

由于 quarterend() 函数中使用了 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的季度。然后，它在一季度前移动，并确定该季度的最后一毫秒。

quarterend() 函数的图表，其中 period_no 为 -1.



交易 8203 发生于 8 月 8 日。quarterend() 函数确定交易发生前的一个季度是 4 月 1 日至 6 月 30 日。然后，函数返回该季度的最后一毫秒，即 6 月 30 日晚上 11:59:59。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 包含以下内容的前置 Load：
  - 设置为 'end_of_quarter' 字段并返回交易发生时季度末的时间戳的 quarterend() 函数。
  - 设置为 'end_of_quarter_timestamp' 字段并返回所选季度结束的确切时间戳的 timestamp() 函数。

然而，在本例中，公司政策是财政年度从 3 月 1 日开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       quarterend(date, 0, 3) as end_of_quarter,
       timestamp(quarterend(date, 0, 3)) as end_of_quarter_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

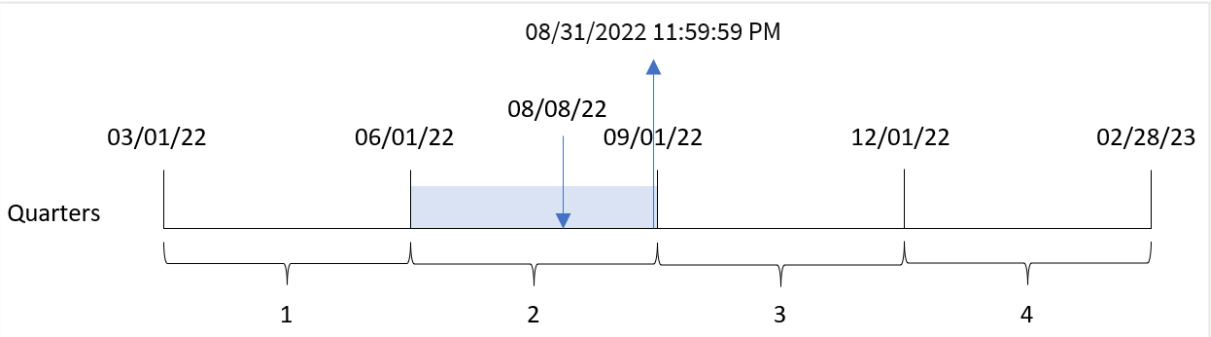
结果表

id	日期	end_of_quarter	end_of_quarter_timestamp
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	05/31/2022	5/31/2022 11:59:59 PM

id	日期	end_of_quarter	end_of_quarter_timestamp
8193	4/1/2022	05/31/2022	5/31/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	08/31/2022	8/31/2022 11:59:59 PM
8197	6/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	11/30/2022	11/30/2022 11:59:59 PM
8206	10/14/2022	11/30/2022	11/30/2022 11:59:59 PM
8207	10/29/2022	11/30/2022	11/30/2022 11:59:59 PM

因为 quarterend() 函数中使用了 3 的 first_month_of_year 参数，所以年初从 1 月 1 日移动到 3 月 1 日。

quarterend() 函数的图表，3 月为一年中的第一个月



交易 8203 发生在 8 月 8 日。由于年初是 3 月 1 日，因此该年的季度发生在 3 月至 5 月、6 月至 8 月、9 月 至 11 月和 12 月至 2 月之间。

quarterend() 函数确定交易发生在 6 月初到 8 月之间的季度，并返回该季度的最后一毫秒，即 8 月 31 日晚上 11:59:59。

示例 4 – 图表对象示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生时季度末时间戳的计算将在应用程序中的图表中创建为度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date

要计算交易发生季度的结束日期，请创建以下度量：

- =quarterend(date)
- =timestamp(quarterend(date))

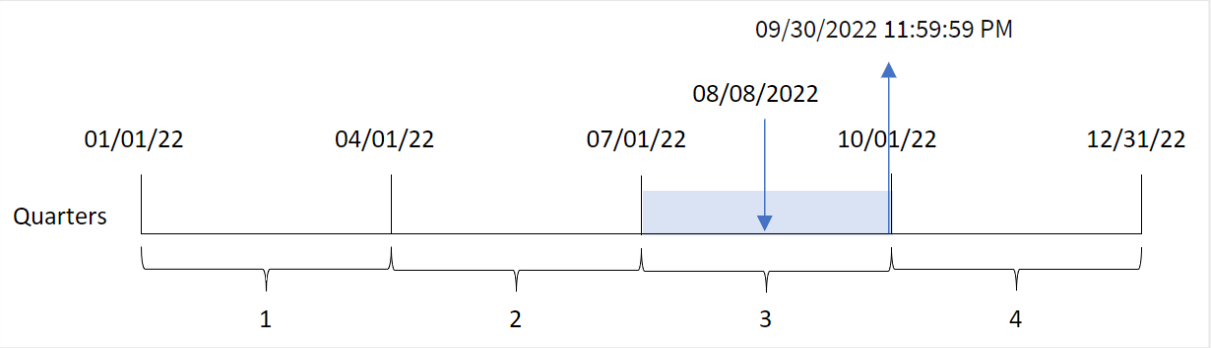
结果表

id	日期	=quarterend(date)	=timestamp(quarterend(date))
8188	1/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8189	1/19/2022	03/31/2022	3/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	06/30/2022	6/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/16/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	09/30/2022	9/30/2022 11:59:59 PM
8199	7/22/2022	09/30/2022	9/30/2022 11:59:59 PM
8200	7/23/2022	09/30/2022	9/30/2022 11:59:59 PM
8201	7/27/2022	09/30/2022	9/30/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	10/29/2022	12/31/2022	12/31/2022 11:59:59 PM

通过使用 quarterend() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'end_of_quarter' 字段。

quarterend() 函数最初确定日期值属于哪一季度，然后返回该季度最后一毫秒的时间戳。

已确定交易 8203 季度末的 quarterend() 函数图



交易 8203 发生在 8 月 8 日。quarterend() 函数标识交易发生在第三季度，并返回该季度的最后一毫秒，即 9 月 30 日下午 11:59:59。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 数据集加载到名为 'Employee_Expenses' 的表中。表格包含以下字段：
  - 员工 ID
  - 员工姓名
  - 每位员工的平均每日费用报销。

最终用户需要一个图表对象，该对象按员工 ID 和员工姓名显示该季度剩余时间仍将发生的估计费用索赔。财政年度从一月份开始。

加载脚本

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- employee_id
- employee_name

要计算累计利息，请创建以下度量：

- =(quarterend(today(1))-today(1))*avg_daily_claim

将度量的**数字格式**设置为**金额**。

结果表

EmployeeID	employee_name	=(quarterend(today(1))-today(1))*avg_daily_claim
182	Mark	$480.00
183	Deryck	$400.00
184	Dexter	$400.00
185	Sydney	$864.00
186	Agatha	$576.00

quarterend() 函数使用今天的日期作为唯一参数，并返回当前月份的结束日期。然后，它从年末日期中减去今天的日期，表达式返回本月剩余的天数。

然后将该值乘以每个员工的平均每日费用索赔，以计算每个员工在剩余季度预计提出的索赔的估计值。

quartername

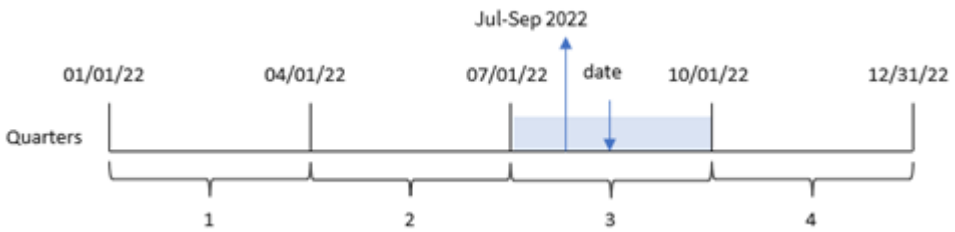
此函数用于返回一个显示值，该值显示季度的月（根据 **MonthNames** 脚本变量的格式）以及年，伴随一个与该季度第一天第一毫秒的时间戳对应的基础数值。

语法：

```
QuarterName(date[, period_no[, first_month_of_year]])
```

返回数据类型：双

quartername() 函数的图表



quartername() 函数确定日期属于哪个季度。然后它返回一个值，显示本季度的开始和结束月份以及年度。此结果的基本数值是季度的第一毫秒。

参数

参数	说明
date	要评估的日期或时间戳。
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。

适用场景

当您希望按季度比较聚合时，quartername() 函数非常有用。例如，如果您希望按季度查看产品的总销售额。

此函数可在加载脚本中用于在主日历表中创建字段。或者，它可以直接在图表中用作计算维度。

以下示例使用日期格式 MM/DD/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。

函数示例

示例	结果
quartername('10/29/2013')	返回 Oct-Dec 2013。
quartername('10/29/2013', -1)	返回 Jul-Sep 2013。
quartername('10/29/2013', 0, 3)	返回 Sep-Nov 2013。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数的日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 transaction_quarter，该字段返回季度发生前的季度末的时间戳。

根据需要在此处添加其他文本，包括列表等。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        quartername(date) as transaction_quarter
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- transaction_quarter

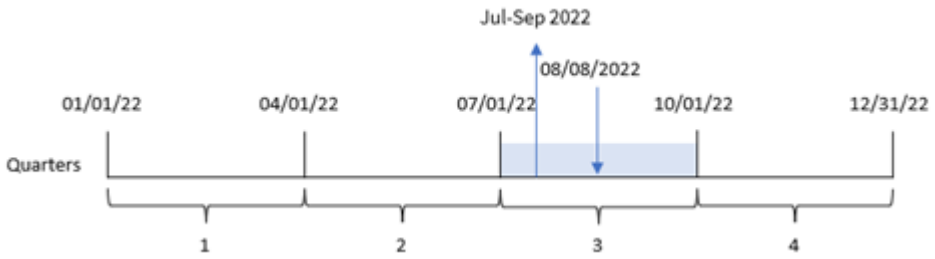
结果表

日期	transaction_quarter
1/7/2022	Jan-Mar 2022
1/19/2022	Jan-Mar 2022
2/5/2022	Jan-Mar 2022
2/28/2022	Jan-Mar 2022
3/16/2022	Jan-Mar 2022
4/1/2022	Apr-Jun 2022
5/7/2022	Apr-Jun 2022
5/16/2022	Apr-Jun 2022
6/15/2022	Apr-Jun 2022
6/26/2022	Apr-Jun 2022
7/9/2022	Jul-Sep 2022
7/22/2022	Jul-Sep 2022
7/23/2022	Jul-Sep 2022
7/27/2022	Jul-Sep 2022
8/2/2022	Jul-Sep 2022
8/8/2022	Jul-Sep 2022
8/19/2022	Jul-Sep 2022
9/26/2022	Jul-Sep 2022
10/14/2022	Oct-Dec 2022
10/29/2022	Oct-Dec 2022

通过使用 quartername() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 transaction_quarter 字段。

quartername() 函数最初标识日期值所在的季度。然后它返回一个值，显示本季度的开始和结束月份以及年度。

quartername() 函数图表，示例没有额外参数



交易 8203 发生于 2022 年 8 月 8 日。quartername() 函数识别交易发生在第三季度，因此返回 2022 年 7 月至 9 月。月份以与 MonthNames 系统变量相同的格式显示。

示例 2–带 period_no 参数的日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_quarter，该字段将上一季度返回到交易发生的时间。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        quartername(date,-1) as previous_quarter
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_quarter

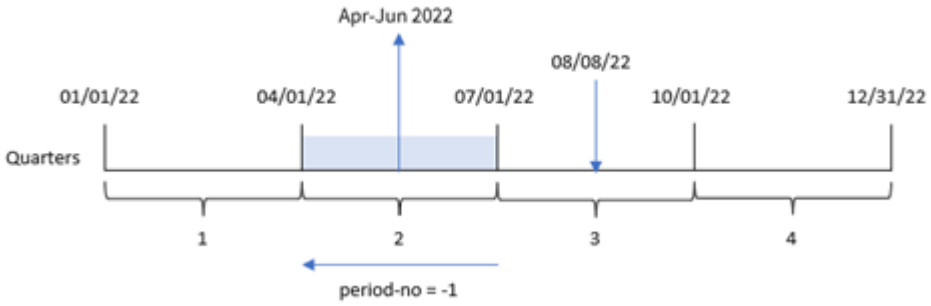
结果表

日期	previous_quarter
1/7/2022	Oct-Dec 2021
1/19/2022	Oct-Dec 2021
2/5/2022	Oct-Dec 2021
2/28/2022	Oct-Dec 2021
3/16/2022	Oct-Dec 2021
4/1/2022	Jan-Mar 2022
5/7/2022	Jan-Mar 2022
5/16/2022	Jan-Mar 2022
6/15/2022	Jan-Mar 2022
6/26/2022	Jan-Mar 2022
7/9/2022	Apr-Jun 2022
7/22/2022	Apr-Jun 2022
7/23/2022	Apr-Jun 2022
7/27/2022	Apr-Jun 2022
8/2/2022	Apr-Jun 2022
8/8/2022	Apr-Jun 2022
8/19/2022	Apr-Jun 2022
9/26/2022	Apr-Jun 2022

日期	previous_quarter
10/14/2022	Jul-Sep 2022
10/29/2022	Jul-Sep 2022

在本例中，由于 quartername() 函数中使用了为 -1 的 period_no 作为偏移参数，因此函数首先识别交易发生在第三季度。然后，它在前一个季度进行移位，并返回一个显示本季度开始和结束月份以及年度的值。

quartername() 函数的图表，period_no 示例



交易 8203 发生于 8 月 8 日。quartername() 函数确定交易发生前的一个季度是 4 月 1 日至 6 月 30 日。因此它返回 Apr-Jun 2022。

示例 3 – 带有 first_week_day 参数的日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在这个例子中，我们需要将 3 月 1 日定为财政年度的开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        quartername(date,0,3) as transaction_quarter
        ;
Load
*
Inline
[
id,date,amount
```

```
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `date`
- `transaction_quarter`

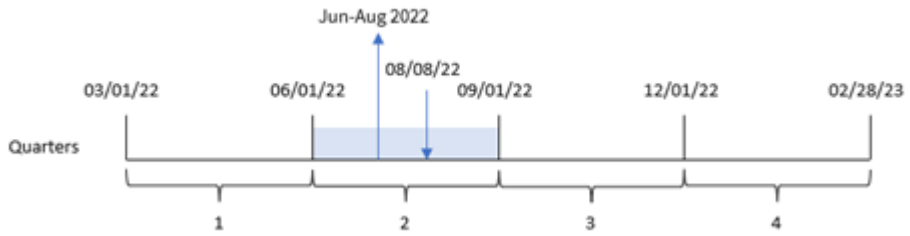
结果表

日期	transaction_quarter
1/7/2022	Dec-Feb 2021
1/19/2022	Dec-Feb 2021
2/5/2022	Dec-Feb 2021
2/28/2022	Dec-Feb 2021
3/16/2022	Mar-May 2022
4/1/2022	Mar-May 2022
5/7/2022	Mar-May 2022
5/16/2022	Mar-May 2022
6/15/2022	Jun-Aug 2022
6/26/2022	Jun-Aug 2022
7/9/2022	Jun-Aug 2022
7/22/2022	Jun-Aug 2022

日期	transaction_quarter
7/23/2022	Jun-Aug 2022
7/27/2022	Jun-Aug 2022
8/2/2022	Jun-Aug 2022
8/8/2022	Jun-Aug 2022
8/19/2022	Jun-Aug 2022
9/26/2022	Sep-Nov 2022
10/14/2022	Sep-Nov 2022
10/29/2022	Sep-Nov 2022

在该例子中，由于 quartername() 函数中使用了为 3 的参数 first_month_of_year，所以一年的开始时间从 1 月 1 日移动到 3 月 1 日。因此，一年中的季度分为 3 月至 5 月、6 月至 8 月、9 月至 11 月和 12 月 至 2 月。

quartername() 函数的图表，first_week_day 示例



交易 8203 发生在 8 月 8 日。quartername() 函数确定交易发生在第二季度，即 6 月初至 8 月底。因此它返回 Jun-Aug 2022。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生时季度末的时间戳的计算作为应用程序的图表对象中的度量创建。

加载脚本

```
Transactions:
Load
*
```

```
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：`date`。

创建以下度量：

```
=quartername(date)
```

结果表

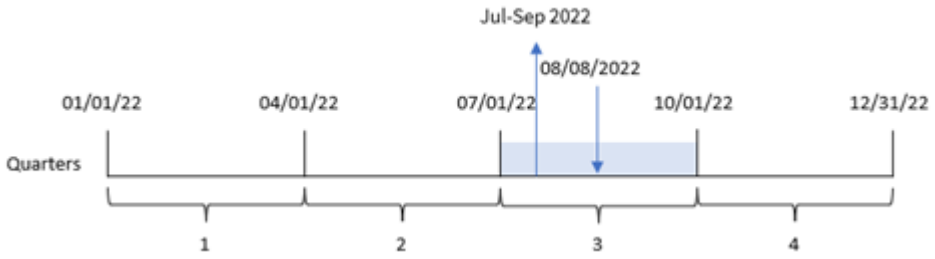
日期	=quartername(date)
1/7/2022	Jan-Mar 2022
1/19/2022	Jan-Mar 2022
2/5/2022	Jan-Mar 2022
2/28/2022	Jan-Mar 2022
3/16/2022	Jan-Mar 2022
4/1/2022	Apr-Jun 2022
5/7/2022	Apr-Jun 2022
5/16/2022	Apr-Jun 2022
6/15/2022	Apr-Jun 2022
6/26/2022	Apr-Jun 2022

日期	=quartername(date)
7/9/2022	Jul-Sep 2022
7/22/2022	Jul-Sep 2022
7/23/2022	Jul-Sep 2022
7/27/2022	Jul-Sep 2022
8/2/2022	Jul-Sep 2022
8/8/2022	Jul-Sep 2022
8/19/2022	Jul-Sep 2022
9/26/2022	Jul-Sep 2022
10/14/2022	Oct-Dec 2022
10/29/2022	Oct-Dec 2022

通过使用 quartername() 函数并将 date 字段作为函数的参数传递，在图表对象中创建 transaction_quarter 度量。

quartername() 函数最初标识日期值所在的季度。然后它返回一个值，显示本季度的开始和结束月份以及年度。

quartername() 函数的图表，图表对象示例



交易 8203 发生于 2022 年 8 月 8 日。quartername() 函数识别交易发生在第三季度，因此返回 2022 年 7 月至 9 月。月份以与 MonthNames 系统变量相同的格式显示。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

最终用户希望得到一个图表对象，该图表对象按季度显示交易的总销售额。即使该维度在数据模型中不可用，也可以使用 quartername() 函数作为图表中的计算维度来实现这点。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/7/2022',17.17
8189,'1/19/2022',37.23
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 使用以下表达式创建计算维度：
   `=quartername(date)`
3. 接下来使用以下聚合度量计算总销售额：
   `=sum(amount)`
4. 将度量的**数字格式**设置为**金额**。

结果表

=quartername(date)	=sum(amount)
Jul-Sep 2022	$446.31
Apr-Jun 2022	$351.48
Jan-Mar 2022	$253.89
Oct-Dec 2022	$163.91

quarterstart

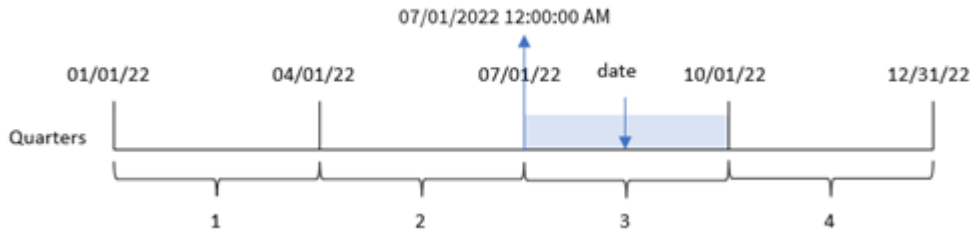
此函数用于返回与包含 **date** 的季度的第一毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
QuarterStart(date[, period_no[, first_month_of_year]])
```

返回数据类型：双

quarterstart() 函数的图表



quarterstart() 函数确定 date 属于哪个季度。然后，它以日期格式返回该季度第一个月的第一毫秒的时间戳。

参数

参数	说明
date	要评估的日期或时间戳。
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。

适用场景

当用户希望计算使用到目前为止已过的一季度的部分时，quarterstart() 函数通常用作表达式的一部分。例如，如果用户想计算一个季度迄今为止累计的利息，可以使用它。

函数示例

示例	结果
quarterstart('10/29/2005')	返回 10/01/2005。
quarterstart('10/29/2005', -1)	返回 07/01/2005。
quarterstart('10/29/2005', 0, 3)	返回 09/01/2005。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建字段 start_of_quarter，其返回交易发生的季度开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        quarterstart(date) as start_of_quarter,
        timestamp(quarterstart(date)) as start_of_quarter_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
```

```
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- start_of_quarter
- start_of_quarter_timestamp

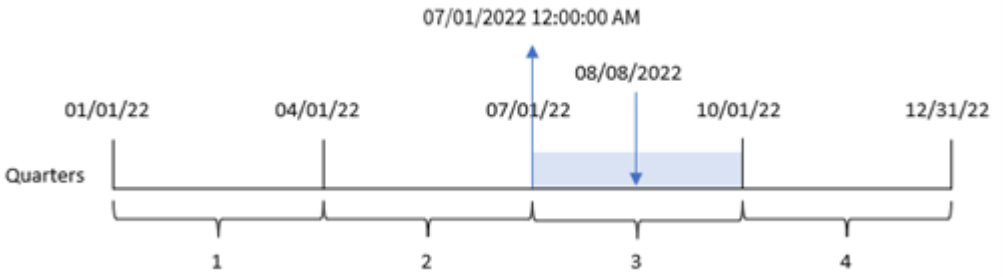
结果表

日期	start_of_quarter	start_of_quarter_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	01/01/2022	1/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2021 12:00:00 AM
5/7/2022	04/01/2022	4/1/2021 12:00:00 AM
5/16/2022	04/01/2022	4/1/2021 12:00:00 AM
6/15/2022	04/01/2022	4/1/2021 12:00:00 AM
6/26/2022	04/01/2022	4/1/2021 12:00:00 AM
7/9/2022	07/01/2022	7/1/2021 12:00:00 AM
7/22/2022	07/01/2022	7/1/2021 12:00:00 AM

日期	start_of_quarter	start_of_quarter_timestamp
7/23/2022	07/01/2022	7/1/2021 12:00:00 AM
7/27/2022	07/01/2022	7/1/2021 12:00:00 AM
8/2/2022	07/01/2022	7/1/2021 12:00:00 AM
8/8/2022	07/01/2022	7/1/2021 12:00:00 AM
8/19/2022	07/01/2022	7/1/2021 12:00:00 AM
9/26/2022	07/01/2022	7/1/2021 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

通过使用 quarterstart() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 start_of_quarter 字段。uarterstart() 函数最初标识日期值属于哪个季度。然后，它返回该季度第一毫秒的时间戳。

quarterstart() 函数图表，示例没有额外参数



交易 8203 发生在 8 月 8 日。quarterstart() f函数标识交易发生在第三季度，并返回该季度的第一毫秒，即 7 月 1 日中午 12:00:00。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_quarter_start，该字段返回季度发生前的季度开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        quarterstart(date,-1) as previous_quarter_start,
        timestamp(quarterstart(date,-1)) as previous_quarter_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_quarter_start
- previous_quarter_start_timestamp

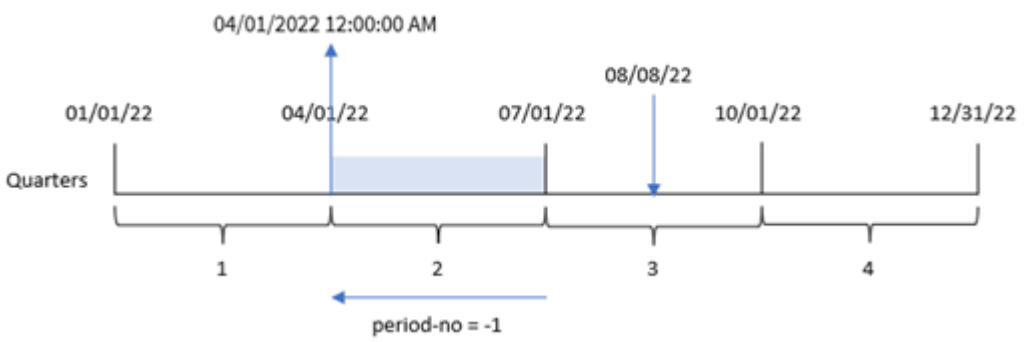
结果表

日期	previous_quarter_start	previous_quarter_start_timestamp
1/7/2022	10/01/2021	10/1/2021 12:00:00 AM
1/19/2022	10/01/2021	10/1/2021 12:00:00 AM

日期	previous_quarter_start	previous_quarter_start_timestamp
2/5/2022	10/01/2021	10/1/2021 12:00:00 AM
2/28/2022	10/01/2021	10/1/2021 12:00:00 AM
3/16/2022	10/01/2021	10/1/2021 12:00:00 AM
4/1/2022	01/01/2022	1/1/2022 12:00:00 AM
5/7/2022	01/01/2022	1/1/2022 12:00:00 AM
5/16/2022	01/01/2022	1/1/2022 12:00:00 AM
6/15/2022	01/01/2022	1/1/2022 12:00:00 AM
6/26/2022	01/01/2022	1/1/2022 12:00:00 AM
7/9/2022	04/01/2022	4/1/2021 12:00:00 AM
7/22/2022	04/01/2022	4/1/2021 12:00:00 AM
7/23/2022	04/01/2022	4/1/2021 12:00:00 AM
7/27/2022	04/01/2022	4/1/2021 12:00:00 AM
8/2/2022	04/01/2022	4/1/2021 12:00:00 AM
8/8/2022	04/01/2022	4/1/2021 12:00:00 AM
8/19/2022	04/01/2022	4/1/2021 12:00:00 AM
9/26/2022	04/01/2022	4/1/2021 12:00:00 AM
10/14/2022	07/01/2022	7/1/2022 12:00:00 AM
10/29/2022	07/01/2022	7/1/2022 12:00:00 AM

在本例中，由于 quarterstart() 函数中使用了为 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的季度。然后，它在一季度前移动，并确定该季度的第一毫秒。

quarterstart() 函数的图表，period_no 示例



交易 8203 发生于 8 月 8 日。quarterstart() 函数确定交易发生前的一个季度是 4 月 1 日至 6 月 30 日。然后返回该季度的第一毫秒，即 4 月 1 日中午 12:00:00。

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在这个例子中，我们需要将 3 月 1 日定为财政年度的开始。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        quarterstart(date,0,3) as start_of_quarter,
        timestamp(quarterstart(date,0,3)) as start_of_quarter_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

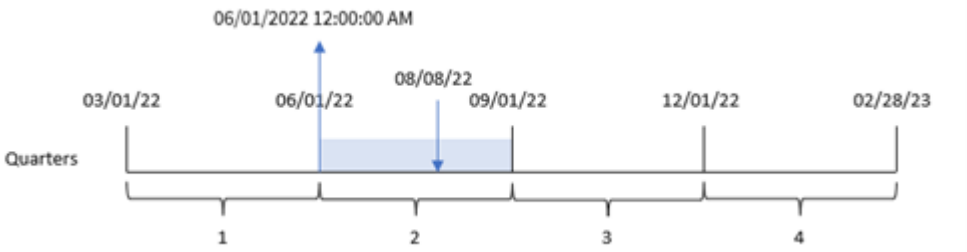
- date
- start_of_quarter
- start_of_quarter_timestamp

结果表

日期	start_of_quarter	start_of_quarter_timestamp
1/7/2022	12/01/2021	12/1/2021 12:00:00 AM
1/19/2022	12/01/2021	12/1/2021 12:00:00 AM
2/5/2022	12/01/2021	12/1/2021 12:00:00 AM
2/28/2022	12/01/2021	12/1/2021 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
4/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	03/01/2022	3/1/2022 12:00:00 AM
5/16/2022	03/01/2022	3/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	06/01/2022	6/1/2022 12:00:00 AM
8/8/2022	06/01/2022	6/1/2022 12:00:00 AM
8/19/2022	06/01/2022	6/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

在该例中，因为 quarterstart() 函数中使用了为 3 的 first_month_of_year 参数，所以年初从 1 月 1 日移动到 3 月 1 日。

quarterstart() 函数 first_month_of_year 示例的图表



交易 8203 发生在 8 月 8 日。由于年初是 3 月 1 日，因此一年中的季度发生在 3 月至 5 月、6 月至 8 月、9 月至 11 月和 12 月至 2 月之间。quarterstart() 函数确定交易发生在 6 月初到 8 月之间的季度，并返回该季度的第一毫秒，即 6 月 1 日中午 12:00:00。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生时季度末的时间戳的计算作为应用程序的图表对象中的度量创建。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
```

```
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

添加以下度量：

- =quarterstart(date)
- =timestamp(quarterstart(date))

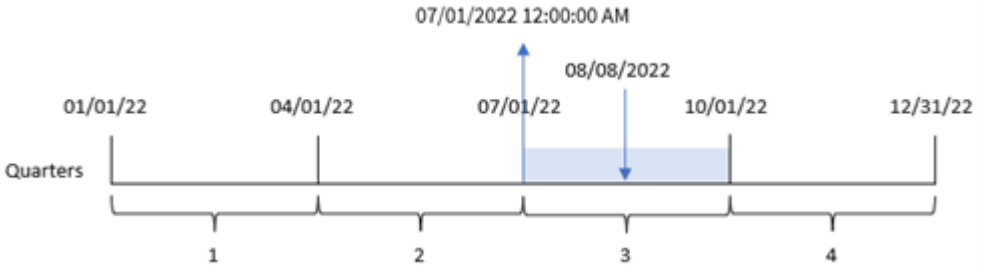
结果表

日期	=quarterstart(date)	=timestamp(quarterstart(date))
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	07/01/2022	7/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/16/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	04/01/2022	4/1/2022 12:00:00 AM
6/26/2022	04/01/2022	4/1/2022 12:00:00 AM
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	01/01/2022	1/1/2022 12:00:00 AM

通过使用 quarterstart() 函数并将 date 字段作为函数的参数传递，在图表对象中创建 start_of_quarter 度量。

quarterstart() 函数标识日期值所属的季度，并返回该季度第一毫秒的时间戳。

quarterstart() 函数的图表，图表对象示例



交易 8203 发生在 8 月 8 日。quarterstart() f函数标识交易发生在第三季度，并返回该季度的第一毫秒。此返回值为 7 月 1 日中午 12:00:00。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含一组贷款余额的数据集，该数据集加载到名为 Loans 的表中。
- 数据包括贷款 ID、本季度初余额和每年每笔贷款收取的简单利率。

最终用户希望有一个图表对象，该对象按贷款 ID 显示季度初至今每笔贷款的应计利息。

加载脚本

```
Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将这些字段添加为维度：
   - loan_id
   - start_balance
2. 接下来，创建该度量来计算累计利息：
   `=start_balance*(rate*(today(1)-quarterstart(today(1)))/365)`
3. 将度量的**数字格式**设置为**金额**。

结果表

loan_id	start_balance	=start_balance*(rate*(today(1)-quarterstart(today(1)))/365)
8188	$10000.00	$15.07
8189	$15000.00	$128.84
8190	$17500.00	$63.29
8191	$21000.00	$107.59
8192	$90000.00	$1139.18

quarterstart() 函数使用今天的日期作为唯一参数，返回当前年份的开始日期。通过从当前日期中减去该结果，表达式将返回季度迄今为止经过的天数。

然后将该值乘以利率并除以 365，以返回该期间产生的实际利率。然后将结果乘以贷款的起始余额，以返回本季度迄今为止累计的利息。

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

适用场景

当您希望按秒比较聚合时，second() 函数非常有用。例如，如果希望按秒查看活动计数分布，可以使用该函数。

这些维度可以在加载脚本中创建，方法是使用函数在主日历表中创建字段，也可以直接在图表中用作计算维度。

函数示例

示例	结果
second('09:14:36')	返回 36
second('0.5555')	返回 55(因为 0.5555 = 13:19:55)

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 变量

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 按时间戳包含交易的数据集，加载到名为 Transactions 的表格中。
- 使用默认 TimeStamp 系统变量 (M/D/YYYY h:mm:ss[.fff] TT)。
- 创建字段 second，以计算购买发生的时间。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        second(date) as second
        ;
Load
*
Inline
[
id,date,amount
9497,'01/05/2022 7:04:57 PM',47.25
9498,'01/03/2022 2:21:53 PM',51.75
9499,'01/03/2022 5:40:49 AM',73.53
```

```
9500,'01/04/2022 6:49:38 PM',15.35
9501,'01/01/2022 10:10:22 PM',31.43
9502,'01/05/2022 7:34:46 PM',13.24
9503,'01/06/2022 10:58:34 PM',74.34
9504,'01/06/2022 11:29:38 AM',50.00
9505,'01/02/2022 8:35:54 AM',36.34
9506,'01/06/2022 8:49:09 AM',74.23
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- second

结果表

日期	second
01/01/2022 10:10:22 PM	22
01/02/2022 8:35:54 AM	54
01/03/2022 5:40:49 AM	49
01/03/2022 2:21:53 PM	53
01/04/2022 6:49:38 PM	38
01/05/2022 7:04:57 PM	57
01/05/2022 7:34:46 PM	46
01/06/2022 8:49:09 AM	9
01/06/2022 11:29:38 AM	38
01/06/2022 10:58:34 PM	34

second 字段中的值是通过使用 second() 函数并将日期作为前置 Load 语句中的表达式传递来创建的。

示例 2 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。second 通过图表对象中的度量计算值。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
*
Inline
[
id,date,amount
9497,'01/05/2022 7:04:57 PM',47.25
9498,'01/03/2022 2:21:53 PM',51.75
9499,'01/03/2022 5:40:49 AM',73.53
9500,'01/04/2022 6:49:38 PM',15.35
9501,'01/01/2022 10:10:22 PM',31.43
9502,'01/05/2022 7:34:46 PM',13.24
9503,'01/06/2022 10:58:34 PM',74.34
9504,'01/06/2022 11:29:38 AM',50.00
9505,'01/02/2022 8:35:54 AM',36.34
9506,'01/06/2022 8:49:09 AM',74.23
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

创建以下度量：

```
=second(date)
```

结果表

日期	=second(date)
01/01/2022 10:10:22 PM	22
01/02/2022 8:35:54 AM	54
01/03/2022 5:40:49 AM	49
01/03/2022 2:21:53 PM	53
01/04/2022 6:49:38 PM	38
01/05/2022 7:04:57 PM	57
01/05/2022 7:34:46 PM	46
01/06/2022 8:49:09 AM	9
01/06/2022 11:29:38 AM	38
01/06/2022 10:58:34 PM	34

second 的值是通过使用 second() 函数并将日期作为表达式传递给图表对象的度量来创建的。

示例 3 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 时间戳数据集，用于表示特定节日门票销售网站的流量。这些时间戳和相应的 id被加载到名为 Web_Traffic 的表中。
- 使用了 TimeStamp 系统变量 M/D/YYYY h:mm:ss[.fff] TT。

在这种情况下，共有 10000 张门票，于 2021 年 5 月 20 日上午 9:00 开始发售。一分钟后，票卖完了。

用户想要一个图表对象，以秒为单位显示网站的访问次数。

加载脚本

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

tmpTimeStampCreator:
load
    makedate(2022,05,20) as date
AutoGenerate 1;

join load
    maketime(9+floor(rand()*2),0,floor(rand()*59)) as time
autogenerate 10000;

Web_Traffic:
load
    recno() as id,
    timestamp(date + time) as timestamp
resident tmpTimeStampCreator;

drop table tmpTimeStampCreator;
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 接下来，使用以下表达式创建计算尺寸：
   `=second(timestamp)`
3. 创建聚合度量值以计算条目总数：
   `=count(id)`

结果表与下表相似，但聚合度量不同：

结果表

second(timestamp)	=count(id)
0	150
1	184
2	163
3	178
4	179
5	158
6	177
7	169
8	149
9	186
10	169
11	179
12	186
13	182
14	180
15	153
16	191
17	203
18	158
19	159
20	163
+ 39 更多行	

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

此函数返回在 Qlik 引擎运行的计算机上定义的时区。

语法：

```
TimeZone( )
```

返回数据类型：双

示例：

```
timezone( )
```

如果您想在应用程序中的度量中看到不同的时区，可以在度量中使用 `localtime()` 函数。

today

此函数用于返回当前日期。函数以 `DateFormat` 系统变量格式返回值。

语法：

```
today([ timer_mode])
```

返回数据类型： 双

today() 函数可以在加载脚本或图表对象中使用。

默认 timer_mode 值为 1。

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。*

函数示例

timer_mode 值	结果(如果在加载脚本中使用)	在图表对象中使用时的结果
0	以 DateFormat 系统变量格式返回最近一次数据重新加载之前最后一次成功数据重新加载的日期。	以 DateFormat 系统变量格式返回最新数据重新加载的日期。
1	以 DateFormat 系统变量格式返回最新数据重新加载的日期。	以 DateFormat 系统变量格式返回函数调用的日期。
2	以 DateFormat 系统变量格式返回应用程序中用户会话开始的日期。除非用户重新加载脚本，否则不会更新该脚本。	以 DateFormat 系统变量格式返回应用程序中用户会话开始的日期。一旦新会话开始或重新加载应用程序中的数据，将刷新此属性。

适用场景

today() 函数通常用作表达式中的组件。例如，它可用于计算截至当前日期的一个月内累积的利息。

下表提供了 today() 函数返回的结果的解释，给出了 timer_mode 参数的不同值：

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这

些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 使用加载脚本生成对象

加载脚本和结果

概述

以下示例使用 `today()` 函数创建三个变量。每个变量都使用其中一个 `timer_mode` 选项来演示其效果。

为了演示变量的用途，请重新加载脚本，然后在 24 小时后再次重新加载脚本。这将导致 `today(0)` 和 `today(1)` 变量显示不同的值，从而正确地显示它们的用途。

加载脚本

```
LET vPreviousDataLoad = today(0);
LET vCurrentDataLoad = today(1);
LET vApplicationOpened = today(2);
```

结果

第二次加载数据后，使用下面的说明创建三个文本框。

首先，为先前加载的数据创建一个文本框。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 将以下度量值添加到对象：
   `=vPreviousDataLoad`
3. 在**外观**下，选择 **Show titles** 并向对象添加标题“上次重新加载时间”。

接下来，为当前正在加载的数据创建一个文本框。

执行以下操作：

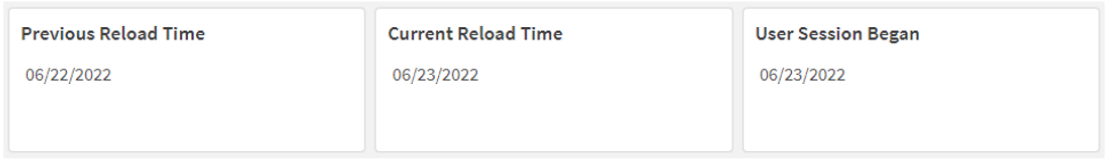
1. 使用**文本和图像**图表对象，创建一个文本框。
2. 将以下度量值添加到对象：
   `=vCurrentDataLoad`
3. 在**外观**下，选择 **Show titles** 并向对象添加标题“当前重新加载时间”。

创建最后一个文本框，以显示用户在应用程序中的会话何时启动。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 将以下度量值添加到对象：
   `=vApplicationOpened`
3. 在**外观**下，选择 **Show titles** 并向对象添加标题“用户会话已开始”。

使用加载脚本中的 today() 函数创建的变量图



上图显示了每个已创建变量的示例值。例如，这些值可以如下所示：

- 上次重新加载时间：06/22/2022
- 当前重新加载时间：06/23/2022
- 用户会话开始：06/23/2022

示例 2 –不使用加载脚本生成对象

加载脚本和图表表达式

概述

以下示例使用 `today()` 函数创建三个图表对象。每个图表对象都使用其中一个 `timer_mode` 选项来演示其效果。

此示例没有加载脚本。

结果

第二次加载数据后，创建三个文本框。

首先，为最新数据重新加载创建一个文本框。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 添加以下度量：
   `=today(0)`
3. 在**外观**下，选择**显示标题**并将标题“最新数据重新加载”添加到对象。

接下来，创建一个文本框以显示当前时间。

执行以下操作：

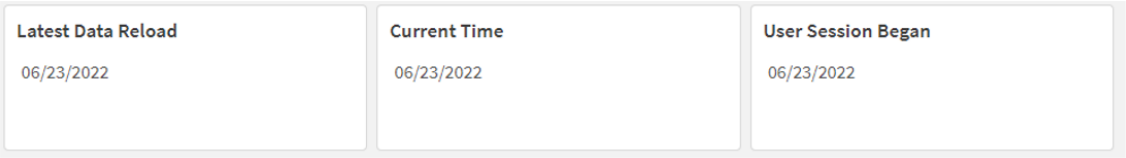
1. 使用**文本和图像**图表对象，创建一个文本框。
2. 添加以下度量：
   `=today(1)`
3. 在**外观**下，选择**显示标题**并向对象添加标题“当前时间”。

创建最后一个文本框，以显示用户在应用程序中的会话何时启动。

执行以下操作：

1. 使用**文本和图像**图表对象，创建一个文本框。
2. 添加以下度量：
   `=today(2)`
3. 在**外观**下，选择**显示标题**并向对象添加标题“用户会话开始”。

使用无加载脚本的 today() 函数创建的对象的图表



上图显示了每个已创建对象的示例值。例如，这些值可以如下所示：

- 最后数据重新加载时间：06/23/2022
- 当前时间：06/23/2022
- 用户会话开始：06/23/2022

“最新数据重新加载”图表对象使用值为 0 的 `timer_mode`。这将返回上次成功重新加载数据的时间戳。

“当前时间”图表对象使用为 1 的 `timer_mode` 值。这将根据系统时钟返回当前时间。如果刷新了工作表或对象，则将更新此值。

“用户会话开始”图表对象使用值为 2 的 `timer_mode`。这将返回应用程序打开和用户会话开始的时间戳。

示例 3 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含一组贷款余额的数据集，该数据集加载到名为 Loans 的表中。
- 包含贷款 ID、月初余额和每年每笔贷款的简单利率字段的表格数据。

最终用户希望有一个图表对象，该对象按贷款 ID 显示月初至今每笔贷款的应计利息。尽管应用程序每周只重新加载一次，但用户希望在刷新对象或应用程序时刷新结果。

加载脚本

```
Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 添加以下字段作为维度：
    - loan_id
    - start_balance
3. 接下来，创建一个度量来计算累计利息：
   `=start_balance*(rate*(today(1)-monthstart(today(1)))/365)`
4. 将度量的**数字格式**设置为**金额**。

结果表

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8188	$10000.00	$16.44
8189	$15000.00	$58.56
8190	$17500.00	$28.77
8191	$21000.00	$48.90
8192	$90000.00	$517.81

monthstart() 函数使用 today() 函数返回今天的日期作为唯一参数，返回当前月份的开始日期。通过从当前日期中减去该结果，再次使用 today() 函数，该表达式返回本月迄今为止已过的天数。

然后将该值乘以利率并除以 365，以返回该期间产生的实际利率。然后将结果乘以贷款的起始余额，以返回本月迄今为止累计的利息。

由于表达式内的 today() 函数中使用值 1 作为 timer_mode 参数，因此每次刷新图表对象时（通过打开应用程序、刷新页面、在工作表之间移动等），返回的日期将为当前日期，结果将相应刷新。

UTC

用于返回当前 Coordinated Universal Time。

语法：

```
UTC( )
```

返回数据类型： 双

示例：

```
utc( )
```

week

此函数返回一个整数，表示与输入的日期对应的周数。

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	要评估的日期或时间戳。
first_week_day	指定一周的开始日期。如果忽略，使用 **FirstWeekDay** 变量的值。 可能的值 **first_week_day** 为：周一为 0，周二为 1，周三为 2，周四为3，周五为 4，周六为 5，星期日为 6。 有关系统变量的详细信息，请参见*FirstWeekDay (page 218)*。
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。
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。

week() 函数确定日期属于哪一周，并返回周数。

在 Qlik Sense 中，创建应用程序时获取区域设置，相应的设置作为环境变量存储在脚本中。这些用于确定周数。

这意味着大多数欧洲应用程序开发人员都会获得以下环境变量，与 ISO 8601 定义相对应：

```
Set FirstWeekDay =0;    // Monday as first week day
Set BrokenWeeks  =0;    // Use unbroken weeks
Set ReferenceDay =4;    // Jan 4th is always in week 1
```

北美应用程序开发人员通常会获得以下环境变量：

```
Set FirstWeekDay =6;    // Sunday as first week day
Set BrokenWeeks  =1;    // Use broken weeks
Set ReferenceDay =1;    // Jan 1st is always in week 1
```

一周的第一天由 FirstWeekDay系统变量确定。您还可以使用 week() 函数中的 first_week_day 参数更改将哪个月设置为第一个月。

如果您的应用程序使用中断周，则周数计数从 1 月 1 日开始，并在 FirstWeekDay 系统变量前一天结束，而不管发生了多少天。

如果您的应用程序使用的是连续的周，则第 1 周可以从上一年开始，也可以从 1 月的前几天开始。这取决于如何使用 FirstWeekDay 和 ReferenceDay 环境变量。

适用场景

当您希望按周比较聚合时，The week() 函数非常有用。例如，如果您想查看每周产品的总销售额，则可以使用它。当用户希望计算不一定使用应用程序的 BrokenWeeks、FirstWeekDay 或 ReferenceDay 系统变量时，会选择函数 week() 而不是 weekname()。

例如，如果您想查看每周产品的总销售额。

如果应用程序使用不间断的周，则第 1 周可能包含上一年 12 月的日期，也可能不包括本年 1 月的日期。如果应用程序使用的是中断周，则第 1 周可能少于 7 天。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

以下示例假设

```
Set DateFormat= 'MM/DD/YYYY';
Set FirstWeekDay=0;
Set BrokenWeeks=0;
Set ReferenceDay=4;
```

函数示例

示例	结果
week('12/28/2021')	返回 52。
week(44614)	返回 8, 因为这是 2022 年 2 月 22 日的序列号。
week('01/03/2021')	返回 53。
week('01/03/2021',6)	返回 1。

示例 1 – 默认系统变量

加载脚本和结果

概述

打开数据加载编辑器, 并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 包含 2021 最后一周和 2022 年前两周的一组事务的数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建字段 week_number, 返回交易发生时的年和周数字。
- 创建一个名为 week_day 的字段, 显示每个交易日期的工作日值。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
    Load
        *,
        WeekDay(date) as week_day,
        Week(date) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
```

```
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- week_day
- week_number

结果表

id	日期	week_day	week_number
8183	12/27/2021	Mon	53
8184	12/28/2021	Tue	53
8185	12/29/2021	Wed	53
8186	12/30/2021	Thu	53
8187	12/31/2021	Fri	53
8188	01/01/2022	Sat	1
8189	01/02/2022	Sun	2
8190	01/03/2022	Mon	2
8191	01/04/2022	Tue	2
8192	01/05/2022	Wed	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	Sun	3
8197	01/10/2022	Mon	3

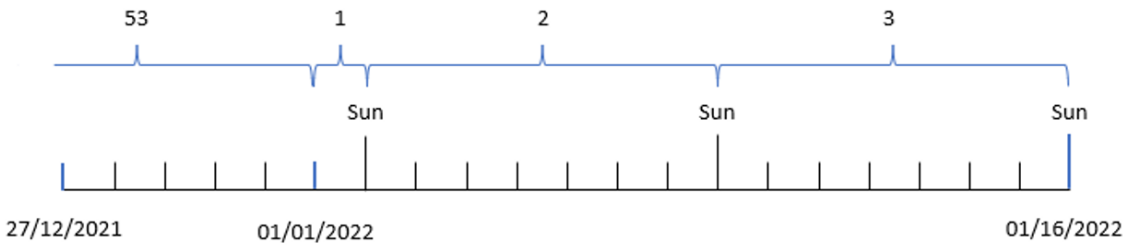
id	日期	week_day	week_number
8198	01/11/2022	Tue	3
8199	01/12/2022	Wed	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

通过使用 week() 函数并将 date 字段作为函数的参数传递，在前置 Load 语句中创建了 week_number 字段。

没有其他参数传递到函数中，因此影响 week() 函数的以下默认变量有效：

- BrokenWeeks：周计数从 1 月 1 日开始
- FirstWeekDay：一周的第一天是星期天

使用默认系统变量的 week() 函数图表



因为应用程序正在使用默认 BrokenWeeks 系统变量，所以第一周从 1 月 1 日（星期六）开始。

由于默认的 FirstWeekDay 系统变量，周在星期天开始。1 月 1 日之后的第一个星期天发生在 1 月 2 日，也就是第 2 周开始的时候。

示例 2 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 创建字段 week_number，返回交易发生时的年和周数字。
- 创建一个名为 week_day 的字段，显示每个交易日期的工作日值。

在本例中，我们希望将工作周的开始时间设置为星期二。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
    Load
        *,
        WeekDay(date) as week_day,
        Week(date,1) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

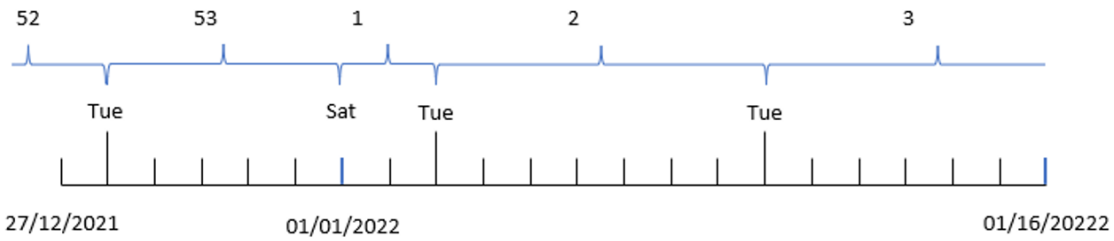
- id
- date
- week_day
- week_number

结果表

id	日期	week_day	week_number
8183	12/27/2021	Mon	52
8184	12/28/2021	Tue	53
8185	12/29/2021	Wed	53
8186	12/30/2021	Thu	53
8187	12/31/2021	Fri	53
8188	01/01/2022	Sat	1
8189	01/02/2022	Sun	1
8190	01/03/2022	Mon	1
8191	01/04/2022	Tue	2
8192	01/05/2022	Wed	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	Sun	2
8197	01/10/2022	Mon	2
8198	01/11/2022	Tue	3
8199	01/12/2022	Wed	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

应用程序仍在使用中断周。但是，week()函数中的 first_week_day 参数已设置为 1。这将一周的第一天设置为星期二。

week() 函数的图表，first_week_day 示例



应用程序正在使用默认 BrokenWeeks 系统变量，所以第一周从 1 月 1 日（星期六）开始。

week() 函数的 first_week_day 参数将第一个工作日设置为星期二。因此，第 53 周从 2021 年 12 月 28 日开始。

但是，由于该函数仍在使用中断周，第 1 周将仅为两天，因为 1 月 1 日之后的第一个星期二发生在 1 月 3 日。

示例 3 – unbroken_weeks

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

在本例中，我们使用非中断周。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
    Load
        *,
        WeekDay(date) as week_day,
        Week(date,6,0) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
```

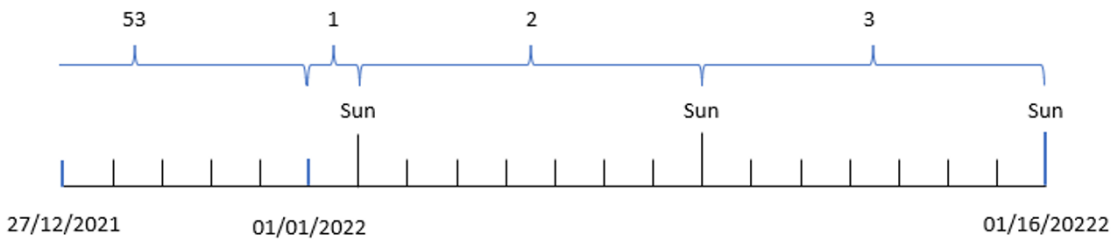
```
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- week_day
- week_number

week() 函数的图表，图表对象示例



结果表

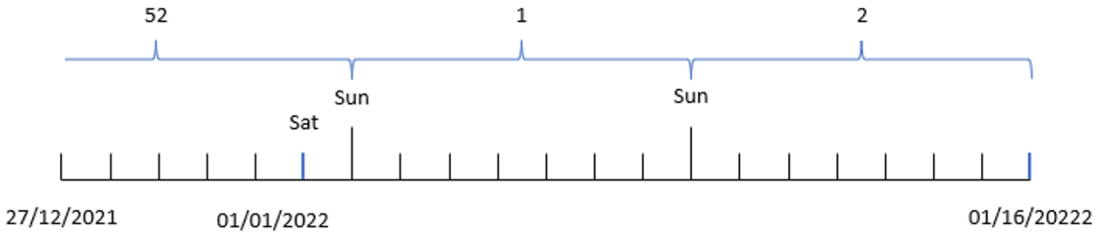
id	日期	week_day	week_number
8183	12/27/2021	Mon	52
8184	12/28/2021	Tue	52
8185	12/29/2021	Wed	52
8186	12/30/2021	Thu	52
8187	12/31/2021	Fri	52
8188	01/01/2022	Sat	52
8189	01/02/2022	Sun	1
8190	01/03/2022	Mon	1
8191	01/04/2022	Tue	1
8192	01/05/2022	Wed	1
8193	01/06/2022	Thu	1
8194	01/07/2022	Fri	1
8195	01/08/2022	Sat	1

id	日期	week_day	week_number
8196	01/09/2022	Sun	2
8197	01/10/2022	Mon	2
8198	01/11/2022	Tue	2
8199	01/12/2022	Wed	2
8200	01/13/2022	Thu	2
8201	01/14/2022	Fri	2

first_week_date 参数设置为 1，使星期二成为一周的第一天。broken_weeks 参数设置为 0，强制函数使用非中断周。最后，第三个参数将 reference_day 设置为 2。

first_week_date 参数设置为 6，使星期日成为一周的第一天。broken_weeks 参数设置为 0，强制函数使用非中断周。

week() 函数图，使用非中断周的示例



通过使用连续周，第一周不一定从 1 月 1 日开始；相反，它要求至少有四天。因此，在数据集中，第 52 周将于 2022 年 1 月 1 日星期六结束。然后，第 1 周从 FirstWeekDay 系统变量开始，即 1 月 2 日星期日。本周将在下一个星期六 1 月 8 日结束。

示例 4 – reference_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第三个示例相同的数据集和场景。
- 创建字段 week_number，返回交易发生时的年和周数字。
- 创建一个名为 week_day 的字段，显示每个交易日期的工作日值。

此外，必须满足以下条件：

- 工作周从星期二开始。
- 公司使用非中断周。
- reference_day 值为 2。换句话说，第 1 周 1 月份的最小天数为 2。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
    Load
        *,
        WeekDay(date) as week_day,
        Week(date,1,0,2) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

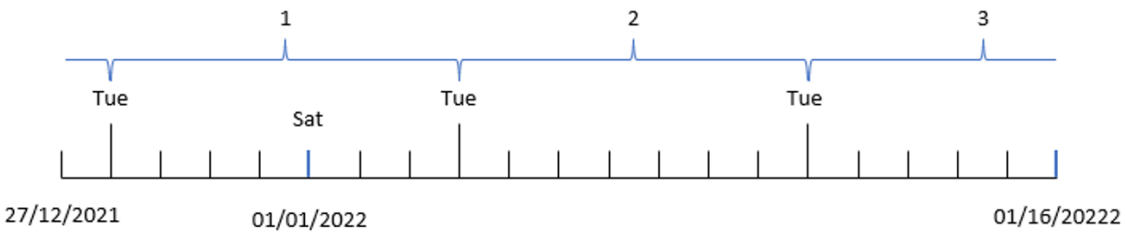
- id
- date
- week_day
- week_number

结果表

id	日期	week_day	week_number
8183	12/27/2021	Mon	52
8184	12/28/2021	Tue	1
8185	12/29/2021	Wed	1
8186	12/30/2021	Thu	1
8187	12/31/2021	Fri	1
8188	01/01/2022	Sat	1
8189	01/02/2022	Sun	1
8190	01/03/2022	Mon	1
8191	01/04/2022	Tue	2
8192	01/05/2022	Wed	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	Sun	2
8197	01/10/2022	Mon	2
8198	01/11/2022	Tue	3
8199	01/12/2022	Wed	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

`first_week_date` 参数设置为 1，使星期二成为一周的第一天。`broken_weeks` 参数设置为 0，强制函数使用非中断周。最后，第三个参数将 `reference_day` 参数设置为2。

week() 函数的图表，reference_day 示例



该函数使用连续的周数，并将 `reference_day` 值 2 用作参数，因此第 1 周只需要包括 1 月份的两天。由于第一个工作日是星期二，第一周从 2021 年 12 月 28 日开始，到 2022 年 1 月 3 日星期一结束。

示例 5 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回周数的计算在图表对象中创建为度量。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 添加以下字段作为维度：
    - id
    - date

3. 接下来，创建以下度量：
   `=week (date)`
4. 创建度量，`week_day`，以显示每个交易日期的工作日值：
   `=weekday(date)`

结果表

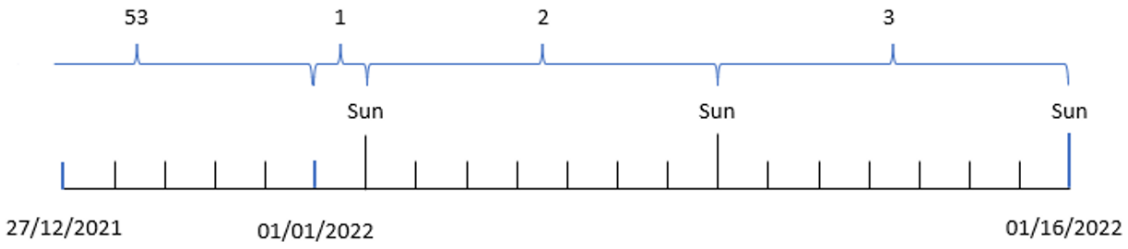
id	日期	=week(date)	=weekday(date)
8183	12/27/2021	53	Mon
8184	12/28/2021	53	Tue
8185	12/29/2021	53	Wed
8186	12/30/2021	53	Thu
8187	12/31/2021	53	Fri
8188	01/01/2022	1	Sat
8189	01/02/2022	2	Sun
8190	01/03/2022	2	Mon
8191	01/04/2022	2	Tue
8192	01/05/2022	2	Wed
8193	01/06/2022	2	Thu
8194	01/07/2022	2	Fri
8195	01/08/2022	2	Sat
8196	01/09/2022	3	Sun
8197	01/10/2022	3	Mon
8198	01/11/2022	3	Tue
8199	01/12/2022	3	Wed
8200	01/13/2022	3	Thu
8201	01/14/2022	3	Fri

通过使用 `week()` 函数并将 `date` 字段作为函数的参数传递，在前置 Load 语句中创建了 `week_number` 字段。

没有其他参数传递到函数中，因此影响 `week()` 函数的以下默认变量有效：

- `BrokenWeeks`：周计数从 1 月 1 日开始
- `FirstWeekDay`：一周的第一天是星期天

week() 函数的图表，图表对象示例



因为应用程序正在使用默认 BrokenWeeks 系统变量，所以第一周从 1 月 1 日（星期六）开始。

由于默认的 FirstWeekDay 系统变量，周在星期天开始。1 月 1 日之后的第一个星期天发生在 1 月 2 日，也就是第 2 周开始的时候。

示例 6 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2019 最后一周和 2020 年前两周的一组事务的数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

该应用程序主要使用仪表板上的休息周。但是，最终用户想要一个图表对象，它使用连续的周来显示每周的总销售额。参考日应为 1 月 2 日，星期二开始。即使该维度在数据模型中不可用，也可以使用 week() 函数作为图表中的计算维度来实现这点。

加载脚本

```
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2019,58.27
8184,12/28/2019,67.42
8185,12/29/2019,23.80
8186,12/30/2019,82.06
```

```
8187,12/31/2019,40.56
8188,01/01/2020,37.23
8189,01/02/2020,17.17
8190,01/03/2020,88.27
8191,01/04/2020,57.42
8192,01/05/2020,53.80
8193,01/06/2020,82.06
8194,01/07/2020,40.56
8195,01/08/2020,53.67
8196,01/09/2020,26.63
8197,01/10/2020,72.48
8198,01/11/2020,18.37
8199,01/12/2020,45.26
8200,01/13/2020,58.23
8201,01/14/2020,18.52
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。新建表格。
2. 创建以下计算维度：
   `=week(date)`
3. 接下来，创建以下聚合度量：
   `=sum(amount)`
4. 将度量的**数字格式**设置为**金额**。
5. 选择**排序**菜单，并删除计算维度的自定义排序。
6. 取消选择**按数字排序**和**按字母排序**选项。

结果表

week(date)	sum(amount)
52	$125.69
53	$146.42
1	$200.09
2	$347.57
3	$122.01

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。

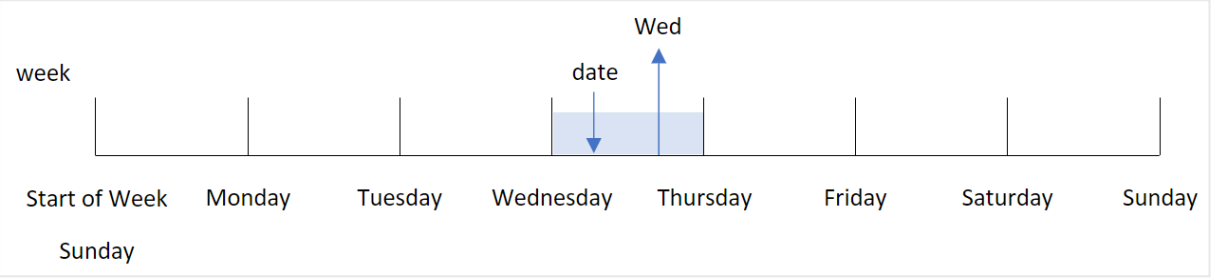
语法：

```
weekday(date [,first_week_day=0])
```

返回数据类型：双

weekday() 函数确定日期发生在一周中的哪一天。然后，它返回表示当天的字符串值。

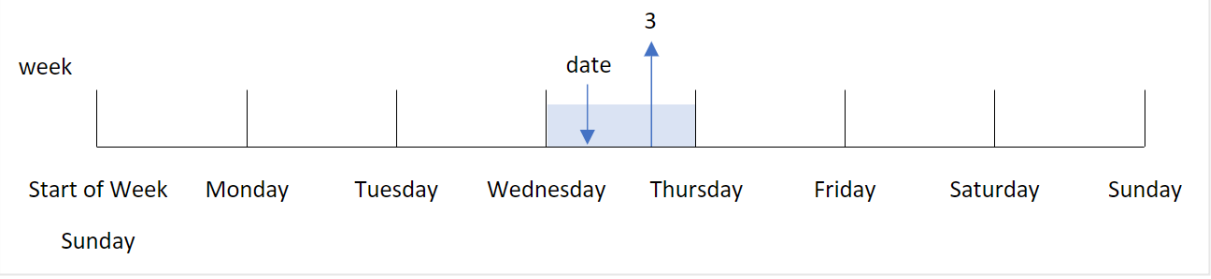
返回日期所在日期名称的 weekday() 函数图表



结果根据一周的开始日期返回与一周的那一天 (0-6) 对应的数值。例如，如果一周的第一天设置为星期天，星期三将返回 3 的数值。该开始日期由 FirstWeekDay 系统变量或 first_week_day 函数参数确定。

可以将此数值用作算术表达式的一部分。例如，将其乘以 1 以返回值本身。

显示日期数值而不是日期名称的函数 weekday() 的图表



适用场景

当您希望按星期几比较聚合时，weekday() 函数非常有用。例如，如果您想比较平日产品的平均销售额。

可以使用函数在**主日历**表中创建字段，在加载脚本中创建这些维度；或者直接在图表中创建作为计算的度量。

相关主题

主题	交互
FirstWeekDay (page 218)	定义每周的开始日期。

参数

参数	说明
date	要评估的日期或时间戳。
first_week_day	指定一周的开始日期。如果忽略，使用 **FirstWeekDay** 变量的值。 *FirstWeekDay (page 218)*

可以使用以下值在 `first_week_day` 参数中设置一周开始的日期：

first_week_day 值

日	值
星期一	0
星期二	1
星期三	2
星期四	3
星期五	4
星期六	5
星期日	6

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

在以下示例中，`FirstWeekDay` 设置为 `0`(除非另有说明)。

函数示例

示例	结果
`weekday('10/12/1971')`	返回 “Tue” 和 1。
`weekday('10/12/1971' , 6)`	返回 “Tue” 和 2。 在此示例中，Sunday (6) 作为一周的第一天。

示例	结果
SET FirstWeekDay=6; ... weekday('10/12/1971')	返回 “Tue” 和 2。

示例 1 - 工作日字符串

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 ‘Transactions’ 的表中。
- 设置为 6（星期日）的 FirstWeekDay 系统变量。
- 设置为使用默认日期名称的 DayNames 变量。
- 包含 weekday() 函数的前置 Load，该函数设置为 ‘week_day’ 字段，并返回事务发生的工作日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
    Load
       *,
       WeekDay(date) as week_day
       ;
Load
*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- week_day

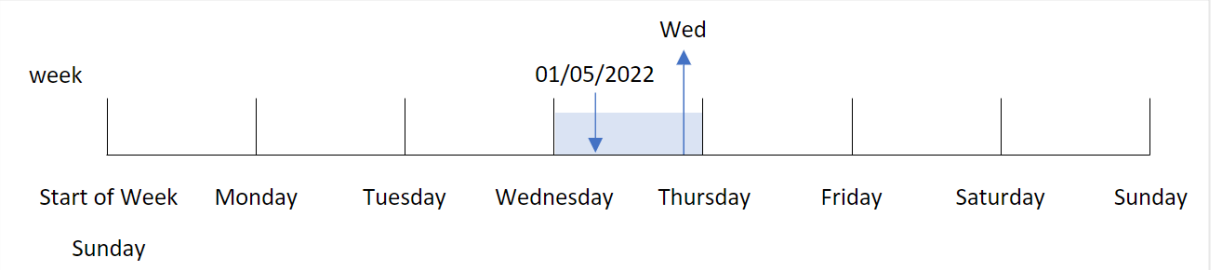
结果表

id	日期	week_day
8188	01/01/2022	Sat
8189	01/02/2022	Sun
8190	01/03/2022	Mon
8191	01/04/2022	Tue
8192	01/05/2022	Wed
8193	01/06/2022	Thu
8194	01/07/2022	Fri

通过使用 weekday() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'week_day' 字段。

weekday() 函数返回工作日字符串值；也就是说，它返回由 DayNames 系统变量设置的工作日的名称。

返回周三作为交易 8192 的工作日的 weekday() 函数的图表



交易 8192 发生在 1 月 5 日。FirstWeekDay 系统变量将一周的第一天设置为星期天。weekday() 函数交易发生在星期三，并在 week_day 字段中以 DayNames 系统变量的缩写形式返回该值。

'week_day' 字段中的值在列中右对齐，因为该字段有双数字和文本结果（周三，3）。要将字段值转换为等效的数字，可以将字段包装在 num() 函数中。例如，在交易 8192 中，周三值将转换为数字 3。

示例 2 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 设置为 6(星期日) 的 FirstWeekDay 系统变量。
- 设置为使用默认日期名称的 DayNames 变量。
- 包含 weekday() 函数的前置 Load，该函数设置为 'week_day' 字段，并返回事务发生的工作日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
    Load
       *,
       WeekDay(date,1) as week_day
       ;
Load
*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];
```

结果

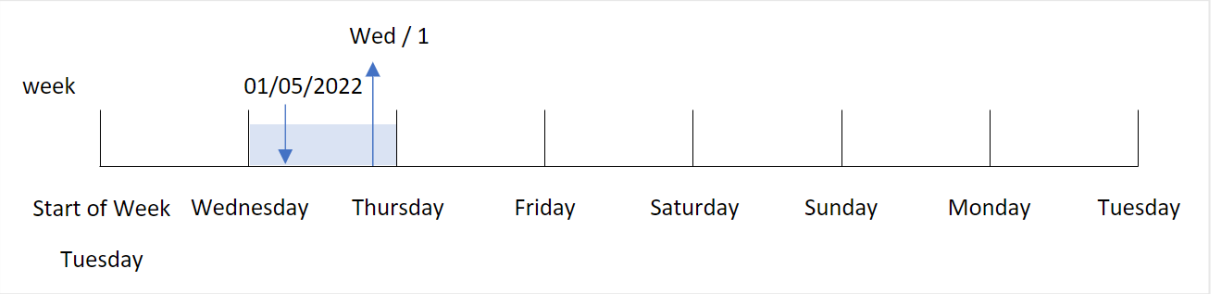
加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- week_day

结果表

id	日期	week_day
8188	01/01/2022	Sat
8189	01/02/2022	Sun
8190	01/03/2022	Mon
8191	01/04/2022	Tue
8192	01/05/2022	Wed
8193	01/06/2022	Thu
8194	01/07/2022	Fri

周三显示的 weekday() 函数图表具有双数值 1



因为 first_week_day 参数在 weekday() 函数中设置为 1，所以一周的第一天是星期二。因此，周二发生的所有交易都将具有双数值 0。

交易 8192 发生在 1 月 5 日。weekday()函数标识这是星期三，因此表达式将返回双数值 1。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 设置为 6(星期日) 的 FirstWeekDay 系统变量。
- 设置为使用默认日期名称的 DayNames 变量。

然而，在本例中，未更改的数据集被加载到应用程序中。标识工作日值的计算在应用程序的图表中创建为度量。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
Load
*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
```

```
8194,01/07/2022,40.39
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date

要计算工作日值，请创建以下度量：

- =weekday(date)

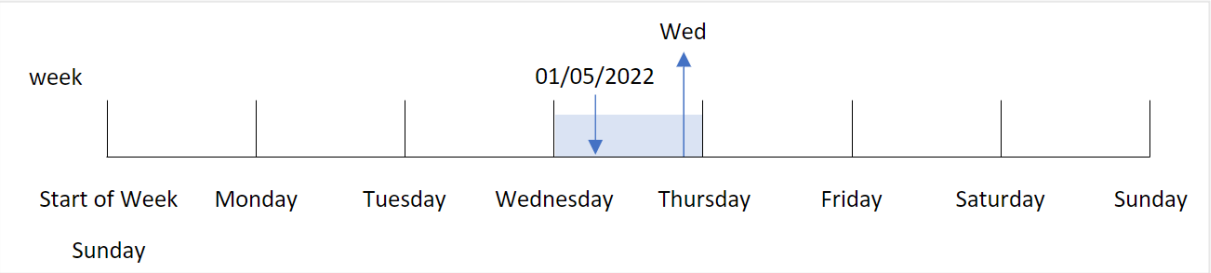
结果表

id	日期	=weekday(date)
8188	01/01/2022	Sat
8189	01/02/2022	Sun
8190	01/03/2022	Mon
8191	01/04/2022	Tue
8192	01/05/2022	Wed
8193	01/06/2022	Thu
8194	01/07/2022	Fri

通过使用 weekday() 函数并将日期字段作为函数的参数传递，在图表中创建了 '=weekday(date)' 字段。

weekday() 函数返回工作日字符串值；也就是说，它返回由 DayNames 系统变量设置的工作日的名称。

返回周三作为交易 8192 的工作日的 weekday() 函数的图表



交易 8192 发生在 1 月 5 日。FirstWeekDay 系统变量将一周的第一天设置为星期天。weekday() 函数交易发生在星期三，并在 =weekday(date) 字段中以 DayNames 系统变量的缩写形式返回该值。

示例 4 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 'Transactions' 的表中。
- 设置为 6（星期日）的 FirstWeekDay 系统变量。
- 设置为使用默认日期名称的 DayNames 变量。

最终用户希望得到一个图表，该图表按工作日显示交易的平均销售额。

加载脚本

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
LOAD
  RecNo() AS id,
  MakeDate(2022, 1, Ceil(Rand() * 31)) as date,
  Rand() * 1000 AS amount

Autogenerate(1000);
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- =weekday(date)
- =avg(amount)

将度量的**数字格式**设置为**金额**。

结果表

weekday(date)	Avg(amount)
Sun	$536.96
Mon	$500.80
Tue	$515.63
Wed	$509.21
Thu	$482.70

weekday(date)	Avg(amount)
Fri	$441.33
Sat	$505.22

weekend

此函数返回一个值，该值对应于包含 **date** 的日历周的最后一天的最后一毫秒的时间戳。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

```
WeekEnd(timestamp [, period_no [, first_week_day ]])
```

返回数据类型：双

weekend() 函数确定日期属于哪个周。然后以日期格式返回该周最后一毫秒的时间戳。一周的第一天由 FirstWeekDay 环境变量确定。但是，这可以被 weekend() 函数中的 first_week_day 参数取代。

参数

参数	说明
timestamp	要评估的日期或时间戳。
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。 **first_week_day** 可能的值为：周一为 0，周二为 1，周三为 2，周四为3，周五为 4，周六为 5，星期日为 6。 有关系统变量的详细信息，请参见 *FirstWeekDay (page 218)*

适用场景

当用户希望计算使用指定日期的一周剩余天数时，weekend() 函数通常用作表达式的一部分。例如，如果用户想计算一周内尚未产生的总利息，则可以使用该方法。

以下示例假设：

```
SET FirstWeekDay=0;
```

示例	结果
weekend('01/10/2013')	返回 01/12/2013 23:59:59。
weekend('01/10/2013', -1)	返回 01/05/2013 23:59:59.。
weekend('01/10/2013', 0, 1)	返回 01/14/2013 23:59:59。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例：

如果您想要周和周数的 ISO 设置，请确保脚本中包含以下内容：

```
Set DateFormat   ='YYYY-MM-DD';
Set FirstWeekDay =0;    // Monday as first week day
Set BrokenWeeks  =0;    //(use unbroken weeks)
Set ReferenceDay =4;    // Jan 4th is always in week 1
```

如果需要 US 设置，请确保脚本中包含以下内容：

```
Set DateFormat   ='M/D/YYYY';
Set FirstWeekDay =6;    // Sunday as first week day
Set BrokenWeeks  =1;    //(use broken weeks)
Set ReferenceDay =1;    // Jan 1st is always in week 1
```

以上示例从 weekend() 函数中得出以下结果：

Weekend 函数示例

日期	ISO 周末	US 周末
2020 年 12 月 26 日星期六	2020-12-27	12/26/2020
2020 年 12 月 27 日星期日	2020-12-27	1/2/2021
2020 年 12 月 28 日星期一	2021-01-03	1/2/2021
2020 年 12 月 29 日星期二	2021-01-03	1/2/2021
2020 年 12 月 30 日星期三	2021-01-03	1/2/2021
2020 年 12 月 31 日星期四	2021-01-03	1/2/2021
2021 年 1 月 1 日星期五	2021-01-03	1/2/2021
2021 年 1 月 2 日周六	2021-01-03	1/2/2021
2021 年 1 月 3 日星期日	2021-01-03	1/9/2021
2021 年 1 月 4 日星期一	2021-01-10	1/9/2021
2021 年 1 月 5 日星期二	2021-01-10	1/9/2021

ISO 列中的周末为星期日，US 列中的周末为周六。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建字段 end_of_week，返回交易发生的周结束的时间戳。

加载脚本

```
SET FirstWeekDay=6;

Transactions:
    Load
        *,
        weekend(date) as end_of_week,
        timestamp(weekend(date)) as end_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
```

```
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- end_of_week
- end_of_week_timestamp

结果表

日期	end_of_week	end_of_week_timestamp
1/7/2022	01/08/2022	1/8/2022 11:59:59 PM
1/19/2022	01/22/2022	1/22/2022 11:59:59 PM
2/5/2022	02/05/2022	2/5/2022 11:59:59 PM
2/28/2022	03/05/2022	3/5/2022 11:59:59 PM
3/16/2022	03/19/2022	3/19/2022 11:59:59 PM
4/1/2022	04/02/2022	4/2/2022 11:59:59 PM
5/7/2022	05/07/2022	5/7/2022 11:59:59 PM
5/16/2022	05/21/2022	5/21/2022 11:59:59 PM
6/15/2022	06/18/2022	6/18/2022 11:59:59 PM
6/26/2022	07/02/2022	7/2/2022 11:59:59 PM
7/9/2022	07/09/2022	7/9/2022 11:59:59 PM
7/22/2022	07/23/2022	7/23/2022 11:59:59 PM
7/23/2022	07/23/2022	7/23/2022 11:59:59 PM
7/27/2022	07/30/2022	7/30/2022 11:59:59 PM
8/2/2022	08/06/2022	8/6/2022 11:59:59 PM
8/8/2022	08/13/2022	8/13/2022 11:59:59 PM
8/19/2022	08/20/2022	8/20/2022 11:59:59 PM
9/26/2022	10/01/2022	10/1/2022 11:59:59 PM
10/14/2022	10/15/2022	10/15/2022 11:59:59 PM
10/29/2022	10/29/2022	10/29/2022 11:59:59 PM

通过使用 weekend() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 end_of_week 字段。

weekend() 函数标识日期值属于哪一周，并返回该周最后一毫秒的时间戳。

weekend() 函数的图表，基本示例

交易 8191 发生在 2 月 5 日。FirstWeekDay 系统变量将一周的第一天设置为星期日。weekend() 函数确定 2 月 5 日之后的第一个星期六，即本周的最后一个星期六是2月5号。因此，该事务的 end_of_week 值将返回当天的最后一毫秒，即二月 5 日午夜 11:59:59。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_week_end，该字段返回事务发生前的周开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        weekend(date,-1) as previous_week_end,
        timestamp(weekend(date,-1)) as previous_week_end_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
```

```
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- previous_week_end
- previous_week_end_timestamp

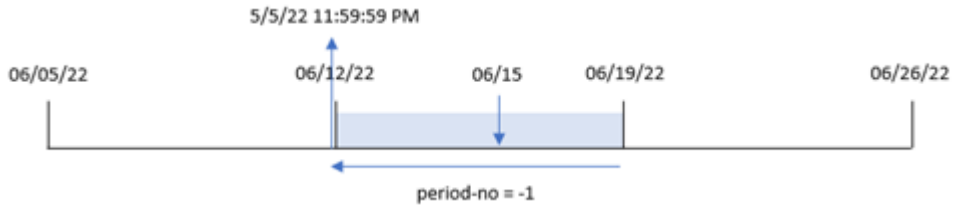
结果表

日期	end_of_week	end_of_week_timestamp
1/7/2022	01/01/2022	1/1/2022 11:59:59 PM
1/19/2022	01/15/2022	1/15/2022 11:59:59 PM
2/5/2022	01/29/2022	1/29/2022 11:59:59 PM
2/28/2022	02/26/2022	2/26/2022 11:59:59 PM
3/16/2022	03/12/2022	3/12/2022 11:59:59 PM
4/1/2022	03/26/2022	3/26/2022 11:59:59 PM
5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
5/16/2022	05/14/2022	5/14/2022 11:59:59 PM
6/15/2022	06/11/2022	6/11/2022 11:59:59 PM
6/26/2022	06/25/2022	6/25/2022 11:59:59 PM
7/9/2022	07/02/2022	7/2/2022 11:59:59 PM
7/22/2022	07/16/2022	7/16/2022 11:59:59 PM
7/23/2022	07/16/2022	7/16/2022 11:59:59 PM
7/27/2022	07/23/2022	7/23/2022 11:59:59 PM
8/2/2022	07/30/2022	7/30/2022 11:59:59 PM
8/8/2022	08/06/2022	8/6/2022 11:59:59 PM
8/19/2022	08/13/2022	8/13/2022 11:59:59 PM
9/26/2022	09/24/2022	9/24/2022 11:59:59 PM

日期	end_of_week	end_of_week_timestamp
10/14/2022	10/08/2022	10/8/2022 11:59:59 PM
10/29/2022	10/22/2022	10/22/2022 11:59:59 PM

在本例中，由于 weekend() 函数中使用了为 -1 的 period_no 作为偏移量参数，因此该函数首先标识交易发生的周。然后，它查找前一周，并确定该周的最后一毫秒。

weekend() 函数的图表，period_no 示例



交易 8196 发生在 6 月 15 日。weekend() 函数确定周从 6 月 12 日开始。因此，前一周于 6 月 11 日午夜 11:59:59 结束；这是为 previous_week_end 字段返回的值。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。但是，在本例中，我们需要将星期二设置为工作周的第一天。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        weekend(date,0,1) as end_of_week,
        timestamp(weekend(date,0,1)) as end_of_week_timestamp,
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
```

```
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- end_of_week
- end_of_week_timestamp

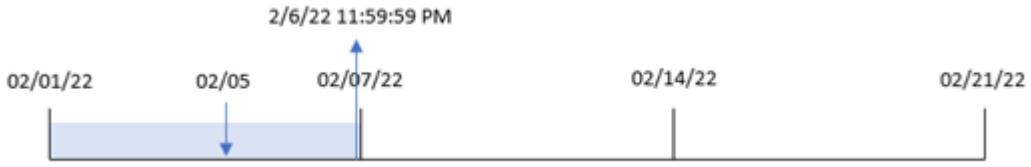
结果表

日期	end_of_week	end_of_week_timestamp
1/7/2022	01/10/2022	1/10/2022 11:59:59 PM
1/19/2022	01/24/2022	1/24/2022 11:59:59 PM
2/5/2022	02/07/2022	2/7/2022 11:59:59 PM
2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
3/16/2022	03/21/2022	3/21/2022 11:59:59 PM
4/1/2022	04/04/2022	4/4/2022 11:59:59 PM
5/7/2022	05/09/2022	5/9/2022 11:59:59 PM
5/16/2022	05/16/2022	5/16/2022 11:59:59 PM
6/15/2022	06/20/2022	6/20/2022 11:59:59 PM
6/26/2022	06/27/2022	6/27/2022 11:59:59 PM
7/9/2022	07/11/2022	7/11/2022 11:59:59 PM
7/22/2022	07/25/2022	7/25/2022 11:59:59 PM
7/23/2022	07/25/2022	7/25/2022 11:59:59 PM
7/27/2022	08/01/2022	8/1/2022 11:59:59 PM

日期	end_of_week	end_of_week_timestamp
8/2/2022	08/08/2022	8/8/2022 11:59:59 PM
8/8/2022	08/08/2022	8/8/2022 11:59:59 PM
8/19/2022	08/22/2022	8/22/2022 11:59:59 PM
9/26/2022	09/26/2022	9/26/2022 11:59:59 PM
10/14/2022	10/17/2022	10/17/2022 11:59:59 PM
10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

在本例中，因为 weekend() 函数中使用了为 1 的 first_week_date 参数，所以它将一周的第一天设置为星期二。

weekend() 函数的图表，first_week_day 示例



交易 8191 发生在 2 月 5 日。weekend() 函数确定该日期之后的第一个星期一，即本周的结束时间和返回值，是 2 月 6 日午夜 11:59:59。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生时周末的时间戳的计算作为应用程序的图表对象中的度量创建。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
```

```
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：`date`。

要计算交易发生的一周的开始时间，请添加以下度量：

- `=weekend(date)`
- `=timestamp(weekend(date))`

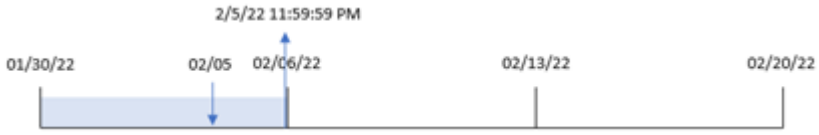
结果表

日期	=weekend(date)	=timestamp(weekend(date))
1/7/2022	01/08/2022	1/8/2022 11:59:59 PM
1/19/2022	01/22/2022	1/22/2022 11:59:59 PM
2/5/2022	02/05/2022	2/5/2022 11:59:59 PM
2/28/2022	03/05/2022	3/5/2022 11:59:59 PM
3/16/2022	03/19/2022	3/19/2022 11:59:59 PM
4/1/2022	04/02/2022	4/2/2022 11:59:59 PM
5/7/2022	05/07/2022	5/7/2022 11:59:59 PM
5/16/2022	05/21/2022	5/21/2022 11:59:59 PM
6/15/2022	06/18/2022	6/18/2022 11:59:59 PM
6/26/2022	07/02/2022	7/2/2022 11:59:59 PM
7/9/2022	07/09/2022	7/9/2022 11:59:59 PM
7/22/2022	07/23/2022	7/23/2022 11:59:59 PM
7/23/2022	07/23/2022	7/23/2022 11:59:59 PM
7/27/2022	07/30/2022	7/30/2022 11:59:59 PM

日期	=weekend(date)	=timestamp(weekend(date))
8/2/2022	08/06/2022	8/6/2022 11:59:59 PM
8/8/2022	08/13/2022	8/13/2022 11:59:59 PM
8/19/2022	08/20/2022	8/20/2022 11:59:59 PM
9/26/2022	10/01/2022	10/1/2022 11:59:59 PM
10/14/2022	10/15/2022	10/15/2022 11:59:59 PM
10/29/2022	10/29/2022	10/29/2022 11:59:59 PM

通过使用 weekend()函数并将日期字段作为函数的参数传递，在图表对象中创建了 end_of_week 度量。weekend() 函数标识日期值属于哪个周，并返回该周最后一毫秒的时间戳。

weekend() 函数的图表，图表对象示例



交易 8191 发生在 2 月 5 日。FirstWeekDay 系统变量将一周的第一天设置为星期日。weekend() 函数确定 2 月 5 日之后的第一个星期六，即本周的最后一个星期六是2月5号。因此，该事务的 end_of_week 值将返回当天的最后一毫秒，即二月 5 日午夜 11:59:59。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Employee_Expenses 的表中的数据集。
- 由员工 ID、员工姓名和每个员工的平均每日费用报销组成的数据。

最终用户需要一个图表对象，该图表对象按员工 ID 和员工姓名显示周剩余时间内仍要发生的估计费用索赔。

加载脚本

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
```

```
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将这些字段添加为维度：
   - employee_id
   - employee_name
2. 接下来，创建一个度量来计算累计利息：
   `=(weekend(today(1))-today(1))*avg_daily_claim`
3. 将度量的**数字格式**设置为**金额**。

结果表

EmployeeID	employee_name	=(weekend(today(1))-today(1))*avg_daily_claim
182	Mark	$90.00
183	Deryck	$75.00
184	Dexter	$75.00
185	Sydney	$162.00
186	Agatha	$108.00

weekend() 函数使用今天的日期作为唯一参数，返回当前周的结束日期。然后，通过从周结束日期中减去今天的日期，表达式返回本周剩余的天数。

然后将该值乘以每个员工的平均每日费用索赔，以计算每个员工在剩余周预计提出的索赔的估计值。

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 [, broken_weeks [, reference_
day]]]])
```

weekname() 函数确定日期属于哪一周，并返回该周的周数和年份。一周的第一天由 FirstWeekDay系统变量确定。但是，您可以使用 weekname() 函数中的 first_week_day 参数更改将哪个月设置为第一个月。

在 Qlik Sense 中，创建应用程序时获取区域设置，相应的设置作为环境变量存储在脚本中。

北美的应用程序开发人员经常在脚本中加入 `Set BrokenWeeks=1;`，对应于中断周。欧洲的应用程序开发人员经常在脚本中加入 `Set BrokenWeeks=0;`，对应于非中断周。

如果您的应用程序使用中断周，则周数计数从 1 月 1 日开始，到 `FirstWeekDay` 系统变量的前一天结束，而不管发生了多少天。

但是，如果您的应用程序使用的是连续的周，则第 1 周可以从上一年开始，也可以从 1 月的前几天开始。这取决于如何使用 `ReferenceDay` 和 `FirstWeekDay` 系统变量。

Weekend 函数示例

日期	ISO 周名称	US 周名称
2020 年 12 月 26 日星期六	2020/52	2020/52
2020 年 12 月 27 日星期日	2020/52	2020/53
2020 年 12 月 28 日星期一	2020/53	2020/53
2020 年 12 月 29 日星期二	2020/53	2020/53
2020 年 12 月 30 日星期三	2020/53	2020/53
2020 年 12 月 31 日星期四	2020/53	2020/53
2021 年 1 月 1 日星期五	2020/53	2021/01
2021 年 1 月 2 日周六	2020/53	2021/01
2021 年 1 月 3 日星期日	2020/53	2021/02
2021 年 1 月 4 日星期一	2021/01	2021/02
2021 年 1 月 5 日星期二	2021/01	2021/02

适用场景

当您希望按周比较聚合时，`weekname()` 函数非常有用。

例如，如果您想查看每周产品的总销售额。要保持与应用程序中 `BrokenWeeks` 环境变量的一致性，请使用 `weekname()` 而不是 `lunarweekname()`。如果应用程序使用不间断的周，则第 1 周可能包含上一年 12 月的日期，也可能不包括本年 1 月的日期。如果应用程序使用的是中断周，则第 1 周可能少于 7 天。

返回数据类型： 双

参数

参数	说明
timestamp	要评估的日期或时间戳。
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。

参数	说明
first_week_day	指定一周的开始日期。如果忽略，使用 **FirstWeekDay** 变量的值。 可能的值 **first_week_day** 为：周一为 0，周二为 1，周三为 2，周四为3，周五为 4，周六为 5，星期日为 6。 有关系统变量的详细信息，请参见*FirstWeekDay (page 218)*。
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。
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。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

以下示例假设：

```
Set FirstWeekDay=0;
Set BrokenWeeks=0;
Set ReferenceDay=4;
```

函数示例

示例	结果
weekname('01/12/2013')	返回 2013/02。
weekname('01/12/2013', -1)	返回 2013/01。
weekname('01/12/2013', 0, 1)	返回 2013/02。

示例 1 – 没有其他参数的日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2021 最后一周和 2022 年前两周的一组交易的数据集加载到名为 'Transactions' 的表中。
- 设置为 MM/DD/YYYY 格式的 DateFormat 系统变量。
- 设置为 1 的 BrokenWeeks 系统变量。
- 设置为 6 的 FirstWeekDay 系统变量。
- 包含以下内容的前置 Load：
    - 设置为字段 'week_number' 的函数 weekday()，返回交易发生时的年份和周数。
    - 设置为名为 'week_day' 的字段的 weekname() 函数，用于显示每个交易日期的工作日值。

加载脚本

```
SET BrokenWeeks=1;
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;

Transactions:
   Load
     *,
     WeekDay(date) as week_day,
     Weekname(date) as week_number
     ;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- week_day
- week_number

结果表

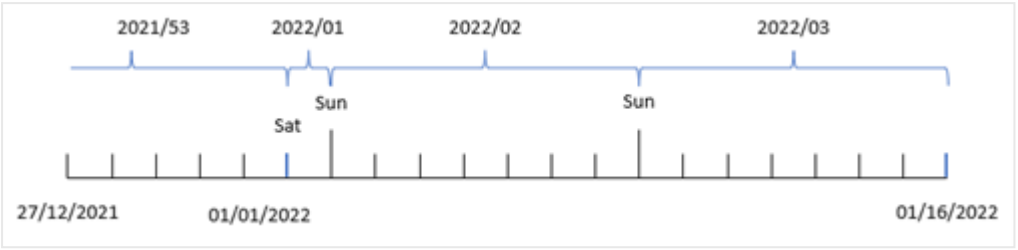
id	日期	week_day	week_number
8183	12/27/2021	Mon	2021/53
8184	12/28/2021	Tue	2021/53
8185	12/29/2021	Wed	2021/53
8186	12/30/2021	Thu	2021/53
8187	12/31/2021	Fri	2021/53
8188	01/01/2022	Sat	2022/01
8189	01/02/2022	Sun	2022/02
8190	01/03/2022	Mon	2022/02
8191	01/04/2022	Tue	2022/02
8192	01/05/2022	Wed	2022/02
8193	01/06/2022	Thu	2022/02
8194	01/07/2022	Fri	2022/02
8195	01/08/2022	Sat	2022/02
8196	01/09/2022	Sun	2022/03
8197	01/10/2022	Mon	2022/03
8198	01/11/2022	Tue	2022/03
8199	01/12/2022	Wed	2022/03
8200	01/13/2022	Thu	2022/03
8201	01/14/2022	Fri	2022/03

通过使用 weekname() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'week_number' 字段。

weekname() 函数最初确定日期值属于哪一周，并返回周数计数和交易发生的年份。

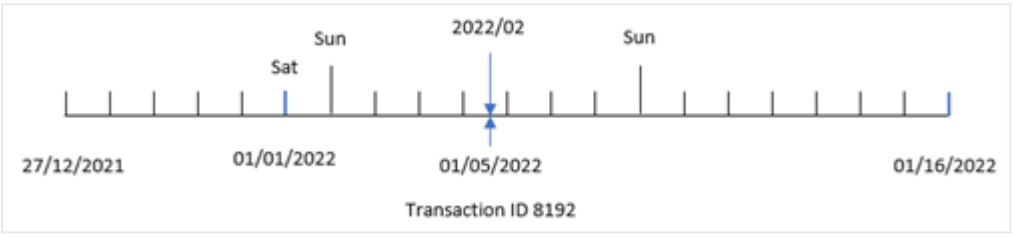
FirstWeekDay 系统变量将星期日设置为一周的第一天。BrokenWeeks 系统变量将应用程序设置为使用中断周，这意味着第 1 周将从 1 月 1 日开始。

带有默认变量的 weekname() 函数图表。



第 1 周开始于 1 月 1 日，即星期六，因此在此日期发生的交易返回值 2022/01（年和周数）。

识别交易 8192 的周数的 weekname() 函数图表。



由于应用程序使用的是不连续的星期，第一个工作日是星期日，因此从 1 月 2 日到 1 月 8 日发生的交易返回值 2022/02（2022 年的第 2 周）。例如，发生在 1 月 5 日的交易 8192 返回 'week_number' 字段的值 2022/02。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

但是，在本例中，任务是创建一个字段 'previous_week_number'，该字段返回交易发生之前的年份和周数。

打开 数据加载编辑器，并将以下加载脚本添加到新选项卡。

加载脚本

```
SET BrokenWeeks=1;
SET FirstWeekDay=6;

Transactions:
   Load
     *,
     weekname(date,-1) as previous_week_number
     ;
Load
*
Inline
```

```
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- `id`
- `date`
- `week_day`
- `week_number`

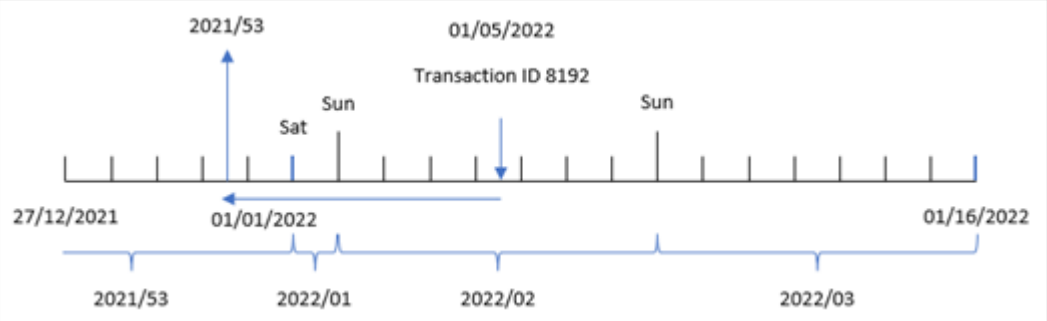
结果表

id	日期	week_day	week_number
8183	12/27/2021	Mon	2021/52
8184	12/28/2021	Tue	2021/52
8185	12/29/2021	Wed	2021/52
8186	12/30/2021	Thu	2021/52
8187	12/31/2021	Fri	2021/52
8188	01/01/2022	Sat	2021/52
8189	01/02/2022	Sun	2021/53
8190	01/03/2022	Mon	2021/53
8191	01/04/2022	Tue	2021/53
8192	01/05/2022	Wed	2021/53

id	日期	week_day	week_number
8193	01/06/2022	Thu	2021/53
8194	01/07/2022	Fri	2021/53
8195	01/08/2022	Sat	2022/01
8196	01/09/2022	Sun	2022/02
8197	01/10/2022	Mon	2022/02
8198	01/11/2022	Tue	2022/02
8199	01/12/2022	Wed	2022/02
8200	01/13/2022	Thu	2022/02
8201	01/14/2022	Fri	2022/02

由于 weekname() 函数中使用了 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的周。然后，它查找前一周，并确定该周的第一毫秒。

weekname() 函数的图表，其中 period_no 偏移为 -1.



交易 8192 发生在 2022 年 1 月 5 日。weekname() 函数查找 2021 年 12 月 30 日前一周，并返回该日期的周数和年份 – 2021/53。

示例 3 – first_week_day

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

However, in this example, the company policy is for the year to begin from April 1.

打开 数据加载编辑器，并将以下加载脚本添加到新选项卡。

加载脚本

```
SET BrokenWeeks=1;
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
   Load
     *,
     weekday(date) as week_day,
     weekname(date,0,1) as week_number
     ;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

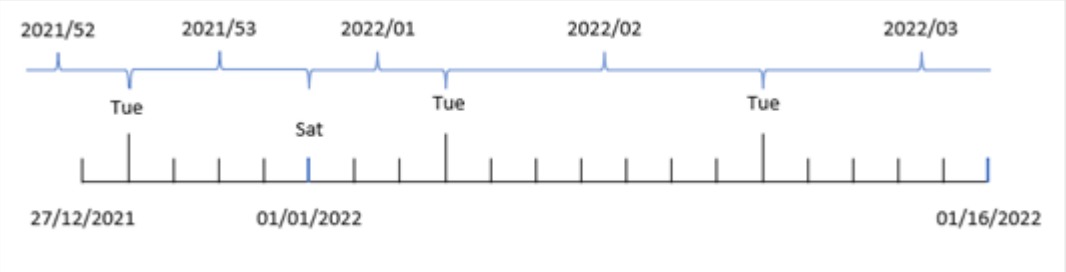
- id
- date
- week_day
- week_number

结果表

id	日期	week_day	week_number
8183	12/27/2021	Mon	2021/52
8184	12/28/2021	Tue	2021/53
8185	12/29/2021	Wed	2021/53
8186	12/30/2021	Thu	2021/53

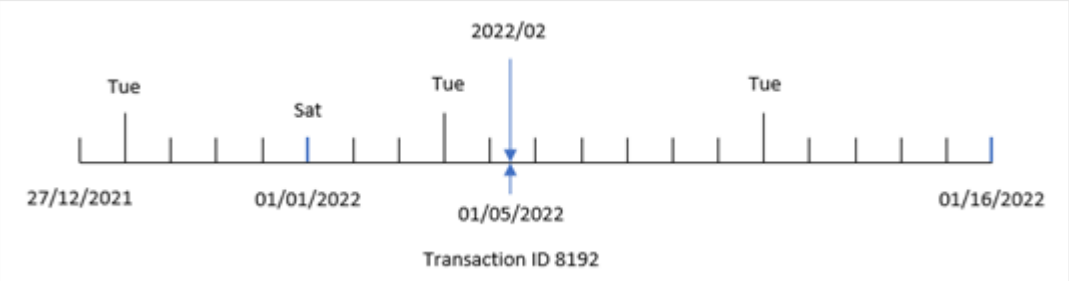
id	日期	week_day	week_number
8187	12/31/2021	Fri	2021/53
8188	01/01/2022	Sat	2022/01
8189	01/02/2022	Sun	2022/01
8190	01/03/2022	Mon	2022/01
8191	01/04/2022	Tue	2022/02
8192	01/05/2022	Wed	2022/02
8193	01/06/2022	Thu	2022/02
8194	01/07/2022	Fri	2022/02
8195	01/08/2022	Sat	2022/02
8196	01/09/2022	Sun	2022/02
8197	01/10/2022	Mon	2022/02
8198	01/11/2022	Tue	2022/03
8199	01/12/2022	Wed	2022/03
8200	01/13/2022	Thu	2022/03
8201	01/14/2022	Fri	2022/03

星期二为一周的第一天的 weekname() 函数的图表。



因为 weekname() 函数中使用了 1 的 first_week_date 参数，所以它使用星期二作为一周的第一天。因此，该函数确定 2021 年第 53 周从 12 月 28 日星期二开始；而且，由于应用程序使用了中断周，第一周从 2022 年 1 月 1 日开始，到 2022 年 2 月 3 日星期一的最后一毫秒结束。

显示交易 8192 的周数的图表，星期二为一周的第一天。



交易 8192 发生在 2022 年 1 月 5 日。因此，使用周二的 first_week_day 参数，weekname() 函数返回 'week_number' 字段的值 2022/02。

示例 4 – 图表对象

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。在应用程序的图表对象中创建了返回交易发生时的周数的计算，将其作为度量。

加载脚本

```
SET BrokenWeeks=1;
Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
```

```
8201,01/14/2022,18.52
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- =week_day (date)

要计算交易发生的周的开始时间，请创建以下度量：

```
=weekname(date)
```

结果表

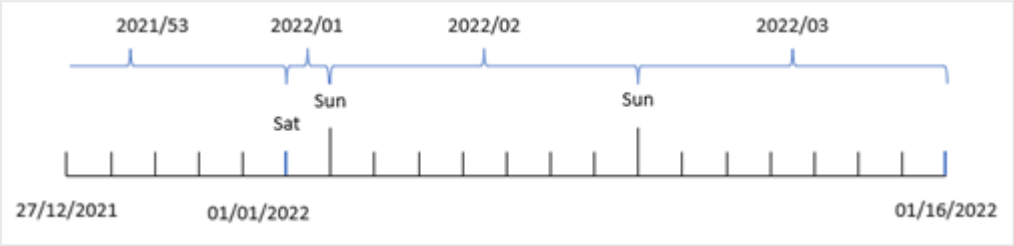
id	日期	=weekday(date)	=weekname(date)
8183	12/27/2021	Mon	2021/53
8184	12/28/2021	Tue	2021/53
8185	12/29/2021	Wed	2021/53
8186	12/30/2021	Thu	2021/53
8187	12/31/2021	Fri	2021/53
8188	01/01/2022	Sat	2022/01
8189	01/02/2022	Sun	2022/02
8190	01/03/2022	Mon	2022/02
8191	01/04/2022	Tue	2022/02
8192	01/05/2022	Wed	2022/02
8193	01/06/2022	Thu	2022/02
8194	01/07/2022	Fri	2022/02
8195	01/08/2022	Sat	2022/02
8196	01/09/2022	Sun	2022/03
8197	01/10/2022	Mon	2022/03
8198	01/11/2022	Tue	2022/03
8199	01/12/2022	Wed	2022/03
8200	01/13/2022	Thu	2022/03
8201	01/14/2022	Fri	2022/03

通过使用 weekname() 函数并将日期字段作为函数的参数传递，将 'week_number' 字段创建为图表对象中的度量。

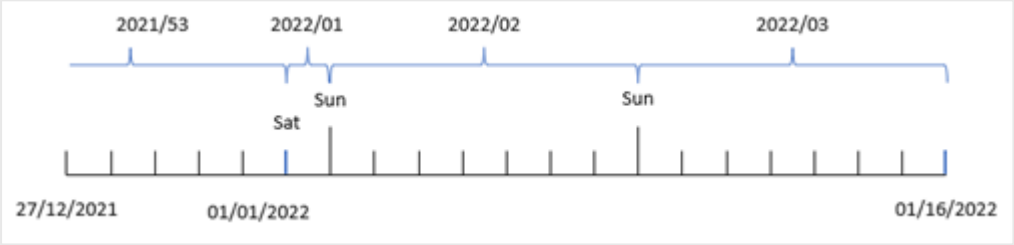
weekname() 函数最初确定日期值属于哪一周，并返回周数计数和交易发生的年份。

FirstWeekDay 系统变量将星期日设置为一周的第一天。BrokenWeeks 系统变量将应用程序设置为使用中断周，这意味着第 1 周从 1 月 1 日开始。

显示周数的图表，星期日为一周的第一天。



显示交易 8192 发生在第二周的图表。



由于应用程序使用的是不连续的星期，第一个工作日是星期日，因此从 1 月 2 日到 1 月 8 日发生的交易返回值 2022/02（2022 年的第 2 周）。请注意，交易 8192 发生在 1 月 5 日，并返回 'week_number' 字段的值 2022/02。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2019 最后一周和 2020 年前两周的一组交易的数据集加载到名为 'Transactions' 的表中。
- 设置为 0 的 BrokenWeeks 系统变量。
- 设置为 2 的 ReferenceDay 系统变量。
- 设置为 MM/DD/YYYY 格式的 DateFormat 系统变量。

加载脚本

```
SET BrokenWeeks=0;
SET ReferenceDay=2;
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2019,58.27
8184,12/28/2019,67.42
8185,12/29/2019,23.80
8186,12/30/2019,82.06
8187,12/31/2019,40.56
8188,01/01/2020,37.23
8189,01/02/2020,17.17
8190,01/03/2020,88.27
8191,01/04/2020,57.42
8192,01/05/2020,53.80
8193,01/06/2020,82.06
8194,01/07/2020,40.56
8195,01/08/2020,53.67
8196,01/09/2020,26.63
8197,01/10/2020,72.48
8198,01/11/2020,18.37
8199,01/12/2020,45.26
8200,01/13/2020,58.23
8201,01/14/2020,18.52
];
```

结果

加载数据并打开工作表。新建表格。

使用以下表达式创建计算维度：

```
=weekname(date)
```

要计算总销售额，请创建以下聚合度量：

```
=sum(amount)
```

将度量的**数字格式**设置为**金额**。

结果表

weekname(date)	=sum(amount)
2019/52	$125.69
2020/01	$346.51
2020/02	$347.57
2020/03	$122.01

要演示在此场景中使用 weekname() 函数的结果，请添加以下字段作为维度：

```
date
```

带有日期字段的结果表

weekname(date)	日期	=sum(amount)
2019/52	12/27/2019	$58.27
2019/52	12/28/2019	$67.42
2020/01	12/29/2019	$23.80
2020/01	12/30/2019	$82.06
2020/01	12/31/2019	$40.56
2020/01	01/01/2020	$37.23
2020/01	01/02/2020	$17.17
2020/01	01/03/2020	$88.27
2020/01	01/04/2020	$57.42
2020/02	01/05/2020	$53.80
2020/02	01/06/2020	$82.06
2020/02	01/07/2020	$40.56
2020/02	01/08/2020	$53.67
2020/02	01/09/2020	$26.63
2020/02	01/10/2020	$72.48
2020/02	01/11/2020	$18.37
2020/03	01/12/2020	$45.26
2020/03	01/13/2020	$58.23
2020/03	01/14/2020	$18.52

由于应用程序使用不间断的周，并且由于 ReferenceDay 系统变量，第 1 周在 1 月至少需要两天，因此 2020 年第 1 周包括 2019 年 12 月 29 日的交易。

weekstart

此函数用于返回与包含 **date** 的日历周的第一天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(timestamp [, period_no [, first_week_day ]])
```

返回数据类型：双

weekstart() 函数确定日期属于哪个周。然后以日期格式返回该周第一毫秒的时间戳。一周的第一天由 FirstWeekDay 环境变量确定。但是，这可以被 weekstart() 函数中的 first_week_day 参数取代。

参数

参数	说明
timestamp	要评估的日期或时间戳。
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。 可能的值 **first_week_day** 为：周一为 0，周二为 1，周三为 2，周四为3，周五为 4，周六为 5，星期日为 6。 有关系统变量的详细信息，请参见*FirstWeekDay (page 218)*。

适用场景

当用户希望计算使用到目前为止已过的一周的部分时，weekstart() 函数通常用作表达式的一部分。例如，如果用户想计算员工一周内迄今为止的工资总额，则可以使用它。

以下示例假设：

```
SET FirstWeekDay=0;
```

函数示例

示例	结果
weekstart('01/12/2013')	返回 01/07/2013。
weekstart('01/12/2013', -1)	返回 11/31/2012。
weekstart('01/12/2013', 0, 1)	返回 01/08/2013。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例：

如果您想要周和周数的 ISO 设置，请确保脚本中包含以下内容：

```
Set DateFormat   ='YYYY-MM-DD';
Set FirstWeekDay =0;     // Monday as first week day
Set BrokenWeeks  =0;     //(use unbroken weeks)
Set ReferenceDay =4;     // Jan 4th is always in week 1
```

如果需要 US 设置，请确保脚本中包含以下内容：

```
Set DateFormat   ='M/D/YYYY';
Set FirstWeekDay =6;    // Sunday as first week day
Set BrokenWeeks  =1;    //(use broken weeks)
Set ReferenceDay =1;    // Jan 1st is always in week 1
```

以上示例从 weekstart() 函数中得出以下结果：

Weekend 函数示例

日期	ISO 周初	US 周初
2020 年 12 月 26 日星期六	2020-12-21	12/20/2020
2020 年 12 月 27 日星期日	2020-12-21	12/27/2020
2020 年 12 月 28 日星期一	2020-12-28	12/27/2020
2020 年 12 月 29 日星期二	2020-12-28	12/27/2020
2020 年 12 月 30 日星期三	2020-12-28	12/27/2020
2020 年 12 月 31 日星期四	2020-12-28	12/27/2020
2021 年 1 月 1 日星期五	2020-12-28	12/27/2020
2021 年 1 月 2 日周六	2020-12-28	12/27/2020
2021 年 1 月 3 日星期日	2020-12-28	1/3/2021
2021 年 1 月 4 日星期一	2021-01-04	1/3/2021
2021 年 1 月 5 日星期二	2021-01-04	1/3/2021

ISO 列中的周初为星期一，US 列中的周初为星期日。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2022 年交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建字段 start_of_week，返回交易发生的周开始的时间戳。

加载脚本

```
SET FirstWeekDay=6;

Transactions:
    Load
        *,
```

```
        weekstart(date) as start_of_week,
        timestamp(weekstart(date)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- start_of_week
- start_of_week_timestamp

结果表

日期	start_of_week	start_of_week_timestamp
1/7/2022	01/02/2022	1/2/2022 12:00:00 AM
1/19/2022	01/16/2022	1/16/2022 12:00:00 AM
2/5/2022	01/30/2022	1/30/2022 12:00:00 AM
2/28/2022	02/27/2022	2/27/2022 12:00:00 AM
3/16/2022	03/13/2022	3/13/2022 12:00:00 AM
4/1/2022	03/27/2022	3/27/2022 12:00:00 AM

日期	start_of_week	start_of_week_timestamp
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/15/2022	5/15/2022 12:00:00 AM
6/15/2022	06/12/2022	6/12/2022 12:00:00 AM
6/26/2022	06/26/2022	6/26/2022 12:00:00 AM
7/9/2022	07/03/2022	7/3/2022 12:00:00 AM
7/22/2022	07/17/2022	7/17/2022 12:00:00 AM
7/23/2022	07/17/2022	7/17/2022 12:00:00 AM
7/27/2022	07/24/2022	7/24/2022 12:00:00 AM
8/2/2022	07/31/2022	7/31/2022 12:00:00 AM
8/8/2022	08/07/2022	8/7/2022 12:00:00 AM
8/19/2022	08/14/2022	8/14/2022 12:00:00 AM
9/26/2022	09/25/2022	9/25/2022 12:00:00 AM
10/14/2022	10/09/2022	10/9/2022 12:00:00 AM
10/29/2022	10/23/2022	10/23/2022 12:00:00 AM

通过使用 weekstart() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 start_of_week 字段。

weekstart() 函数初始标识日期值属于哪个周，并返回该周第一毫秒的时间戳。

weekstart() 函数图表，示例没有额外参数

交易 8191 发生在 2 月 5 日。FirstWeekDay 系统变量将一周的第一天设置为星期日。weekstart() 函数确定 2 月 5 日之前的第一个星期日(因此是一周的开始)是 1 月 30 日。因此，该交易的start_of_week 值返回当天的第一毫秒，即 1 月 30 号中午 12:00:00。

示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 previous_week_start, 该字段返回季度发生前的季度开始的时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        weekstart(date,-1) as previous_week_start,
        timestamp(weekstart(date,-1)) as previous_week_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

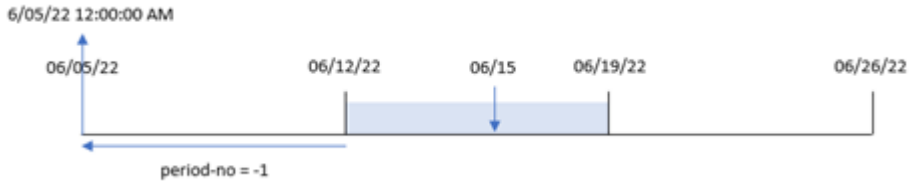
- date
- previous_week_start
- previous_week_start_timestamp

结果表

日期	previous_week_start	previous_week_start_timestamp
1/7/2022	12/26/2021	12/26/2021 12:00:00 AM
1/19/2022	01/09/2022	1/9/2022 12:00:00 AM
2/5/2022	01/23/2022	1/23/2022 12:00:00 AM
2/28/2022	02/20/2022	2/20/2022 12:00:00 AM
3/16/2022	03/06/2022	3/6/2022 12:00:00 AM
4/1/2022	03/20/2022	3/20/2022 12:00:00 AM
5/7/2022	04/24/2022	4/24/2022 12:00:00 AM
5/16/2022	05/08/2022	5/8/2022 12:00:00 AM
6/15/2022	06/05/2022	6/5/2022 12:00:00 AM
6/26/2022	06/19/2022	6/19/2022 12:00:00 AM
7/9/2022	06/26/2022	6/26/2022 12:00:00 AM
7/22/2022	07/10/2022	7/10/2022 12:00:00 AM
7/23/2022	07/10/2022	7/10/2022 12:00:00 AM
7/27/2022	07/17/2022	7/17/2022 12:00:00 AM
8/2/2022	07/24/2022	7/24/2022 12:00:00 AM
8/8/2022	07/31/2022	7/31/2022 12:00:00 AM
8/19/2022	08/07/2022	8/7/2022 12:00:00 AM
9/26/2022	09/18/2022	9/18/2022 12:00:00 AM
10/14/2022	10/02/2022	10/2/2022 12:00:00 AM
10/29/2022	10/16/2022	10/16/2022 12:00:00 AM

在本例中，由于 weekstart() 函数中使用了为 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的周。然后，它查找前一周，并确定该周的第一毫秒。

weekstart() 函数的图表，period_no 示例



交易 8196 发生在 6 月 15 日。weekstart() 函数确定周从 6 月 12 日开始。因此，前一周开始于 6 月 5 日中午 12:00:00；这是为 previous_week_start 字段返回的值。

示例 3 – first_week_day

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。但是，在本例中，我们需要将星期二设置为工作周的第一天。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        weekstart(date,0,1) as start_of_week,
        timestamp(weekstart(date,0,1)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- start_of_week
- start_of_week_timestamp

结果表

日期	start_of_week	start_of_week_timestamp
1/7/2022	01/04/2022	1/4/2022 12:00:00 AM
1/19/2022	01/18/2022	1/18/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/22/2022	2/22/2022 12:00:00 AM
3/16/2022	03/15/2022	3/15/2022 12:00:00 AM
4/1/2022	03/29/2022	3/29/2022 12:00:00 AM
5/7/2022	05/03/2022	5/3/2022 12:00:00 AM
5/16/2022	05/10/2022	5/10/2022 12:00:00 AM
6/15/2022	06/14/2022	6/14/2022 12:00:00 AM
6/26/2022	06/21/2022	6/21/2022 12:00:00 AM
7/9/2022	07/05/2022	7/5/2022 12:00:00 AM
7/22/2022	07/19/2022	7/19/2022 12:00:00 AM
7/23/2022	07/19/2022	7/19/2022 12:00:00 AM
7/27/2022	07/26/2022	7/26/2022 12:00:00 AM
8/2/2022	08/02/2022	8/2/2022 12:00:00 AM
8/8/2022	08/02/2022	8/2/2022 12:00:00 AM
8/19/2022	08/16/2022	8/16/2022 12:00:00 AM
9/26/2022	09/20/2022	9/20/2022 12:00:00 AM
10/14/2022	10/11/2022	10/11/2022 12:00:00 AM
10/29/2022	10/25/2022	10/25/2022 12:00:00 AM

在本例中，因为 weekstart() 函数中使用了为 1 的 first_week_date 参数，所以它将一周的第一天设置为星期二。

weekstart() 函数的图表，first_week_day 示例

交易 8191 发生在 2 月 5 日。weekstart() 函数确定该日期之前的第一个星期二，即本周的开始时间和返回的值，是 2 月 1 日中午 12:00:00。

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生时周初的时间戳的计算作为应用程序的图表对象中的度量创建。

加载脚本

```
Transactions:

Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

要计算交易发生的一周的开始时间，请添加以下度量：

- =weekstart(date)
- =timestamp(weekstart(date))

结果表

日期	start_of_week	start_of_week_timestamp
1/7/2022	01/02/2022	1/2/2022 12:00:00 AM
1/19/2022	01/16/2022	1/16/2022 12:00:00 AM
2/5/2022	01/30/2022	1/30/2022 12:00:00 AM
2/28/2022	02/27/2022	2/27/2022 12:00:00 AM
3/16/2022	03/13/2022	3/13/2022 12:00:00 AM
4/1/2022	03/27/2022	3/27/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/15/2022	5/15/2022 12:00:00 AM
6/15/2022	06/12/2022	6/12/2022 12:00:00 AM
6/26/2022	06/26/2022	6/26/2022 12:00:00 AM
7/9/2022	07/03/2022	7/3/2022 12:00:00 AM
7/22/2022	07/17/2022	7/17/2022 12:00:00 AM
7/23/2022	07/17/2022	7/17/2022 12:00:00 AM
7/27/2022	07/24/2022	7/24/2022 12:00:00 AM
8/2/2022	07/31/2022	7/31/2022 12:00:00 AM
8/8/2022	08/07/2022	8/7/2022 12:00:00 AM
8/19/2022	08/14/2022	8/14/2022 12:00:00 AM
9/26/2022	09/25/2022	9/25/2022 12:00:00 AM
10/14/2022	10/09/2022	10/9/2022 12:00:00 AM
10/29/2022	10/23/2022	10/23/2022 12:00:00 AM

通过使用 weekstart() 函数并将 date 字段作为函数的参数传递，在图表对象中创建 start_of_week 度量。

weekstart() 函数初始标识日期值属于哪个周，并返回该周第一毫秒的时间戳。

weekstart() 函数的图表，图表对象示例

交易 8191 发生在 2 月 5 日。FirstWeekDay 系统变量将一周的第一天设置为星期日。weekstart() 函数确定 2 月 5 日之前的第一个星期日(因此是本周的开始)是 1 月 30 日。因此，该交易的 start_of_week 值返回当天的第一毫秒，即 1 月 30 日上午 12:00:00。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Payroll 的表中的数据集。
- 由员工 ID、员工姓名和每位员工的每日工资组成的数据。

员工周一开始上班，每周工作六天。不得修改 FirstWeekDay 系统变量。

最终用户需要一个图表对象，该对象按员工 ID 和员工姓名显示本周迄今为止的工资。

加载脚本

```
Payroll:
Load
*
Inline
[
employee_id,employee_name,day_rate
182,Mark, $150
183,Deryck, $125
184,Dexter, $125
185,Sydney,$270
186,Agatha,$128
];
```

结果

执行以下操作：

1. 加载数据并打开工作表。创建新表并将这些字段添加为维度：
   - employee_id
   - employee_name
2. 接下来，创建一个度量来计算本周迄今为止的工资：
   `=if(today(1)-weekstart(today(1),0,0)<7,(today(1)-weekstart(today(1),0,0))*day_rate,day_rate*6)`
3. 将度量的**数字格式**设置为**金额**。

结果表

EmployeeID	employee_name	=if(today(1)-weekstart(today(1),0,0)<7,(today(1)-weekstart(today(1),0,0))*day_rate,day_rate*6)
182	Mark	$600.00
183	Deryck	$500.00
184	Dexter	$500.00
185	Sydney	$1080.00
186	Agatha	$512.00

weekstart() 函数使用今天的日期作为第一个参数，使用 0 作为第三个参数，将星期一设置为一周的第一天，并返回当前一周的开始日期。通过从当前日期中减去该结果，表达式将返回本周迄今为止经过的天数。

然后，条件评估本周是否有超过六天的时间。如果是这样，则员工的 day_rate 乘以 6 天。否则，将 day_rate 乘以本周迄今为止发生的天数。

weekyear

此函数用于返回根据环境变量周数所属的年份。星期数范围在 1 和大约 52 之间。

语法：

```
weekyear(timestamp [, first_week_day [, broken_weeks [, reference_day]]])
```

返回数据类型： 整数

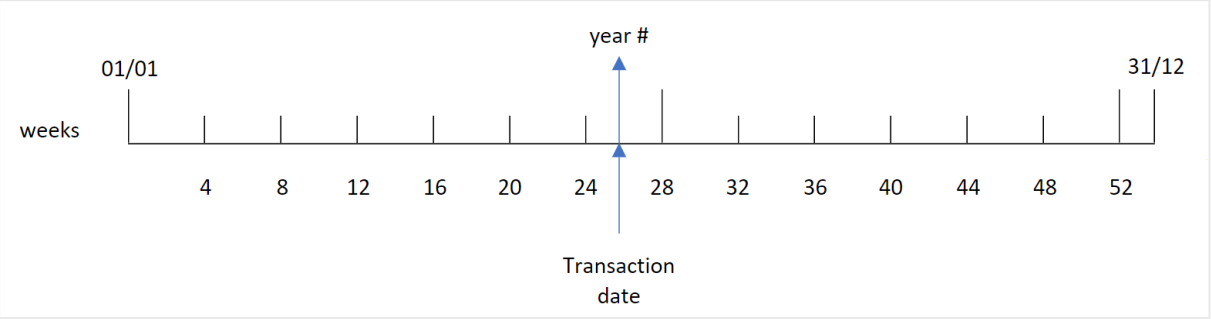
参数

参数	说明
timestamp	要评估的日期或时间戳。
first_week_day	指定一周的开始日期。如果忽略，使用 **FirstWeekDay** 变量的值。 可能的值 **first_week_day** 为：周一为 0，周二为 1，周三为 2，周四为3，周五为 4，周六为 5，星期日为 6。 有关系统变量的详细信息，请参见*FirstWeekDay (page 218)*。
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。
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。

weekyear() 函数确定日期属于年份的哪个周。然后返回与该周数对应的年份。

如果 BrokenWeeks 设置为 0 (false)，weekyear() 将返回与 year() 相同的。

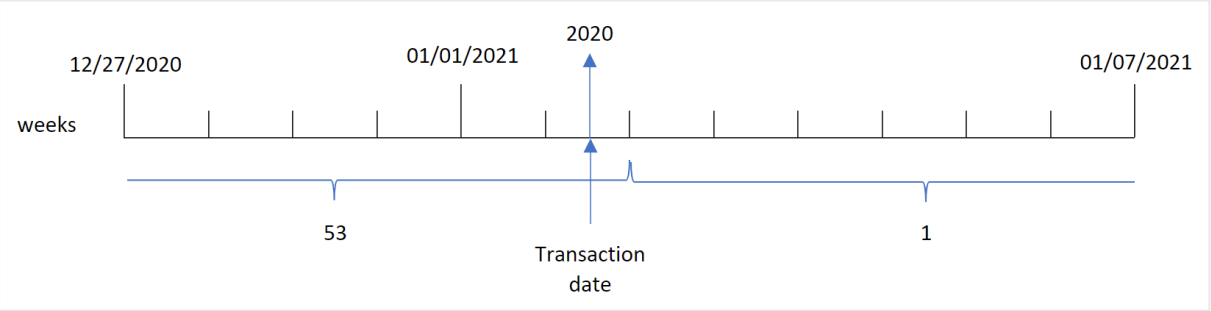
weekyear() 函数范围的图表



但是，如果 BrokenWeeks 系统变量设置为使用连续周，则根据 ReferenceDay 系统变量中指定的值，第 1 周只能包含 1 月份的特定天数。

例如，如果使用 4 的 ReferenceDay 值，则第 1 周必须包括 1 月份的至少四天。第 1 周可以包括上一年 12 月的日期，或者一年的最后一周可以包括下一年 1 月的日期。在这种情况下，weekyear() 函数将向 year() 函数返回不同的值。

使用非中断周时的 weekyear() 函数范围的图表



适用场景

当您希望按年份比较聚合时，weekyear() 函数非常有用。例如，如果您希望按年份查看产品的总销售额。当用户希望与应用程序中的 BrokenWeeks 系统变量保持一致时，可以选择 weekyear() 函数而非 year()。

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
weekyear('12/30/1996',0,0,4)	返回 1997，因为 1997 年第 1 周从 1996 年 12 月 30 日开始
weekyear('01/02/1997',0,0,4)	返回 1997
weekyear('12/28/1997',0,0,4)	返回 1997
weekyear('12/30/1997',0,0,4)	返回 1998，因为 1998 年第 1 周从 1997 年 12 月 29 日开始
weekyear('01/02/1999',0,0,4)	返回 1998，因为 1998 的第 53 周在 1999 年的 1 月 3 日结束

相关主题

主题	交互
week (page 1005)	返回根据 ISO 8601 表示周数的整数
year (page 1075)	根据标准数字解释当表达式被解释为日期时返回一个表示年份的整数。

示例 1 - 中断周

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年最后一周和 2021 第一周的一组交易的数据集，该数据集加载到名为 'Transactions' 的表中。
- BrokenWeeks 变量，设置为 1。
- 包含以下内容的前置 Load：
  - weekyear() 函数，设置为字段 'week_year'，返回交易发生的年份。
  - week() 函数设置为字段 'week'，显示每个交易日期的周数。

加载脚本

```
SET BrokenWeeks=1;

Transactions:
       Load
       *,
       week(date) as week,
       weekyear(date) as week_year
       ;
Load
*
Inline
[
```

```
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- week
- week_year

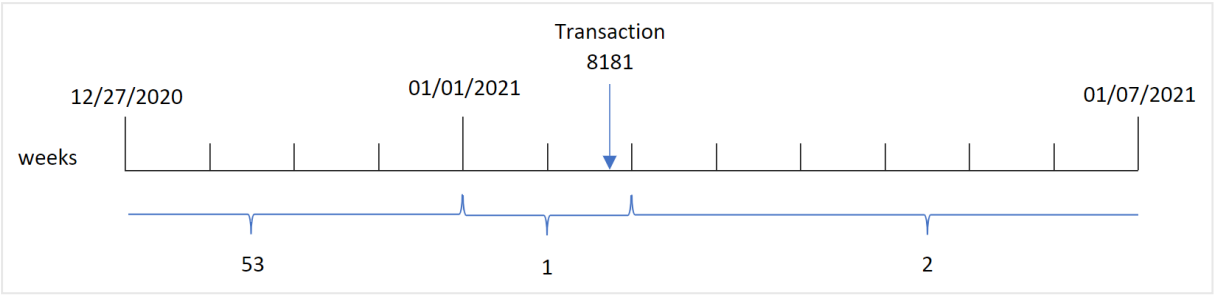
结果表

id	日期	周	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	1	2021
8181	01/02/2021	1	2021
8182	01/03/2021	2	2021
8183	01/04/2021	2	2021
8184	01/05/2021	2	2021
8185	01/06/2021	2	2021
8186	01/07/2021	2	2021

通过使用 weekyear() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'week_year' 字段。

BrokenWeeks 系统变量设置为 1，意味着应用程序使用中断周。第 1 周从 1 月 1 日开始。

使用中断周的 weekyear() 函数范围的图表



交易 8181 发生在1月2日，这是第 1 周的一部分。因此，它为 'week_year' 字段返回 2021 的值。

示例 2 - 非中断周

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年最后一周和 2021 第一周的一组交易的数据集，该数据集加载到名为 'Transactions' 的表中。
- BrokenWeeks 变量，设置为 0。
- 包含以下内容的前置 Load：
  - weekyear() 函数，设置为字段 'week_year'，返回交易发生的年份。
  - week() 函数设置为字段 'week'，显示每个交易日期的周数。

但是，在本例中，公司政策是使用非中断周。

加载脚本

```
SET BrokenWeeks=0;

Transactions:
        Load
        *,
        week(date) as week,
        weekyear(date) as week_year
        ;
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
```

```
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- week
- week_year

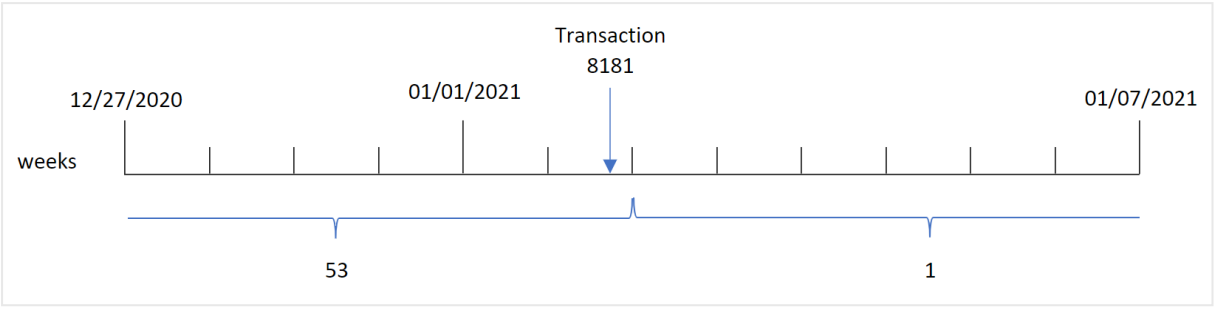
结果表

id	日期	周	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	53	2020
8181	01/02/2021	53	2020
8182	01/03/2021	1	2021
8183	01/04/2021	1	2021
8184	01/05/2021	1	2021
8185	01/06/2021	1	2021
8186	01/07/2021	1	2021

BrokenWeeks 系统变量设置为 0 表示应用程序使用非中断的周。因此，第一周不需要从 1 月 1 日开始。

2020 年第 53 周持续到 2021 年 1 月 2 日，2021 第 1 周从 2021 年 1 月 3 日星期日开始。

使用非中断周的 weekyear() 函数范围的图表



交易 8181 发生在1月2日，这是第 1 周的一部分。因此，它为 'week_year' 字段返回 2021 的值。

示例 3 – 图表对象示例

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生年份的周数的计算将在应用程序中的图表中创建为度量。

加载脚本

```
SET BrokenWeeks=1;

Transactions:
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date

要计算交易发生的周，请创建以下度量：

- =week(date)

要根据周数计算交易发生的年份，请创建以下度量：

- =weekyear(date)

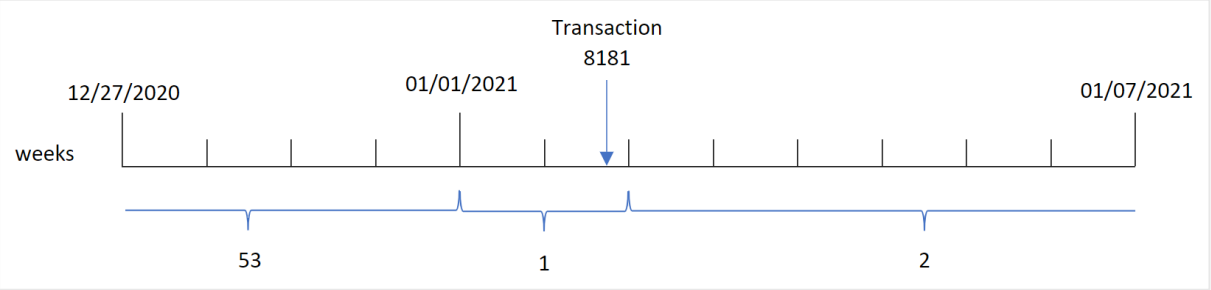
结果表

id	日期	周	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	1	2021
8181	01/02/2021	1	2021
8182	01/03/2021	2	2021
8183	01/04/2021	2	2021
8184	01/05/2021	2	2021
8185	01/06/2021	2	2021
8186	01/07/2021	2	2021

通过使用 weekyear() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'week_year' 字段。

BrokenWeeks 系统变量设置为 1，意味着应用程序使用中断周。周 1 从 1 月 1 日开始。

使用中断周的 weekyear() 函数范围的图表



交易 8181 发生在1月2日，这是第 1 周的一部分。因此，它为 'week_year' 字段返回 2021 的值。

示例 4 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年最后一周和 2021 第一周的一组交易的数据集，该数据集加载到名为 'Transactions' 的表中。
- BrokenWeeks 变量，设置为 0。这意味着应用程序将使用非中断周。
- ReferenceDay 变量，设置为 2。这意味着这一年将从 1 月 2 日开始，1 月至少有两天。
- FirstWeekDay 变量，设置为 1。这意味着一周的第一天是星期二。

公司的政策是使用中断周。最终用户想要一个按年度列出总销售额的图表。该应用程序使用连续的周，第一周在 1 月至少包含两天。

加载脚本

```
SET BrokenWeeks=0;
SET ReferenceDay=2;
SET FirstWeekDay=1;

Transactions:
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

结果

加载数据并打开工作表。新建表格。

要根据周数计算交易发生的年份，请创建以下度量：

- =weekyear(date)

要计算总销售额，请创建以下度量：

- sum(amount)

将度量的**数字格式**设置为**金额**。

结果表

weekyear(date)	=sum(amount)
2020	19.42
2021	373.37

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

year() 函数可作为脚本和图表函数使用。该函数返回特定日期的年份。它通常用于在主日历中创建年份字段作为维度。

适用场景

当您希望按年份比较聚合时，year() 函数非常有用。例如，如果您想查看每年产品的总销售额，则函数可以使用它。

通过使用函数在主日历表中创建字段，可以在 Load 脚本中创建这些维度。或者，它可以直接在图表中用作计算维度。

函数示例

示例	结果
year('2012-10-12')	返回 2012
year('35648')	返回 1997，因为 35648 = 1997-08-06

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – DateFormat 数据集(脚本)

加载脚本和结果

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

- 加载到名为 Master Calendar 的表中的日期数据集。
- 使用默认 DateFormat 系统变量 MM/DD/YYYY。
- 使用 year() 函数创建另一个名为 year 的字段的前置 Load。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Master_Calendar:

    Load
        date,
        year(date) as year
        ;
Load
date
Inline
[
date
12/28/2020
12/29/2020
12/30/2020
12/31/2020
01/01/2021
01/02/2021
01/03/2021
01/04/2021
01/05/2021
01/06/2021
01/07/2021
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度:

- date
- year

结果表

日期	年
12/28/2020	2020
12/29/2020	2020
12/30/2020	2020
12/31/2020	2020
01/01/2021	2021
01/02/2021	2021
01/03/2021	2021
01/04/2021	2021
01/05/2021	2021
01/06/2021	2021
01/07/2021	2021

示例 2–ANSI 日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Master Calendar 的表中的日期数据集。
- 使用默认 DateFormat 系统变量 (MM/DD/YYYY)。但是，数据集中包含的日期采用 ANSI 标准日期格式。
- 使用 year() 函数创建另一个名为 year 的字段的前置 Load。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Master_Calendar:
    Load
        date,
        year(date) as year
        ;
Load
date
Inline
[
date
2020-12-28
```

```
2020-12-29
2020-12-30
2020-12-31
2021-01-01
2021-01-02
2021-01-03
2021-01-04
2021-01-05
2021-01-06
2021-01-07
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- year

结果表

日期	年
2020-12-28	2020
2020-12-29	2020
2020-12-30	2020
2020-12-31	2020
2021-01-01	2021
2021-01-02	2021
2021-01-03	2021
2021-01-04	2021
2021-01-05	2021
2021-01-06	2021
2021-01-07	2021

示例 3 – 未格式化日期

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 加载到名为 Master Calendar 的表中的数字格式日期数据集。
- 使用默认 DateFormat 系统变量 (MM/DD/YYYY)。
- 使用 year() 函数创建另一个名为 year 的字段的前置 Load。

原始的未格式化日期被加载并命名为 unformatted_date，为了清晰起见，使用另一个名为 long_date 的字段，使用 date() 函数将数字日期转换为格式化日期字段。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Master_Calendar:
    Load
        unformatted_date,
        date(unformatted_date) as long_date,
        year(unformatted_date) as year
        ;
Load
unformatted_date
Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- unformatted_date
- long_date
- year

结果表

unformatted_date	long_date	年
44868	11/03/2022	2022
44898	12/03/2022	2022
44928	01/02/2023	2023

unformatted_date	long_date	年
44958	02/01/2023	2023
44988	03/03/2023	2023
45008	03/23/2023	2023
45018	04/02/2023	2023
45038	04/22/2023	2023
45048	05/02/2023	2023
45068	05/22/2023	2023
45078	06/01/2023	2023

示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

在本例中，所下订单的数据集被加载到名为 Sales 的表中。表格包含三个字段：

- id
- sales_date
- amount

产品销售保修期为自销售之日起两年。任务是在图表中创建一个度量，以确定每个保修的到期年份。

加载脚本

```
Sales:
Load
id,
sales_date,
amount
Inline
[
id,sales_date,amount
1,12/28/2020,231.24,
2,12/29/2020,567.28,
3,12/30/2020,364.28,
4,12/31/2020,575.76,
5,01/01/2021,638.68,
6,01/02/2021,785.38,
7,01/03/2021,967.46,
8,01/04/2021,287.67
9,01/05/2021,764.45,
```

```
10,01/06/2021,875.43,
11,01/07/2021,957.35
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：sales_date。

创建以下度量：

```
=year(sales_date+365*2)
```

结果表

sales_date	=year(sales_date+365*2)
12/28/2020	2022
12/29/2020	2022
12/30/2020	2022
12/31/2020	2022
01/01/2021	2023
01/02/2021	2023
01/03/2021	2023
01/04/2021	2023
01/05/2021	2023
01/06/2021	2023
01/07/2021	2023

该度量的结果见上表。要将一个日期加上两年，请将 365 乘以 2，并将结果加到销售日期。因此，2020 年销售额的到期年份为 2022 年。

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**。

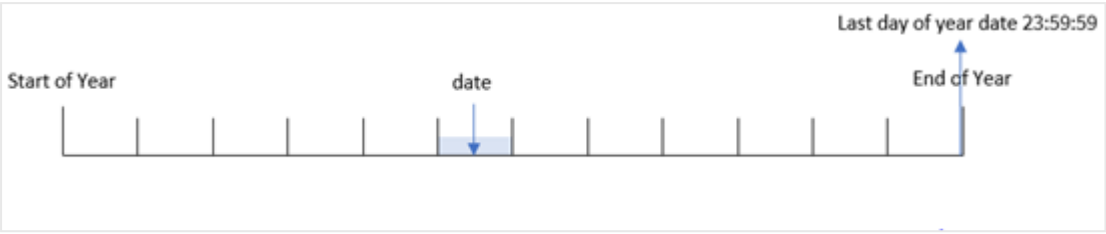
语法：

```
YearEnd( date[, period_no[, first_month_of_year = 1]])
```

换句话说，yearend() 函数确定日期属于哪一年。然后以日期格式返回该年最后一毫秒的时间戳。默认情况下，一年的第一个月是一月。但是，您可以使用 yearend() 函数中的 first_month_of_year 参数更改将哪个月设置为第一个月。

> *yearend() 函数不考虑 FirstMonthOfYear 系统变量。这一年从 1 月 1 日开始，除非用 first_month_of_year 参数来改变它。*

yearend() 函数的图表。



适用场景

当您希望计算使用尚未发生的年份分数时，yearend() 函数将用作表达式的一部分。例如，您想计算一年中尚未发生的利息总额。

返回数据类型： 双

参数

参数	说明
date	要评估的日期或时间戳。
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。

可以使用以下值在 first_month_of_year 参数中设置一年中的第一个月：

first_month_of_year 值

月	值
二月	2
三月	3
四月	4
五月	5
六月	6
七月	7
八月	8
九月	9
十月	10
十一月	11
十二月	12

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
yearend('10/19/2001')	Returns 12/31/2001 23:59:59.
yearend('10/19/2001', -1)	Returns 12/31/2000 23:59:59.
yearend('10/19/2001', 0, 4)	Returns 03/31/2002 23:59:59.

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年和 2022 年之间的一组事务的数据集加载到名为 'Transactions' 的表中。
- 日期字段以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 前置 Load 语句包含以下内容：
  - 设置为 year_end 字段的 yearend() 函数。
  - 设置为 year_end_timestamp 字段的 Timestamp() 函数。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       yearend(date) as year_end,
       timestamp(yearend(date)) as year_end_timestamp
       ;
Load
```

```
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- year_end
- year_end_timestamp

结果表

id	日期	year_end	year_end_timestamp
8188	01/13/2020	12/31/2020	12/31/2020 11:59:59 PM
8189	02/26/2020	12/31/2020	12/31/2020 11:59:59 PM
8190	03/27/2020	12/31/2020	12/31/2020 11:59:59 PM
8191	04/16/2020	12/31/2020	12/31/2020 11:59:59 PM
8192	05/21/2020	12/31/2020	12/31/2020 11:59:59 PM
8193	08/14/2020	12/31/2020	12/31/2020 11:59:59 PM
8194	10/07/2020	12/31/2020	12/31/2020 11:59:59 PM
8195	12/05/2020	12/31/2020	12/31/2020 11:59:59 PM

id	日期	year_end	year_end_timestamp
8196	01/22/2021	12/31/2021	12/31/2021 11:59:59 PM
8197	02/03/2021	12/31/2021	12/31/2021 11:59:59 PM
8198	03/17/2021	12/31/2021	12/31/2021 11:59:59 PM
8199	04/23/2021	12/31/2021	12/31/2021 11:59:59 PM
8200	05/04/2021	12/31/2021	12/31/2021 11:59:59 PM
8201	06/30/2021	12/31/2021	12/31/2021 11:59:59 PM
8202	07/26/2021	12/31/2021	12/31/2021 11:59:59 PM
8203	12/27/2021	12/31/2021	12/31/2021 11:59:59 PM
8204	06/06/2022	12/31/2022	12/31/2022 11:59:59 PM
8205	07/18/2022	12/31/2022	12/31/2022 11:59:59 PM
8206	11/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	12/12/2022	12/31/2022	12/31/2022 11:59:59 PM

通过使用 yearend() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'year_end' 字段。

yearend() 函数最初确定日期值属于哪一年，并返回该年最后一毫秒的时间戳。

选择了交易 8199 的 yearend() 函数的图表。

交易 8199 发生在 2021 年 4 月 23 日。yearend() 函数返回该年的最后一毫秒，即 12 月 31 日晚上 11:59:59。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

但是，在本例中，任务是创建一个字段 'previous_year_end'，该字段返回交易发生年份前一年的结束日期时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       yearend(date,-1) as previous_year_end,
       timestamp(yearend(date,-1)) as previous_year_end_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- previous_year_end
- previous_ year_end_timestamp

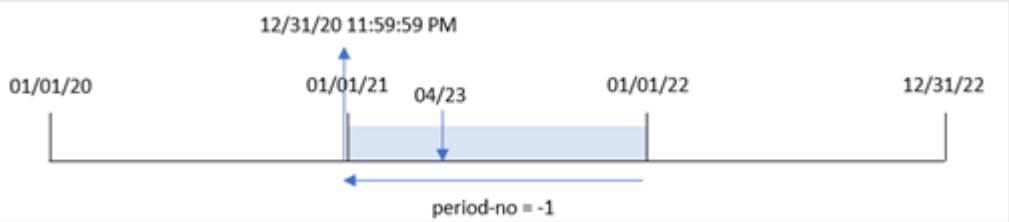
结果表

id	日期	previous_year_end	previous_year_end_timestamp
8188	01/13/2020	12/31/2019	12/31/2019 11:59:59 PM

id	日期	previous_year_end	previous_year_end_timestamp
8189	02/26/2020	12/31/2019	12/31/2019 11:59:59 PM
8190	03/27/2020	12/31/2019	12/31/2019 11:59:59 PM
8191	04/16/2020	12/31/2019	12/31/2019 11:59:59 PM
8192	05/21/2020	12/31/2019	12/31/2019 11:59:59 PM
8193	08/14/2020	12/31/2019	12/31/2019 11:59:59 PM
8194	10/07/2020	12/31/2019	12/31/2019 11:59:59 PM
8195	12/05/2020	12/31/2019	12/31/2019 11:59:59 PM
8196	01/22/2021	12/31/2020	12/31/2020 11:59:59 PM
8197	02/03/2021	12/31/2020	12/31/2020 11:59:59 PM
8198	03/17/2021	12/31/2020	12/31/2020 11:59:59 PM
8199	04/23/2021	12/31/2020	12/31/2020 11:59:59 PM
8200	05/04/2021	12/31/2020	12/31/2020 11:59:59 PM
8201	06/30/2021	12/31/2020	12/31/2020 11:59:59 PM
8202	07/26/2021	12/31/2020	12/31/2020 11:59:59 PM
8203	12/27/2021	12/31/2020	12/31/2020 11:59:59 PM
8204	06/06/2022	12/31/2021	12/31/2021 11:59:59 PM
8205	07/18/2022	12/31/2021	12/31/2021 11:59:59 PM
8206	11/14/2022	12/31/2021	12/31/2021 11:59:59 PM
8207	12/12/2022	12/31/2021	12/31/2021 11:59:59 PM

由于 yearend() 函数中使用了 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的年。然后，它查找前一年，并确定该年的最后一毫秒。

yearend() 函数的图表，其中 period_no 为 -1.



交易 8199 发生在 2021 年 4 月 23 日。yearend() 函数返回 'previous_year_end' 字段上一年（2020 年 12 月 31 日晚上 11:59:59）的最后一毫秒。

示例 3 – first_month_of_year

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，公司政策是从 4 月 1 日开始的一年。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       yearend(date,0,4) as year_end,
       timestamp(yearend(date,0,4)) as year_end_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

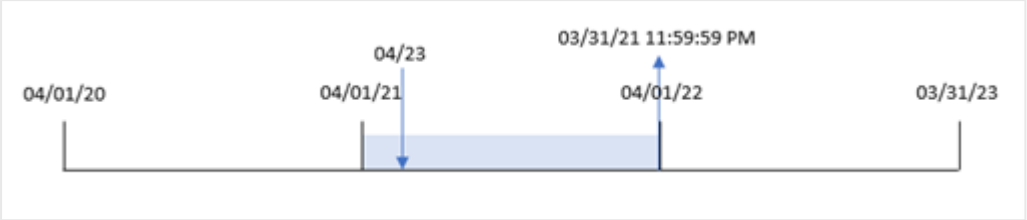
- id
- date
- year_end
- year_end_timestamp

结果表

id	日期	year_end	year_end_timestamp
8188	01/13/2020	03/31/2020	3/31/2020 11:59:59 PM
8189	02/26/2020	03/31/2020	3/31/2020 11:59:59 PM
8190	03/27/2020	03/31/2020	3/31/2020 11:59:59 PM
8191	04/16/2020	03/31/2021	3/31/2021 11:59:59 PM
8192	05/21/2020	03/31/2021	3/31/2021 11:59:59 PM
8193	08/14/2020	03/31/2021	3/31/2021 11:59:59 PM
8194	10/07/2020	03/31/2021	3/31/2021 11:59:59 PM
8195	12/05/2020	03/31/2021	3/31/2021 11:59:59 PM
8196	01/22/2021	03/31/2021	3/31/2021 11:59:59 PM
8197	02/03/2021	03/31/2021	3/31/2021 11:59:59 PM
8198	03/17/2021	03/31/2021	3/31/2021 11:59:59 PM
8199	04/23/2021	03/31/2022	3/31/2022 11:59:59 PM
8200	05/04/2021	03/31/2022	3/31/2022 11:59:59 PM
8201	06/30/2021	03/31/2022	3/31/2022 11:59:59 PM
8202	07/26/2021	03/31/2022	3/31/2022 11:59:59 PM
8203	12/27/2021	03/31/2022	3/31/2022 11:59:59 PM
8204	06/06/2022	03/31/2023	3/31/2023 11:59:59 PM
8205	07/18/2022	03/31/2023	3/31/2023 11:59:59 PM
8206	11/14/2022	03/31/2023	3/31/2023 11:59:59 PM
8207	12/12/2022	03/31/2023	3/31/2023 11:59:59 PM

由于 yearend() 函数中使用了 4 的 first_month_of_year 参数，因此它将一年中的第一天设置为 4 月 1 日，将一年的最后一天设置为 3 月 31 日。

yearend() 函数的图表，4 月为一年中的第一个月。



交易 8199 发生在 2021 年 4 月 23 日。由于 yearend() 函数将年初设置为 4 月 1 日，因此返回 March 31, 2022 作为交易的 'year_end' 值。

示例 4 – 图表对象

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生年份的结束日期时间戳的计算是作为应用程序的图表对象中的度量创建的。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date

要计算交易发生的年份，请创建以下度量：

- =yearend(date)
- =timestamp(yearend(date))

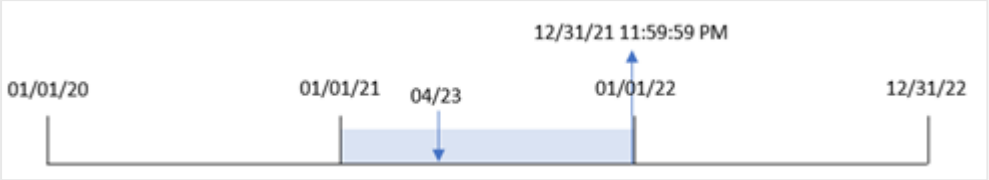
结果表

id	日期	=yearend(date)	=timestamp(yearend(date))
8188	01/13/2020	12/31/2020	12/31/2020 11:59:59 PM
8189	02/26/2020	12/31/2020	12/31/2020 11:59:59 PM
8190	03/27/2020	12/31/2020	12/31/2020 11:59:59 PM
8191	04/16/2020	12/31/2020	12/31/2020 11:59:59 PM
8192	05/21/2020	12/31/2020	12/31/2020 11:59:59 PM
8193	08/14/2020	12/31/2020	12/31/2020 11:59:59 PM
8194	10/07/2020	12/31/2020	12/31/2020 11:59:59 PM
8195	12/05/2020	12/31/2020	12/31/2020 11:59:59 PM
8196	01/22/2021	12/31/2021	12/31/2021 11:59:59 PM
8197	02/03/2021	12/31/2021	12/31/2021 11:59:59 PM
8198	03/17/2021	12/31/2021	12/31/2021 11:59:59 PM
8199	04/23/2021	12/31/2021	12/31/2021 11:59:59 PM
8200	05/04/2021	12/31/2021	12/31/2021 11:59:59 PM
8201	06/30/2021	12/31/2021	12/31/2021 11:59:59 PM
8202	07/26/2021	12/31/2021	12/31/2021 11:59:59 PM
8203	12/27/2021	12/31/2021	12/31/2021 11:59:59 PM
8204	06/06/2022	12/31/2022	12/31/2022 11:59:59 PM
8205	07/18/2022	12/31/2022	12/31/2022 11:59:59 PM
8206	11/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	12/12/2022	12/31/2022	12/31/2022 11:59:59 PM

通过使用 yearend()函数并将日期字段作为函数的参数传递，在图表对象中创建了 'end_of_year' 度量。

yearend() 函数最初确定日期值属于哪一年，并返回该年最后一毫秒的时间戳。

显示交易 8199 发生在 4 月的 yearend() 函数的图表。



交易 8199 发生在 2021 年 4 月 23 日。yearend() 函数返回该年的最后一毫秒，即 12 月 31 日晚上 11:59:59。

示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 数据集加载到名为 'Employee_Expenses' 的表中。表格包含以下字段：
  - 员工 ID
  - 员工名称
  - 每位员工的平均每日费用报销

最终用户需要一个图表对象，该对象按员工 ID 和员工姓名显示该年剩余时间仍将发生的估计费用索赔。财政年度从一月份开始。

加载脚本

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- employee_id
- employee_name

要计算预计费用索赔，请创建以下度量：

```
=(yearend(today(1))-today(1))*avg_daily_claim
```

将度量的**数字格式**设置为**金额**。

结果表

EmployeeID	employee_name	=(yearend(today(1))-today(1))*avg_daily_claim
182	Mark	$3240.00
183	Deryck	$2700.00
184	Dexter	$2700.00
185	Sydney	$5832.00
186	Agatha	$3888.00

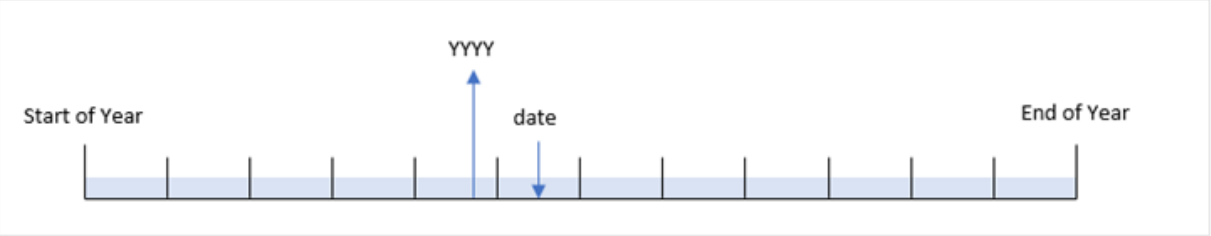
通过使用今天的日期作为其唯一参数，yearend() 函数返回当前年份的结束日期。然后，通过从年终日期中减去今天的日期，表达式返回一年中剩余的天数。

然后将该值乘以每个员工的平均每日费用索赔，以计算每个员工在剩余年份预计提出的索赔的估计值。

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。

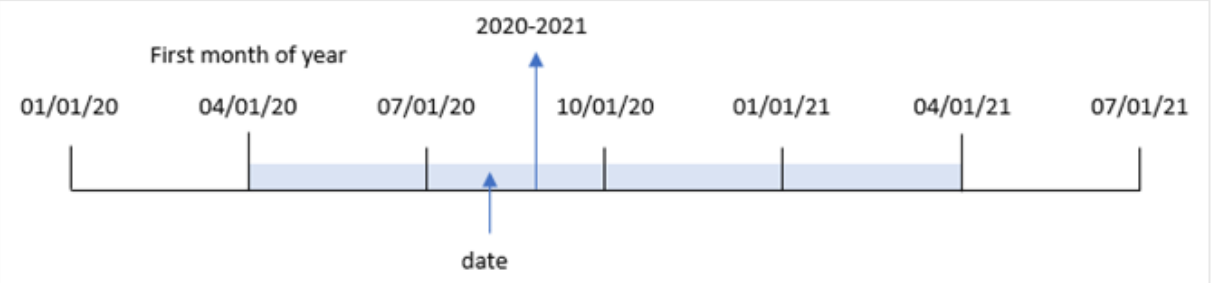
yearname() 函数的时间范围的图表。



yearname() 函数与 year() 函数不同，因为它允许您偏移要评估的日期，并允许您设置一年中的第一个月。

如果一年的第一个月不是一月，则函数将返回包含日期的十二个月期间的两个四位数年份。例如，如果年初为 4 月，评估日期为 2020 年 6 月 30 日，则返回的结果为 2020-2021。

yearname() 函数的图表，4 月为一年中的第一个月。



语法：

```
YearName(date[, period_no[, first_month_of_year]] )
```

返回数据类型：双

参数	说明
date	要评估的日期或时间戳。
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。

可以使用以下值在 `first_month_of_year` 参数中设置一年中的第一个月：

first_month_of_year 值

月	值
二月	2
三月	3
四月	4
五月	5
六月	6
七月	7
八月	8
九月	9
十月	10
十一月	11
十二月	12

适用场景

yearname() 函数用于按年份比较聚合。例如，如果您想查看每年产品的总销售额。

通过使用函数在主日历表中创建字段，可以在 Load 脚本中创建这些维度。它们也可以在图表中创建为计算尺寸

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 SET DateFormat 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
yearname('10/19/2001')	Returns '2001.'
yearname('10/19/2001',-1)	Returns '2000.'
yearname('10/19/2001',0,4)	Returns '2001-2002.'

相关主题

主题	说明
year (page 1075)	此函数用于根据标准数字解释当表达式被解释为日期时返回一个表示年份的整数。

示例 1 – 没有其他参数

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年和 2022 年之间的一组事务的数据集加载到名为 'Transactions' 的表中。
- DateFormat 系统变量，设置为 'MM/DD/YYYY'。
- 使用 yearname() 并且其被设置为 year_name 字段的前置 Load。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
      *,
      yearname(date) as year_name
      ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- year_name

结果表

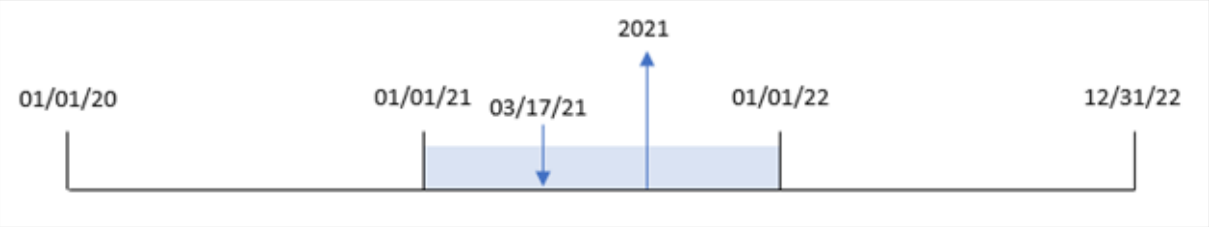
日期	year_name
01/13/2020	2020
02/26/2020	2020
03/27/2020	2020
04/16/2020	2020

日期	year_name
05/21/2020	2020
08/14/2020	2020
10/07/2020	2020
12/05/2020	2020
01/22/2021	2021
02/03/2021	2021
03/17/2021	2021
04/23/2021	2021
05/04/2021	2021
06/30/2021	2021
07/26/2021	2021
12/27/2021	2021
06/06/2022	2022
07/18/2022	2022
11/14/2022	2022
12/12/2022	2022

通过使用 yearname() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'year_name' 字段。

yearname() 函数确定日期值属于哪一年，并将其作为四位数的年值返回。

显示 2021 为年份值的 yearname() 函数的图表。



示例 2 – period_no

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年和 2022 年之间的一组交易的数据集被加载到名为“交易”的表中。
- DateFormat 系统变量，设置为 'MM/DD/YYYY'。
- 使用 yearname() 并且其被设置为 year_name 字段的前置 Load。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
      *,
      yearname(date,-1) as prior_year_name
      ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- prior_year_name

结果表

日期	prior_year_name
01/13/2020	2019

日期	prior_year_name
02/26/2020	2019
03/27/2020	2019
04/16/2020	2019
05/21/2020	2019
08/14/2020	2019
10/07/2020	2019
12/05/2020	2019
01/22/2021	2020
02/03/2021	2020
03/17/2021	2020
04/23/2021	2020
05/04/2021	2020
06/30/2021	2020
07/26/2021	2020
12/27/2021	2020
06/06/2022	2021
07/18/2022	2021
11/14/2022	2021
12/12/2022	2021

由于 yearname() 函数中使用了 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的年。然后，函数移到前一年并返回结果年。

yearname() 函数的图表，其中 period_no 设置为 -1.

示例 3 – first_month_of_year

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个 中的相同数据集。
- DateFormat 系统变量，设置为 'MM/DD/YYYY'。
- 使用 yearname() 并且其被设置为 year_name 字段的前置 Load。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
      *,
      yearname(date,0,4) as year_name
      ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- year_name

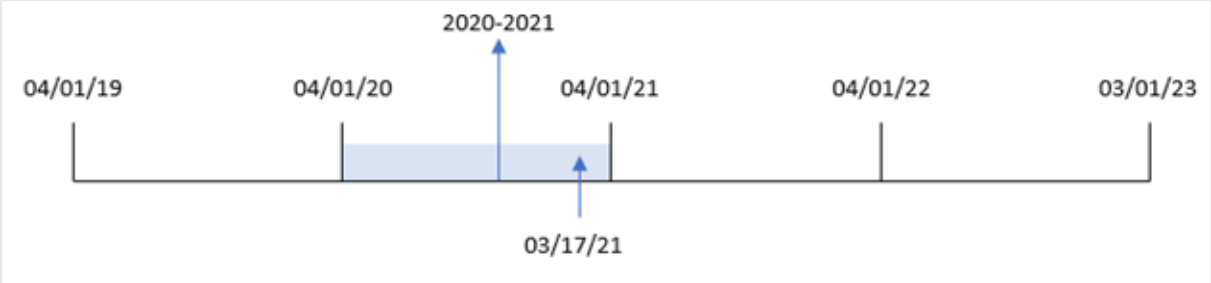
结果表

日期	year_name
01/13/2020	2019-2020
02/26/2020	2019-2020
03/27/2020	2019-2020
04/16/2020	2020-2021
05/21/2020	2020-2021
08/14/2020	2020-2021
10/07/2020	2020-2021
12/05/2020	2020-2021
01/22/2021	2020-2021
02/03/2021	2020-2021
03/17/2021	2020-2021
04/23/2021	2021-2022
05/04/2021	2021-2022
06/30/2021	2021-2022
07/26/2021	2021-2022
12/27/2021	2021-2022
06/06/2022	2022-2023
07/18/2022	2022-2023
11/14/2022	2022-2023
12/12/2022	2022-2023

由于yearname() 函数中使用了 4 的 first_month_of_year 参数，因此年初从 1 月 1 日移动到 4 月 1 日。因此，每个 12 个月的周期跨越两个日历年，yearname() 函数返回两个四位数的年份作为评估日期。

交易 8198 发生在 2021 年 3 月 17 日。yearname() 函数将年初设置为 4 月 1 日，结束日期设置为 3 月 30 日。因此，交易 8198 发生在 2020 年 4 月 1 日至 2021 年 3 月 30 日这一年期间。因此，yearname() 函数返回 2020-2021 的值。

yearname() 函数的图表，3 月为一年中的第一个月。



示例 4 – 图表对象

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个 中的相同数据集。
- DateFormat 系统变量，设置为 'MM/DD/YYYY'。

但是，返回交易发生年份的字段是作为图表对象中的度量创建的。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
```

```
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：

```
date
```

要计算 year_name 字段，请创建此度量：

```
=yearname(date)
```

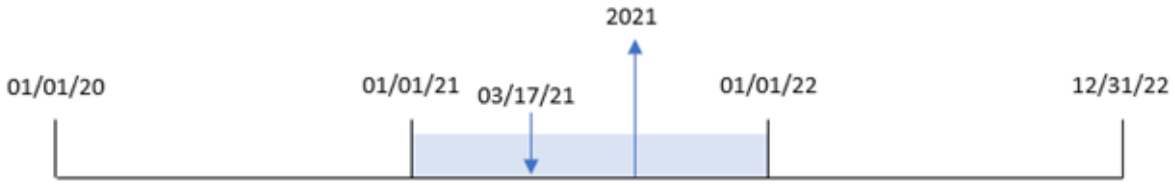
结果表

日期	=yearname(date)
01/13/2020	2020
02/26/2020	2020
03/27/2020	2020
04/16/2020	2020
05/21/2020	2020
08/14/2020	2020
10/07/2020	2020
12/05/2020	2020
01/22/2021	2021
02/03/2021	2021
03/17/2021	2021
04/23/2021	2021
05/04/2021	2021
06/30/2021	2021
07/26/2021	2021
12/27/2021	2021
06/06/2022	2022
07/18/2022	2022
11/14/2022	2022
12/12/2022	2022

通过使用 yearname()函数并将日期字段作为函数的参数传递，在图表对象中创建了 'year_name' 度量。

yearname() 函数确定日期值属于哪一年，并将其作为四位数的年值返回。

显示 2021 为年份值的 yearname() 函数的图表。



示例 5 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 第一个 中的相同数据集。
- DateFormat 系统变量，设置为 'MM/DD/YYYY'。

最终用户希望得到一个图表，该图表按季度显示交易的总销售额。当 yearname() 维度在数据模型中不可用时，使用 yearname() 函数作为计算维度创建此图表。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
```

```
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

结果

加载数据并打开工作表。新建表格。

要按年份比较聚合，请创建以下计算维度：

```
=yearname(date)
```

创建该度量：

```
=sum(amount)
```

将度量的**数字格式**设置为**金额**。

结果表

yearname(date)	=sum(amount)
2020	$463.55
2021	$457.69
2022	$294.35

yearstart

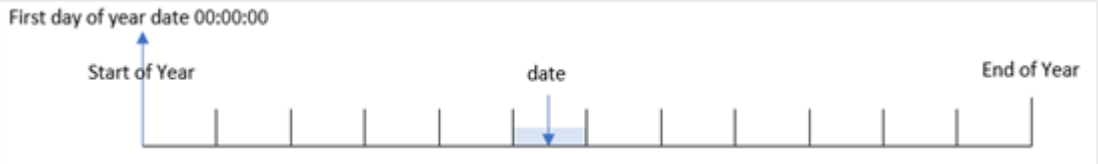
此函数用于返回与包含 **date** 的年份的第一天的开始时间对应的时间戳。默认的输出格式为在脚本中所设置的 **DateFormat**。

语法：

YearStart(date[, period_no[, first_month_of_year]])

换句话说，yearstart() 函数确定日期属于哪一年。然后以日期格式返回该年第一毫秒的时间戳。默认情况下，一年的第一个月是 1 月；但是，您可以使用 yearstart() 函数中的 first_month_of_year 参数更改将哪个月设置为第一个月。

yearstart() 函数的图表，显示函数可以覆盖的时间范围。



适用场景

当您希望计算使用目前已经过的年份分数时，yearstart() 函数将用作表达式的一部分。例如，如果您要计算年初至今累计的利息。

返回数据类型：双

参数

参数	说明
date	要评估的日期或时间戳。
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。

以下月份可用于 `first_month_of_year argument`：

first_month_of_year 值

月	值
二月	2
三月	3
四月	4
五月	5
六月	6
七月	7
八月	8
九月	9
十月	10
十一月	11
十二月	12

区域设置

除非另有规定，本主题中的示例使用以下日期格式：MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素，系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者，您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典，则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

函数示例

示例	结果
yearstart('10/19/2001')	Returns 01/01/2001 00:00:00.
yearstart('10/19/2001',-1)	Returns 01/01/2000 00:00:00.
yearstart('10/19/2001',0,4)	Returns 04/01/2001 00:00:00.

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年和 2022 年之间的一组事务的数据集加载到名为 'Transactions' 的表中。
- 日期字段以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 前置 Load 语句包含以下内容：
    - 设置为 year_start 字段的 yearstart() 函数。
    - 设置为 year_start_timestamp 字段的 Timestamp() 函数

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       yearstart(date) as year_start,
       timestamp(yearstart(date)) as year_start_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
```

```
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- year_start
- year_start_timestamp

结果表

id	日期	year_start	year_start_timestamp
8188	01/13/2020	01/01/2020	1/1/2020 12:00:00 AM
8189	02/26/2020	01/01/2020	1/1/2020 12:00:00 AM
8190	03/27/2020	01/01/2020	1/1/2020 12:00:00 AM
8191	04/16/2020	01/01/2020	1/1/2020 12:00:00 AM
8192	05/21/2020	01/01/2020	1/1/2020 12:00:00 AM
8193	08/14/2020	01/01/2020	1/1/2020 12:00:00 AM
8194	10/07/2020	01/01/2020	1/1/2020 12:00:00 AM
8195	12/05/2020	01/01/2020	1/1/2020 12:00:00 AM
8196	01/22/2021	01/01/2021	1/1/2021 12:00:00 AM
8197	02/03/2021	01/01/2021	1/1/2021 12:00:00 AM
8198	03/17/2021	01/01/2021	1/1/2021 12:00:00 AM
8199	04/23/2021	01/01/2021	1/1/2021 12:00:00 AM
8200	05/04/2021	01/01/2021	1/1/2021 12:00:00 AM
8201	06/30/2021	01/01/2021	1/1/2021 12:00:00 AM
8202	07/26/2021	01/01/2021	1/1/2021 12:00:00 AM
8203	12/27/2021	01/01/2021	1/1/2021 12:00:00 AM
8204	06/06/2022	01/01/2022	1/1/2022 12:00:00 AM

id	日期	year_start	year_start_timestamp
8205	07/18/2022	01/01/2022	1/1/2022 12:00:00 AM
8206	11/14/2022	01/01/2022	1/1/2022 12:00:00 AM
8207	12/12/2022	01/01/2022	1/1/2022 12:00:00 AM

通过使用 yearstart() 函数并将日期字段作为函数的参数传递，在前置 Load 语句中创建了 'year_start' 字段。

yearstart() 函数最初确定日期值属于哪一年，并返回该年第一毫秒的时间戳。

yearstart() 函数和交易 8199 的图表。

交易 8199 发生在 2021 年 4 月 23 日。yearstart() 函数返回该年的第一毫秒，即 1 月 1 日上午 12:00:00。

示例 2 – period_no

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

但是，在本例中，任务是创建一个字段 'previous_year_start'，该字段返回交易发生年份前一年的开始日期时间戳。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       yearstart(date,-1) as previous_year_start,
       timestamp(yearstart(date,-1)) as previous_year_start_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
```

```
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- previous_year_start
- previous_ year_start_timestamp

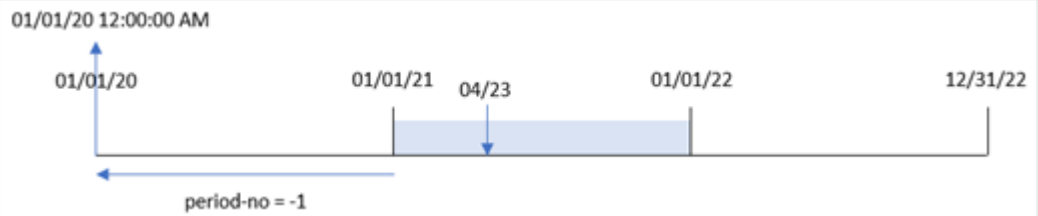
结果表

id	日期	previous_year_start	previous_year_start_timestamp
8188	01/13/2020	01/01/2019	1/1/2019 12:00:00 AM
8189	02/26/2020	01/01/2019	1/1/2019 12:00:00 AM
8190	03/27/2020	01/01/2019	1/1/2019 12:00:00 AM
8191	04/16/2020	01/01/2019	1/1/2019 12:00:00 AM
8192	05/21/2020	01/01/2019	1/1/2019 12:00:00 AM
8193	08/14/2020	01/01/2019	1/1/2019 12:00:00 AM
8194	10/07/2020	01/01/2019	1/1/2019 12:00:00 AM
8195	12/05/2020	01/01/2019	1/1/2019 12:00:00 AM
8196	01/22/2021	01/01/2020	1/1/2020 12:00:00 AM
8197	02/03/2021	01/01/2020	1/1/2020 12:00:00 AM
8198	03/17/2021	01/01/2020	1/1/2020 12:00:00 AM
8199	04/23/2021	01/01/2020	1/1/2020 12:00:00 AM
8200	05/04/2021	01/01/2020	1/1/2020 12:00:00 AM

id	日期	previous_year_start	previous_year_start_timestamp
8201	06/30/2021	01/01/2020	1/1/2020 12:00:00 AM
8202	07/26/2021	01/01/2020	1/1/2020 12:00:00 AM
8203	12/27/2021	01/01/2020	1/1/2020 12:00:00 AM
8204	06/06/2022	01/01/2021	1/1/2021 12:00:00 AM
8205	07/18/2022	01/01/2021	1/1/2021 12:00:00 AM
8206	11/14/2022	01/01/2021	1/1/2021 12:00:00 AM
8207	12/12/2022	01/01/2021	1/1/2021 12:00:00 AM

在本例中，由于 yearstart() 函数中使用了 -1 的 period_no 作为偏移参数，因此函数首先标识交易发生的年。然后，它查找前一年，并确定该年的第一一毫秒。

yearstart() 函数的图表，其中 period_no 为 -1。



交易 8199 发生在 2021 年 4 月 23 日。yearstart() 函数返回 'previous_year_start' 字段上一年 (2020 年 1 月 1 日午夜 12:00:00) 的第一毫秒。

示例 3 – first_month_of_year

加载脚本和结果

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，公司政策是从 4 月 1 日开始的一年。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       yearstart(date,0,4) as year_start,
       timestamp(yearstart(date,0,4)) as year_start_timestamp
       ;
Load
*
Inline
```

```
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date
- year_start
- year_start_timestamp

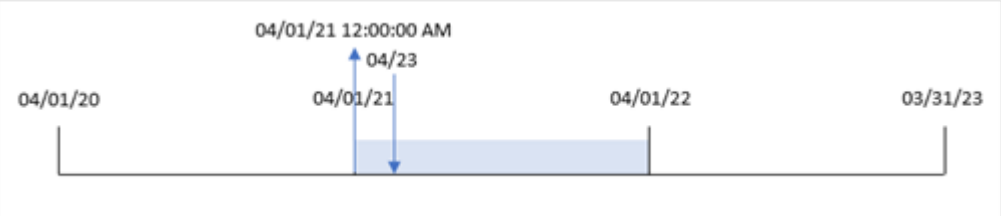
结果表

id	日期	year_start	year_start_timestamp
8188	01/13/2020	04/01/2019	4/1/2019 12:00:00 AM
8189	02/26/2020	04/01/2019	4/1/2019 12:00:00 AM
8190	03/27/2020	04/01/2019	4/1/2019 12:00:00 AM
8191	04/16/2020	04/01/2020	4/1/2020 12:00:00 AM
8192	05/21/2020	04/01/2020	4/1/2020 12:00:00 AM
8193	08/14/2020	04/01/2020	4/1/2020 12:00:00 AM
8194	10/07/2020	04/01/2020	4/1/2020 12:00:00 AM
8195	12/05/2020	04/01/2020	4/1/2020 12:00:00 AM
8196	01/22/2021	04/01/2020	4/1/2020 12:00:00 AM

id	日期	year_start	year_start_timestamp
8197	02/03/2021	04/01/2020	4/1/2020 12:00:00 AM
8198	03/17/2021	04/01/2020	4/1/2020 12:00:00 AM
8199	04/23/2021	04/01/2021	4/1/2021 12:00:00 AM
8200	05/04/2021	04/01/2021	4/1/2021 12:00:00 AM
8201	06/30/2021	04/01/2021	4/1/2021 12:00:00 AM
8202	07/26/2021	04/01/2021	4/1/2021 12:00:00 AM
8203	12/27/2021	04/01/2021	4/1/2021 12:00:00 AM
8204	06/06/2022	04/01/2022	4/1/2022 12:00:00 AM
8205	07/18/2022	04/01/2022	4/1/2022 12:00:00 AM
8206	11/14/2022	04/01/2022	4/1/2022 12:00:00 AM
8207	12/12/2022	04/01/2022	4/1/2022 12:00:00 AM

在本例中，由于 yearstart() 函数中使用了 4 的 first_month_of_year 参数，因此它将一年中的第一天设置为 4 月 1 日，将一年的最后一天设置为 3 月 31 日。

yearstart() 函数的图表，4 月设定为第一个月。



交易 8199 发生在 2021 年 4 月 23 日。由于 yearstart() 函数将年初设置为 4 月 1 日，因此将它返回作为交易的 'year_start' 值。

示例 4 – 图表对象

加载脚本和图表表达式

概述

使用与第一个 相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。返回交易发生年份的开始日期时间戳的计算是作为应用程序的图表对象中的度量创建的。

加载脚本

```
Transactions:
Load
*
```

```
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- id
- date

要计算交易发生的年份，请创建以下度量：

- =yearstart(date)
- =timestamp(yearstart(date))

结果表

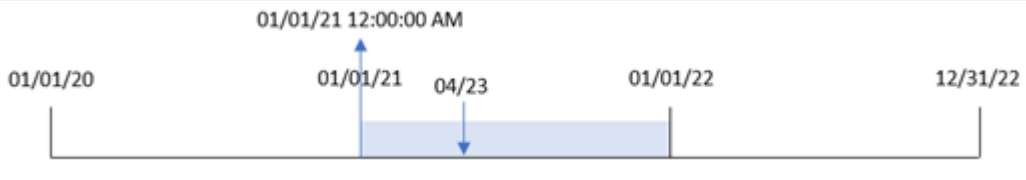
id	日期	=yearstart(date)	=timestamp(yearstart(date))
8188	06/06/2022	01/01/2022	1/1/2022 12:00:00 AM
8189	07/18/2022	01/01/2022	1/1/2022 12:00:00 AM
8190	11/14/2022	01/01/2022	1/1/2022 12:00:00 AM
8191	12/12/2022	01/01/2022	1/1/2022 12:00:00 AM
8192	01/22/2021	01/01/2021	1/1/2021 12:00:00 AM
8193	02/03/2021	01/01/2021	1/1/2021 12:00:00 AM
8194	03/17/2021	01/01/2021	1/1/2021 12:00:00 AM

id	日期	=yearstart(date)	=timestamp(yearstart(date))
8195	04/23/2021	01/01/2021	1/1/2021 12:00:00 AM
8196	05/04/2021	01/01/2021	1/1/2021 12:00:00 AM
8197	06/30/2021	01/01/2021	1/1/2021 12:00:00 AM
8198	07/26/2021	01/01/2021	1/1/2021 12:00:00 AM
8199	12/27/2021	01/01/2021	1/1/2021 12:00:00 AM
8200	01/13/2020	01/01/2020	1/1/2020 12:00:00 AM
8201	02/26/2020	01/01/2020	1/1/2020 12:00:00 AM
8202	03/27/2020	01/01/2020	1/1/2020 12:00:00 AM
8203	04/16/2020	01/01/2020	1/1/2020 12:00:00 AM
8204	05/21/2020	01/01/2020	1/1/2020 12:00:00 AM
8205	08/14/2020	01/01/2020	1/1/2020 12:00:00 AM
8206	10/07/2020	01/01/2020	1/1/2020 12:00:00 AM
8207	12/05/2020	01/01/2020	1/1/2020 12:00:00 AM

通过使用 yearstart()函数并将日期字段作为函数的参数传递，在图表对象中创建了 'start_of_year' 度量。

yearstart() 函数最初确定日期值属于哪一年，并返回该年第一毫秒的时间戳。

yearstart() 函数和交易 8199 的图表。



交易 8199 发生在 2021 年 4 月 23 日。yearstart() 函数返回该年的第一毫秒，即 1 月 1 日上午 12:00:00。

示例 5 – 场景

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 数据集加载到名为 'Loans' 的表中。表格包含以下字段：
  - 贷款 ID。
  - 年初余额。
  - 每笔贷款每年收取的简单利率。

最终用户希望有一个图表对象，该对象按贷款 ID 显示年初至今每笔贷款的应计利息。

加载脚本

```
Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- loan_id
- start_balance

要计算累计利息，请创建以下度量：

```
=start_balance*(rate*(today(1)-yearstart(today(1)))/365)
```

将度量的**数字格式**设置为**金额**。

结果表

loan_id	start_balance	=start_balance*(rate*(today(1)-yearstart(today(1)))/365)
8188	$10000.00	$39.73
8189	$15000.00	$339.66
8190	$17500.00	$166.85
8191	$21000.00	$283.64
8192	$90000.00	$3003.29

yearstart() 函数使用今天的日期作为唯一参数，返回当前年份的开始日期。通过从当前日期中减去该结果，表达式将返回今年迄今为止经过的天数。

然后将该值乘以利率并除以 365，以返回该期间的实际利率。然后将该期间的实际利率乘以贷款的起始余额，以返回今年迄今为止的应计利息。

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)。

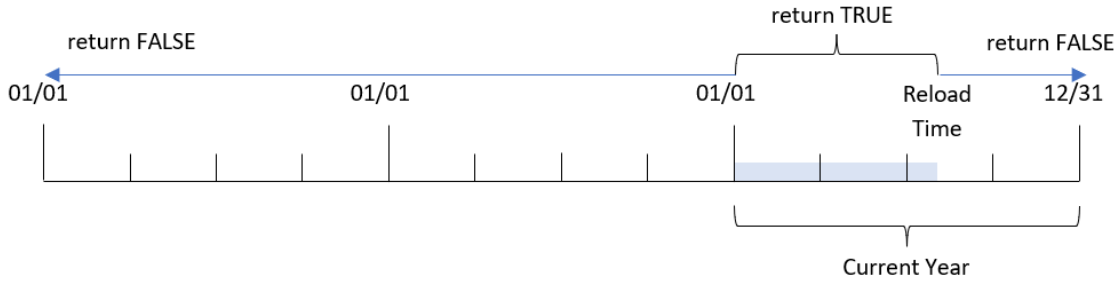
语法：

```
YearToDate(timestamp[ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

返回数据类型：布尔值

> *在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。*

yeartodate() 函数的示例图表



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。

换言之，当在没有其他参数的情况下触发 yeartodate() 函数时，该函数用于评估时间戳，并根据该日期是否发生在日历年内(包括重新加载发生的日期)返回布尔结果。

但是，也可以使用 firstmonth 参数取代年份的开始日期，以及使用 yearoffset 参数与前几年或后几年进行比较。

最后，在历史数据集的实例中，yeartodate() 函数提供了一个要设置 todaydate 的参数，该参数会将时间戳与 todaydate 参数中提供的日期(包括该日期)之前的日历年进行比较。

参数

参数	说明
timestamp	评估的时间戳，如 '10/12/2012'.。
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。

参数	说明
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。值 1 表示从 1 月 1 日开始的财政年度,值 12 表示从 12 月 1 日起的财政年度。
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。

适用场景

yeartodate() 函数返回布尔结果。通常,这种类型的函数将用作 if 表达式中的条件。这将返回一个聚合或计算,取决于评估日期是否发生在应用程序的最后一个重新加载日期之前(包括该日期)。

例如,YearToDate() 函数可用于标识当前年份迄今为止制造的所有设备。

下例假设上次重新加载时间 = 11/18/2011。

函数示例

示例	结果
`yeartodate( '11/18/2010')`	返回 False
`yeartodate( '02/01/2011')`	返回 True
`yeartodate( '11/18/2011')`	返回 True
`yeartodate( '11/19/2011')`	返回 False
`yeartodate( '11/19/2011', 0, 1, '12/31/2011')`	返回 True
`yeartodate( '11/18/2010', -1)`	返回 True
`yeartodate( '11/18/2011', -1)`	返回 False
`yeartodate( '04/30/2011', 0, 5)`	返回 False
`yeartodate( '05/01/2011', 0, 5)`	返回 True

区域设置

除非另有规定,本主题中的示例使用以下日期格式:MM/DD/YYYY。日期格式已经在数据加载脚本中的 `SET DateFormat` 语句中指定。由于区域设置和其他因素,系统中的默认日期格式可能有所不同。您可以更改以下示例中的格式以满足您的要求。或者,您可以更改加载脚本中的格式以匹配这些示例。

应用程序中的默认区域设置基于安装 Qlik Sense 的计算机或服务器的区域系统设置。如果您访问的 Qlik Sense 服务器设置为瑞典,则数据加载编辑器将使用瑞典地区设置的日期、时间和货币。这些区域格式设置与 Qlik Sense 用户界面中显示的语言无关。Qlik Sense 将以与您使用的浏览器相同的语言显示。

示例 1 – 基本示例

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。
- 创建一个字段 year_to_date，用于确定截至上次重新加载日期的日历年中发生了哪些交易。

在撰写本文时，日期为 2022 年 4 月 26 日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        yeartodate(date) as year_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

结果

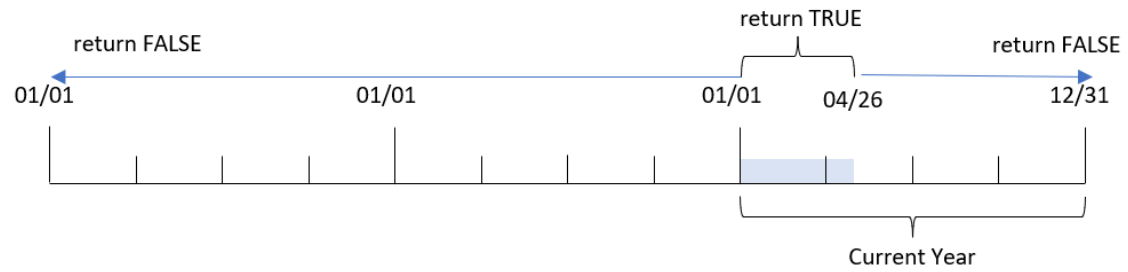
加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- year_to_date

结果表

日期	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

yeartodate() 函数的图表，基本示例



通过使用 yeartodate() 函数并将 date 字段作为函数的参数传递，在前置 Load 语句中创建了 year_to_date 字段。

由于没有进一步的参数被传递到函数中，因此 yeartodate() 函数最初会标识重新加载日期，从而标识当前日历年(从 1 月 1 日开始)的边界，并返回 TRUE 的布尔值结果。

因此，在 1 月 1 日至 4 月 26 日(重新加载日期)之间发生的任何交易都将返回 TRUE 的布尔值结果。2022 开始之前发生的任何交易都将返回 FALSE 的布尔值结果。

示例 2 – yearoffset

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 two_years_prior，用于确定哪些交易在日历年之前整整两年发生。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        yeartodate(date,-2) as two_years_prior
        ;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
```

```
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- two_years_prior

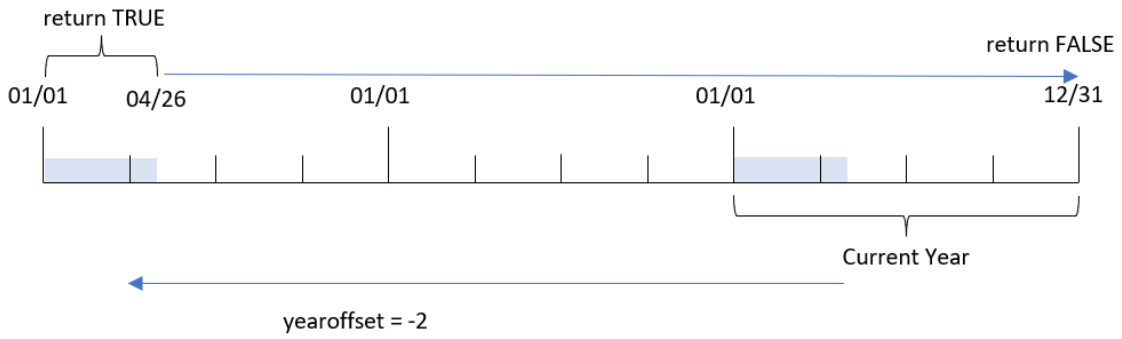
结果表

日期	two_years_prior
01/10/2020	-1
02/28/2020	-1
04/09/2020	-1
04/16/2020	-1
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0

日期	two_years_prior
07/26/2021	0
12/27/2021	0
02/02/2022	0
02/26/2022	0
03/07/2022	0
03/11/2022	0

通过在 yeartodate() 函数中将 -2 用作 yearoffset 参数，该函数将比较器日历年份段的边界移动了整两年。最初，年份段等于 2022 年 1 月 1 日至 4 月 26 日。然后，yearoffset 参数将这一段偏移到两年前。日期边界将在 2020 年 1 月 1 日至 4 月 26 日之间。

yeartodate() 函数的图表，yearoffset 示例



因此，2020 年 1 月 1 日至 4 月 26 日之间发生的任何交易都将返回布尔值结果 TRUE。在此段之前或之后出现的任何交易都将返回 FALSE。

示例 3 – firstmonth

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 year_to_date，用于确定截至上次重新加载日期的日历年中发生了哪些交易。

在本例中，我们将会计年度的开始日期设置为 7 月 1 日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        yeartodate(date,0,7) as year_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- date
- year_to_date

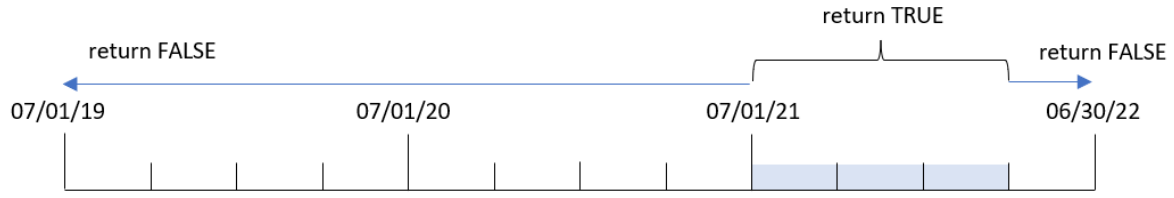
结果表

日期	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0

日期	year_to_date
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	-1
12/27/2021	-1
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

在本例中，由于 yeartodate() 函数中使用了为 7 的 firstmonth 参数，因此它将一年中的第一天设置为 7 月 1 日，将一年的最后一天设置为 6 月 30 日。

yeartodate() 函数的图表，firstmonth 示例



因此，在 2021 年 7 月 1 日至 2022 年 4 月 26 日(重新加载日期)之间发生的任何交易都将返回 TRUE 的布尔值结果。2021年 7 月 1 日之前发生的任何交易都将返回 FALSE 的布尔值结果。

示例 4 – todaydate

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 与第一个示例相同的数据集和场景。
- 创建一个字段 year_to_date，用于确定截至上次重新加载日期的日历年中发生了哪些交易。

然而，在本例中，我们需要确定截至 2022 年 3 月 1 日（含）的日历年内发生的所有交易。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
        *,
        yeartodate(date, 0, 1, '03/01/2022') as year_to_date
;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

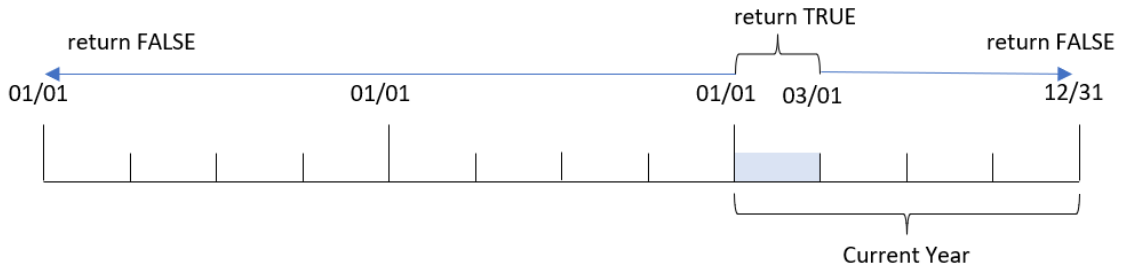
- date
- year_to_date

结果表

日期	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	0
03/11/2022	0

在这种情况下，由于 yeartodate() 函数中使用了为 03/01/2022 的 todaydate 参数，它将比较日历年段的结束边界设置为 2022 年 3 月 1 日。提供 firstmonth 参数（介于 1 和 12 之间）至关重要；否则函数将返回空结果。

yeartodate() 函数的图表，使用 todaydate 参数的示例



因此，对于在 2022 年 1 月 1 日和 2022 年 3 月 1 日之间发生的任何交易，todaydate 参数都将返回 TRUE 的布尔值结果。2022 年 1 月 1 日之前或 2022 年 3 月 1 日之后发生的任何交易都将返回 FALSE 的布尔值结果。

示例 5 – 图表对象示例

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含与第一个示例相同的数据集和场景。

然而，在本例中，未更改的数据集被加载到应用程序中。确定截至上次重新加载日期的日历年中发生的交易的计算将在应用程序的图表对象中创建为度量。

加载脚本

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
```

```
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度：date。

添加以下度量：

```
=yeartodate(date)
```

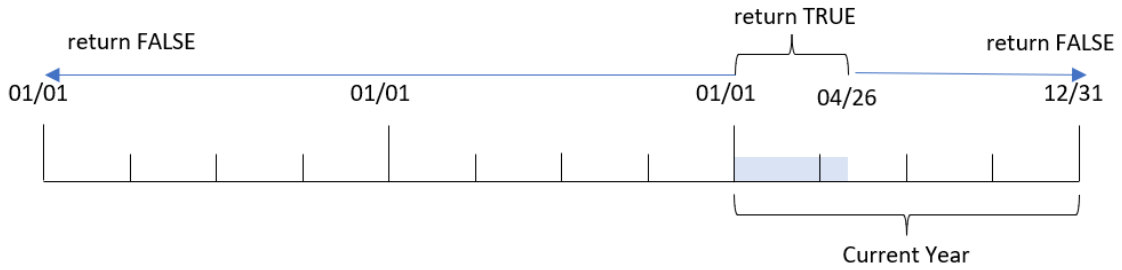
结果表

日期	=yeartodate(date)
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

通过使用 yeartodate() 函数并将 date 字段作为函数的参数传递，在图表对象中创建 year_to_date 度量。

由于没有进一步的参数被传递到函数中，因此 yeartodate() 函数最初会标识重新加载日期，从而标识当前日历年(从 1 月 1 日开始)的边界，并返回 TRUE 的布尔值结果。

yeartodate() 函数的图表，使用图表对象的示例



在 1 月 1 日至 4 月 26 日(重新加载日期)之间发生的任何交易都将返回 TRUE 的布尔值结果。2022 开始之前发生的任何交易都将返回 FALSE 的布尔值结果。

示例 6 – 场景

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 包含 2020 年至 2022 年间交易集的数据集，该数据集加载到名为 Transactions 的表中。
- 日期字段已以 DateFormat 系统变量 (MM/DD/YYYY) 格式提供。

最终用户需要一个 KPI 对象，该对象将 2021 等效期间的总销售额表示为截至上次重新加载时的当前年份。

在撰写本文时，日期为 2022 年 6 月 16 日。

加载脚本

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
```

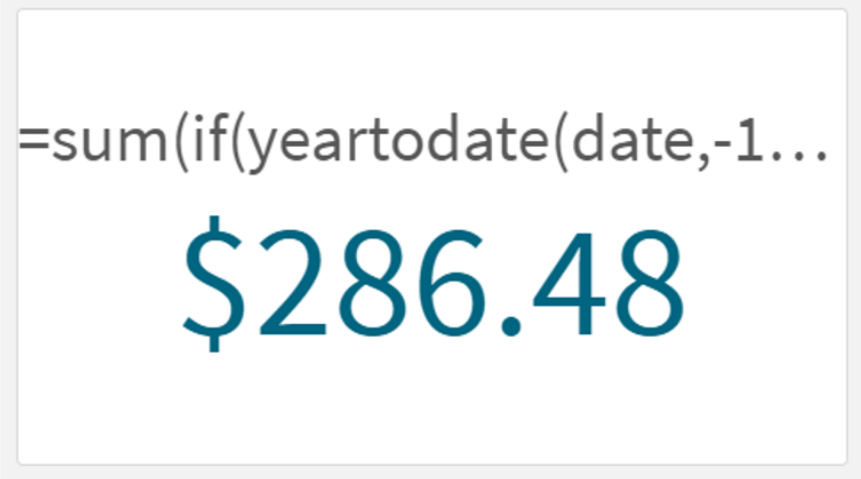
```
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

结果

执行以下操作：

1. 创建 KPI 对象。
2. 创建以下聚合度量以计算总销售额：
   `=sum(if(yeartodate(date,-1),amount,0))`
3. 将度量的**数字格式**设置为**金额**。

2021 年的 KPI `yeartodate()` 图表



yeartodate() 函数在计算每个事务 ID 的日期时返回布尔值。由于重新加载发生在 2022 年 6 月 16 日，yeartodate 函数将该年期间分段成介于 01/01/2022 和 06/16/2022。但是，由于函数中使用的 period_no 值为 -1，因此这些边界将移到上一年。因此，对于 01/01/2021 和 06/16/2021 之间发生的任何交易，yeartodate() 函数返回布尔值 TRUE 并对金额求和。

8.8 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

8.9 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的相异值的数量。仅计算排除的(深灰色)字段。备选值(浅灰色)和选定的排除值(带复选标记的深灰色)不计算在内。

```
GetExcludedCount - 图表函数(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])
```

GetStateCounts

GetStateCounts() 图表函数用于计算与指定选择状态匹配的唯一值的总数。

```
GetStateCounts - 图表函数 (field_name, state_name [, state_type1,...state_typeN])
```

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

> *选择栏中使用的颜色以及每个选择项状态都可以使用自定义主题进行修改。如果您所用的是使用自定义主题的应用程序，您可能会注意到您的选择显示的颜色与帮助主题中描述的颜色不同。*

参数：

参数

参数	描述
field_name	包含要度量的数据范围的字段。

下表列出了与此函数相关的其他函数。

相关函数

函数	交互
GetStateCounts - 图表函数 (page 1149)	使用 **GetStateCounts()**，您可以使用单个函数调用组合以下计数的计算： • 所选包含值的计数。 • 可能的值计数。 • 替代值的计数。 • 排除值的计数，不包括替代和选定的排除值。 • 所选排除值的计数。
GetSelectedCount - 图表函数 (page 1146)	返回所选包含值的计数。
GetPossibleCount - 图表函数 (page 1144)	返回可能值的计数。
GetAlternativeCount - 图表函数 (page 1134)	返回排除值的计数，不包括替代和选定的排除值。

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的相异值的数量。仅计算排除的(深灰色)字段。备选值(浅灰色)和选定的排除值(带复选标记的深灰色)不计算在内。

语法:

```
GetExcludedCount (field_name)
```

返回数据类型: 字符串

> *选择栏中使用的颜色以及每个选择项状态都可以使用自定义主题进行修改。如果您所用的是使用自定义主题的应用程序, 您可能会注意到您的选择显示的颜色与帮助主题中描述的颜色不同。*

参数

参数	描述
field_name	包含要度量的数据范围的字段。

下表列出了与此函数相关的其他函数。

相关函数

函数	交互
GetStateCounts - 图表函数 (page 1149)	使用 **GetStateCounts()**，您可以使用单个函数调用组合以下计数的计算： • 所选包含值的计数。 • 可能的值计数。 • 替代值的计数。 • 排除值的计数，不包括替代和选定的排除值。 • 所选排除值的计数。
GetSelectedCount - 图表函数 (page 1146)	返回所选包含值的计数。
GetPossibleCount - 图表函数 (page 1144)	返回可能值的计数。
GetAlternativeCount - 图表函数 (page 1134)	返回替代值的计数。

示例和结果：

将下面的示例脚本加载到应用程序中后，创建三个筛选器窗格：一个用于 **First name**，一个用于 **Last name**，一个用于 **Initials**。表中的每个示例表达式都可以添加为 KPI 图表。

示例和结果

示例	结果
假定未在 **First name** 中选择任何值。 `GetExcludedCount (Initials)`	结果为 0，因为没有选择。
假定选择 **John**（在 **First name** 中）。 `GetExcludedCount (Initials)`	结果为 5。**首字母**中有 5 个排除的值，颜色为深灰色。**JA** 值将为白色，因为它与 **First name** 中的选择 **John** 相关联。
假定已选择 **John** 和 **Peter**。 `GetExcludedCount (Initials)`	结果为 3。**John** 与 1 个值相关联，**Peter** 与 **Initials** 中的 2 个值相关联。

示例	结果
假设 **First name** 中选择了 **John** 和 **Peter**，然后在 **Last name** 中选择 **Franc**。 `GetExcludedCount ([First name])`	结果为 3。**First name** 中有 3 个排除值，颜色为深灰色。**GetExcludedCount()** 仅计算排除的值。备选和选定的排除值不包括在计数中。
假设 **First name** 中选择了 **John** 和 **Peter**，然后在 **Last name** 中选择 **Franc** 和 **Anderson**。 `GetExcludedCount (Initials)`	结果为 4。**Initials**中有 4 个排除的值，颜色为深灰色。其他两个值（**JA** 和 **PF**）将为白色，因为它们与 **First name** 中的选择 **John** 和 **Peter** 相关联。
假设 **First name** 中选择了 **John** 和 **Peter**，然后在 **Last name** 中选择 **Franc** 和 **Anderson**。 `GetExcludedCount ([Last name])`	结果为 3。**Last name**中有 3 个排除的值，颜色为深灰色。**Brown**、**Carr** 和 **Elliot**。值 **Devonshire** 为浅灰色（表示它是可选的），因此不包括在计数中。

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	说明
'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。

格式	说明
'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”

示例	结果
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	描述
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。

下表列出了与此函数相关的其他函数。

相关函数

函数	交互
GetStateCounts - 图表函数 (page 1149)	使用 **GetStateCounts()**，您可以使用单个函数调用组合以下计数的计算： • 所选包含值的计数。 • 可能的值计数。 • 取消选择的值的计数(仅当字段处于“和”模式时可用)。 • 替代值的计数。 • 排除值的计数，不包括替代和选定的排除值。 • 所选排除值的计数。
GetSelectedCount - 图表函数 (page 1146)	返回所选包含值的计数。
GetPossibleCount - 图表函数 (page 1144)	返回可能值的计数。
GetAlternativeCount - 图表函数 (page 1134)	返回替代值的计数。
GetExcludedCount - 图表函数 (page 1137)	返回排除值的计数，不包括替代和选定的排除值。

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。

不能在以下位置的图表中使用此函数：标题、副标题、页脚、参考线表达式和最小/最大表达式。

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	CustomerID	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。

不能在以下位置的图表中使用此函数:标题、副标题、页脚、参考线表达式和最小/最大表达式。

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。

不能在以下位置的图表中使用此函数：标题、副标题、页脚、参考线表达式和最小/最大表达式。

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

CustomerID	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)
```

选择栏中使用的颜色以及每个选择项状态都可以使用自定义主题进行修改。如果您所用的是使用自定义主题的应用程序，您可能会注意到您的选择显示的颜色与帮助主题中描述的颜色不同。

参数：

参数

参数	描述
field_name	包含要度量的数据范围的字段。

下表列出了与此函数相关的其他函数。

相关函数

函数	交互
GetStateCounts - 图表函数 (page 1149)	使用 **GetStateCounts()**，您可以使用单个函数调用组合以下计数的计算： • 所选包含值的计数。 • 可能的值计数。 • 替代值的计数。 • 排除值的计数，不包括替代和选定的排除值。 • 所选排除值的计数。
GetSelectedCount - 图表函数 (page 1146)	返回所选包含值的计数。
GetAlternativeCount - 图表函数 (page 1134)	返回替代值的计数。
GetPossibleCount - 图表函数 (page 1144)	返回排除值的计数，不包括替代和选定的排除值。

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。

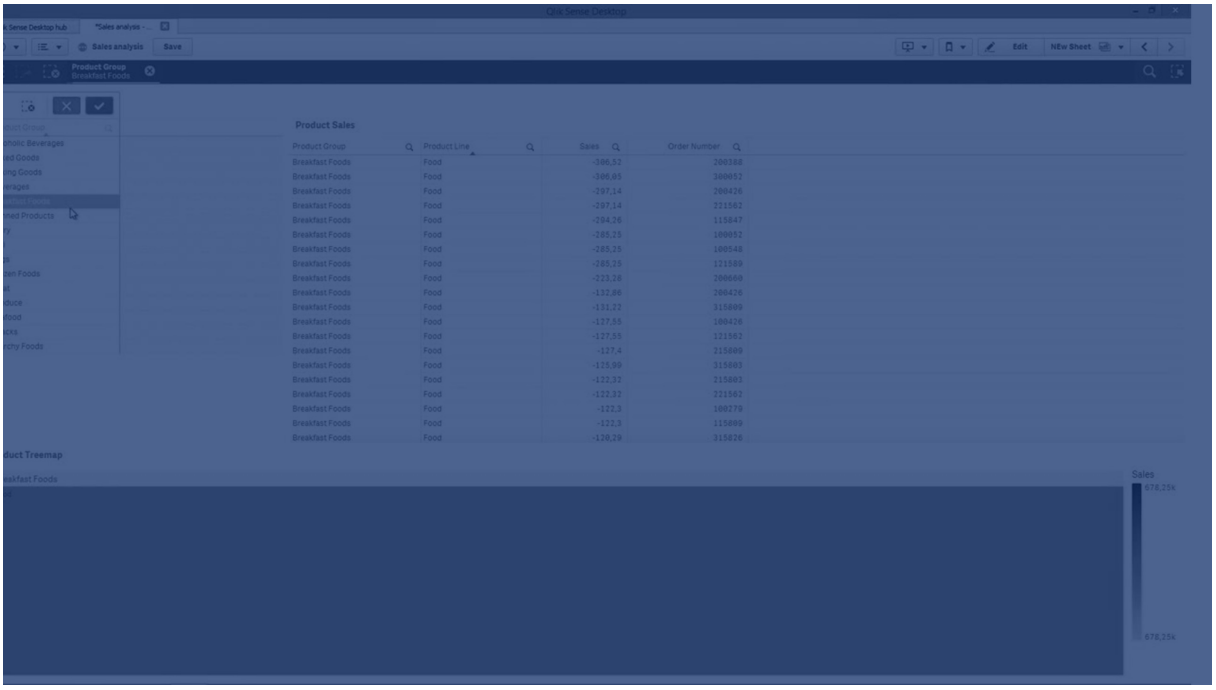
示例	结果
假定选择 **John**(在 **First name** 中)。 `GetPossibleCount ([First name])`	1, 因为 **First name** 中有 1 个选择项 **John**。
假定选择 **Peter**(在 **First name** 中)。 `GetPossibleCount ([Initials])`	2, 因为 Peter 与 **Initials** 中的 2 个值关联。
假定未在 **First name** 中选择任何值。 `GetPossibleCount ([First name])`	5, 因为没有选择项, 并且 **First 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

> *选择栏中使用的颜色以及每个选择项状态都可以使用自定义主题进行修改。如果您所用的是使用自定义主题的应用程序，您可能会注意到您的选择显示的颜色与帮助主题中描述的颜色不同。*

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。

下表列出了与此函数相关的其他函数。

相关函数

函数	交互
GetStateCounts - 图表函数 (page 1149)	使用 **GetStateCounts()**，您可以使用单个函数调用组合以下计数的计算： • 所选包含值的计数。 • 可能的值计数。 • 替代值的计数。 • 排除值的计数，不包括替代和选定的排除值。 • 所选排除值的计数。
GetPossibleCount - 图表函数 (page 1144)	返回可能值的计数。
GetAlternativeCount - 图表函数 (page 1134)	返回替代值的计数。
GetSelectedCount - 图表函数 (page 1146)	返回排除值的计数，不包括替代和选定的排除值。

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 **MC** 和 **PD** 的 **Initials** 值的选择项的 **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 '|');
```

GetStateCounts - 图表函数

GetStateCounts() 图表函数用于计算与指定选择状态匹配的唯一值的总数。

通过 **GetStateCounts()**，您可以将以下函数的计算合并到一个函数调用中：**GetSelectedCount()**、**GetNotSelectedCount()**、**GetAlternativeCount()**、**GetPossibleCount()** 和 **GetExcludedCount()**。所选排除值的计数也可添加到计算中。您可以指定每个函数计算是对返回的总和进行相加还是相减。

语法：

```
GetStateCounts (field_name, state_name [, state_type1,...state_typeN])
```

返回数据类型： 整数

参数

参数	描述
field_name	您正在计算选择项状态的字段。不存在的字段名会导致空结果。
state_name	备用状态的名称。如果参数为空 ('') 或 null，则使用继承的备用状态。使用 $ 来显式使用默认状态。与现有状态不匹配的命名（非空）状态名称将导致 null 结果。
state_type	字段值的一个或多个状态类型的列表。这些状态类型将被聚合到一个计数中。状态类型使用键来指定。用单引号输入每个键。 当省略此参数时，函数将返回一个字符串，其中包含字段的所有可用状态计数，顺序与枚举相同。 有关可以使用的状态列表，请参阅下表。

使用特定键引用状态类型。您可以使用键的数字或文本版本。在同一表达式中组合多个键以进一步自定义结果。您可以从总数中减去状态计数，而不是将其相加。为此，请使用文本键并在状态类型前加上减号 (-)。

状态类型以及减号（如果适用）需要用一组单引号括起来。

每个字段状态类型的键

字段状态类型	描述	数字键	文本键
已选择	在计算中包括选定值。有关等效函数，请参阅 *GetSelectedCount - 图表函数 (page 1146)*。	1	S
可选	在计算中包括可选（未选择，但可以选择）值。有关等效函数，请参阅 *GetPossibleCount - 图表函数 (page 1144)*。	2	O

字段状态类型	描述	数字键	文本键
取消选择	在计算中包括未选定值。此状态类型仅在字段处于“和”模式时可用。 假设该函数中的 **include_excluded** 参数设置为默认值 False，则此状态类型返回与 **GetNotSelectedCount()** 函数返回的相同的计算结果。有关 **GetNotSelectedCount()** 的更多信息，请参阅 *GetNotSelectedCount - 图表函数 (page 1141)*。	3	D
替代	在计算中包括替代值。有关等效函数，请参阅 *GetAlternativeCount - 图表函数 (page 1134)*。	4	A
排除	在计算中包括排除(未选定)值。有关等效函数，请参阅 *GetExcludedCount - 图表函数 (page 1137)*。	5	X
选定排除	在计算中包括选定排除值。	6	XS

适用场景

通过 **GetStateCounts()**，您可以计算自定义选择项状态。该函数允许您将多个函数调用合并为一个函数调用，简化了编写表达式的过程。

例如，您可能需要计算字段的排除值、替代值和选定排除值的总数。您可使用 **GetStateCounts()** 计算这一合计。

示例和结果

示例	结果
`=GetStateCounts(ProductName, Null(), 'S')`	以继承的替代状态返回 *ProductName* 的选定计数。
`=GetStateCounts(ProductName, '', 'X', 'A', 'XS')`	返回 *ProductName* 的排除值、选定排除值和替代值的总数。使用继承的替代状态。
`=GetStateCounts(ProductName, '', 'S', 'XS')`	返回处于继承状态的 *ProductName* 的用户选择项的总数。
假定 *ProductName* 字段处于“和”模式。 `=GetStateCounts(ProductName, '', 'D', '-O')`	返回的 *ProductName* 未选定值的数量减去可能值的数量。使用继承的替代状态。
`=GetStateCounts(ProductName, '', 'X', , 'A', 'XS')`	返回 *ProductName* 的排除值、选定排除值和替代值的总数。使用继承的替代状态。
`=GetStateCounts(ProductName, '$', 'O')`	在默认备用状态下返回 *ProductName* 的可能计数。
`=GetStateCounts(ProductName, 'StateA', 'S')`	返回在名为 *StateA* 的替代状态下的 *ProductName* 选定计数。

示例 1 - 计算用户选择项的总数(包括选定的排除值)

图表表达式和结果

概述

GetSelectedCount() 函数返回选定值的总数，不包括选定排除值的总数。通过 **GetExcludedCount()**，您可以在图表的计算中纳入选定排除值的计数。

打开 数据加载编辑器，并将以下加载脚本添加到新选项卡。

加载脚本

```
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 '|');
```

结果

执行以下操作：

1. 加载数据并打开工作表。
2. 创建筛选器窗格
3. 在第一个筛选窗格中添加 **First name** 为维度。在第二个筛选窗格中添加 **Last name** 为维度。
4. 下一步，用这个度量创建 KPI：
   `=GetStateCounts([First name], '', 'X', 'A', 'XS')`
5. 在 **First name** 筛选器窗格中，选择值 **Mark** 和 **Peter**。在 **Last name**筛选器窗格中，选择值 **Carr**。
6. 观察到 KPI 显示值 2。这反映了您选择的所有值，包括所选的排除值 **Peter**，都将被计算在内。

示例 2 - 结合排除、选定排除和替代计数

图表表达式和结果

概述

GetExcludedCount() 函数返回未选定和排除的唯一值的总数。如果要在可视化中包含选定的排除计数和替代计数，可以按如下方式使用 **GetStateCounts()** 函数。

打开 数据加载编辑器，并将以下加载脚本添加到新选项卡。

加载脚本

```
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 '|');
```

结果

执行以下操作:

1. 加载数据并打开工作表。
2. 创建筛选器窗格
3. 在第一个筛选窗格中添加 **First name** 为维度。在第二个筛选窗格中添加 **Last name** 为维度。
4. 下一步,用这个度量创建 KPI:
   `=GetStateCounts([First name], '', 'X', 'A', 'XS')`
5. 在 **First name** 筛选器窗格中,选择值 **John** 和 **Peter**。在 **Last name**筛选器窗格中,选择值 **Franc**。
6. 观察到 KPI 显示值 4。这反映了除了排除值的计数外,选定排除值 **John** 也包含在计数中。
7. 编辑 KPI 并用以下内容替换现有度量:
   `=GetStateCounts([Last name], '', 'X', 'A', 'XS')`
8. 保留现有选择项,在 **Last name** 筛选器窗格中选择值 **Anderson**。
9. 观察到 KPI 仍显示值 4。这反映了除了排除值的计数外,替代值 **Devonshire** 也包含在计数中。

8.10 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 函数以 UTC 格式返回指定文件上次修改的时间戳。如果未指定文件，函数将返回当前读取的表文件的最后修改的 UTC 时间戳。

```
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。在时间戳中,时间以 UTC 表示。

```
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*** • 相对于 Qlik Sense 应用程序工作目录。 **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 1449)

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	作为文件夹或 web 文件数据连接的文件名（如有必要，包括路径）。如果未指定文件名，则使用当前正在读取的表文件。 **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*** • 相对于 Qlik Sense 应用程序工作目录。 **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 函数以 UTC 格式返回指定文件上次修改的时间戳。如果未指定文件，函数将返回当前读取的表文件的最后修改的 UTC 时间戳。

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*** • 相对于 Qlik Sense 应用程序工作目录。 **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记录读取的 X 字段中返回文件 (abc.txt) 上一次修改的时间戳。
`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。在时间戳中,时间以 UTC 表示。

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*** • 相对于 Qlik Sense 应用程序工作目录。 **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*** • 相对于 Qlik Sense 应用程序工作目录。 **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*** • 相对于 Qlik Sense 应用程序工作目录。 **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*** • 相对于 Qlik Sense 应用程序工作目录。 **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*** • 相对于 Qlik Sense 应用程序工作目录。 **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')
```

8.11 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。

参数	说明
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
`BlackAndScholе(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。默认情况下，结果将被格式化为货币。.

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。默认情况下，结果将被格式化为货币。.

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。默认情况下，结果将被格式化为货币。.

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。

参数	说明
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%

8.12 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 1211)

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 159)*

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

所有双返回值都右对齐。

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	说明
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。

No formatting	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。

Discount	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

8.13 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

示例	结果
`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...
`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

8.14 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;
```

8.15 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#(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#(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#(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#(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#(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# (text[, format])
```

另请参见：

- 格式函数 (page 1178)

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	说明
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

A	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：图表表达式

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 [
String
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

8.16 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)。

当在图表的任何表达式中使用记录间图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用记录间图表功能时，可视化的排序将返回到记录间功能的排序输入。此限制不适用于等效的脚本函数(如果有)。

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 (expr)
```

另请参见：

范围函数 (page 1275)

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。
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 1275)*
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。
- 当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 1226)*
- *Bottom - 图表函数 (page 1229)*
- *Top - 图表函数 (page 1257)*
- *RangeAvg (page 1277)*

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。
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 1275)*
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。
- 当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 1222)*
- *Bottom - 图表函数 (page 1229)*
- *Top - 图表函数 (page 1257)*
- *RangeAvg (page 1277)*

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 1275)*
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。
- 当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 1257)*

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：

- 递归调用将返回 NULL 值。
- 如果 **ColumnNo** 引用没有度量的列，则返回 NULL 值。
- 当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
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 1233)* 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 1234)*。

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。

当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 ',');
```

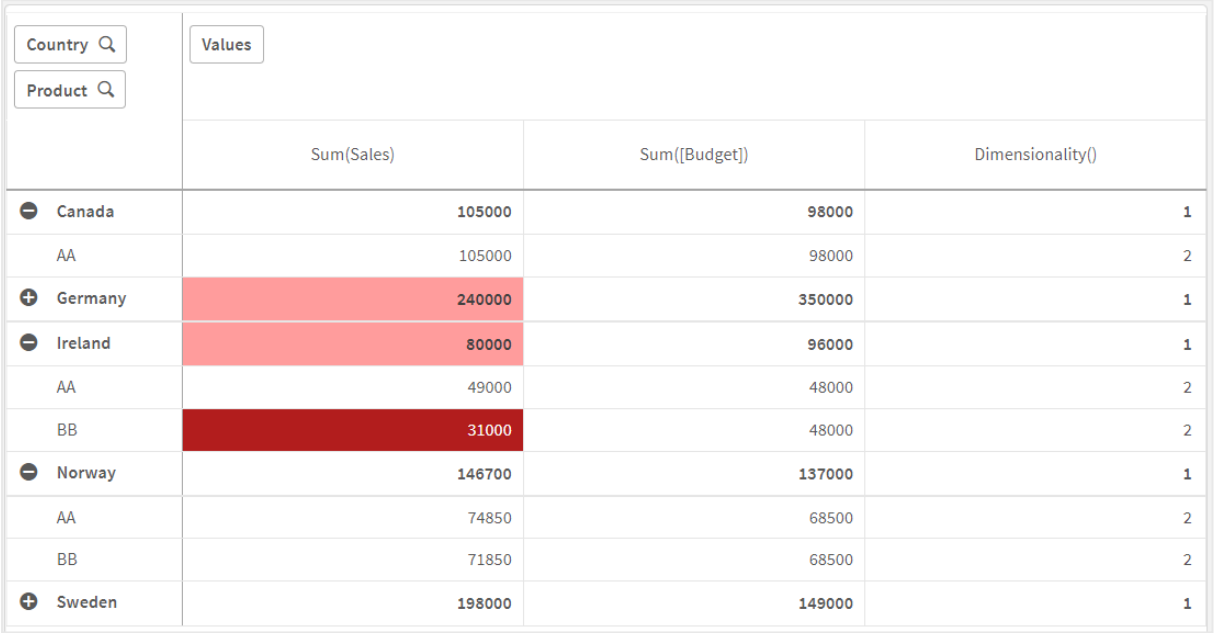
图表表达式

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，您可借助维度Employee 和 Address 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 表的前置 Load。
  `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。
- 当在图表的任何表达式中使用此图表函数时,不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此,这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时,可视化的排序将返回到此函数的排序输入。此限制不适用于等效的脚本函数。

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。
脚本函数:在加载表格 **Names** 作为示例数据的情况下:	

示例	结果
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。
- 当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。此限制不适用于等效的脚本函数。

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|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 FieldValue('First name',1) as MyPos1
Resident 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。

> *当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。此限制不适用于等效的脚本函数。*

语法：

```
FieldValueCount(field_name)
```

返回数据类型： 整数

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。

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。

示例	结果
`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)。

参数	说明
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 1268)*

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。

当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 556)

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

月	产品	金额	AmountMonthBefore	AmountMonthAfter
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
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

月	销售额值	增大
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 1275)*
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。
- 当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 1229)*

Above - 图表函数 (page 1222)
Sum - 图表函数 (page 334)
RangeAvg (page 1277)
范围函数 (page 1275)

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。
- 当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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 ]])
```

当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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。

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。

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。

当在图表的任何表达式中使用此图表函数时,不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此,这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时,可视化的排序将返回到此函数的排序输入。

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。

> *当在图表的任何表达式中使用此图表函数时,不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此,这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时,可视化的排序将返回到此函数的排序输入。*

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。

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。
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。

> *当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。*

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。

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 **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.

当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。

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。

> *当在图表的任何表达式中使用此图表函数时，不允许对图表中的 y 值进行排序或按表中的表达式列进行排序。因此，这些排序替代项会自动禁用。当您在可视化或表格中使用此图表函数时，可视化的排序将返回到此函数的排序输入。*

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 )))
```

8.17 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

8.18 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。禁用的示例：`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
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

8.19 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( )
```

8.20 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

ID	Value	ValueNullConv	IsItNull
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
2	-	-	T
3	Value	Value	F

8.21 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() 用于返回现金流时间表的内部回报率(每年)。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() 用于返回通过 **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)的净现值。所有付款按 365 天一年年折扣。

```
RangeXNPV (discount_rate, values, dates[, Expression])
```

另请参见：

内部记录函数 (page 1218)

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 383)*
- *Count - 图表函数 (page 338)*

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 '|');
```

在以 ID1 作维度和度量 RangeCorrel(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6))的表中，**RangeCorrel()** 函数在六对 x,y 中查找 **Correl** 值，以查找每个 ID1 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 387)

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 338)*

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 1222)*
- *Fractile - 图表函数 (page 390)*

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。

该函数使用牛顿法的简化版本来计算内部回报率 (IRR)。

语法：

```
RangeIRR(value[, value][, Expression])
```

返回数据类型： 数字

参数

参数	描述
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

示例	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。
`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 1218)*

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 397)*

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
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 509)

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 326)

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 512)

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 341)

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	-

MyField	RangeMode(Above(MyField,0,3))
5	-
9	-

另请参见：

- *Mode - 图表函数 (page 329)*

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 1311)*。

语法：

```
RangeNPV(discount_rate, value[,value][, Expression])
```

返回数据类型： 数字

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

示例	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。
`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 1218)

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 344)

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 347)

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 331)

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 427)*

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

示例	结果
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 429)

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 334)*
- *Above - 图表函数 (page 1222)*

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 350)

RangeXIRR

RangeXIRR() 用于返回现金流时间表的内部回报率(每年)。要计算一系列周期性现金流的内部回报率，请使用 **RangeIRR** 函数。

Qlik 的 XIRR 函数(**XIRR()** 和 **RangeXIRR()** 函数)使用以下方程来求解 `Rate` 值，以确定正确的 XIRR 值：

```
XNPV(Rate, pmt, date) = 0
```

这个方程是用简化版的牛顿法求解的。

语法：

```
RangeXIRR(value, date{, value, date})
```

返回数据类型： 数字

参数

参数	描述
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。

使用此功能时，以下限制适用：

- 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。
- 此函数需要至少一个有效的负付款和至少一个无效的正付款（具有相应的有效日期）。如果没有提供这些付款，则会返回 NULL 值。

以下主题可以帮助您使用此函数：

- *RangeXNPV (page 1311)*：使用此函数计算不一定是周期性的现金流表的净现值。
- *XIRR (page 363)*：**XIRR()** 函数计算现金流表（不必是周期性的）的聚合内部回报率（年）。

在 Qlik Sense 客户端托管 的不同版本中，此函数使用的底层算法存在差异。有关算法最近更新的更多信息，请参阅支持文章，请参阅支持文章 XIRR 函数修复和更新。

示例和结果：

示例和结果

示例	结果
`RangeXIRR(-2500,'2008-01-01',2750,'2008-09-01')`	返回 0.1532

另请参见：

- *RangeIRR (page 1286)*
- *RangeXNPV (page 1311)*
- *XIRR (page 363)*
- XIRR 函数修复和更新

RangeXNPV

RangeXNPV() 用于返回通过 **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）的净现值。所有付款按 365 天一年年折扣。

语法：

```
RangeXNPV(discount_rate, value, date{, value, date})
```

返回数据类型： 数字

参数

参数	说明
discount_rate	**discount_rate** 是付款折扣时依据的年费率。
value	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。
date	对应现金流支付的付款日期或付款日期计划表。

使用此功能时，以下限制适用：

- 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 - 脚本

加载脚本和结果

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名为 RangeTab3 的表中包含的财务数据。
- 使用 **RangeXNPV()** 函数计算净现值。

加载脚本

```
RangeTab3:
LOAD *,
recno() as RangeID,
RangeXNPV(DiscountRate,Value1,Date1,Value2,Date2) as RangeXNPV;
LOAD * INLINE [
DiscountRate|Value1|Date1|Value2|Date2
0.1|-100|2021-01-01|100|2022-01-01|
0.1|-100|2021-01-01|110|2022-01-01|
0.1|-100|2021-01-01|125|2022-01-01|
] (delimiter is '|');
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- RangeID
- RangeXNPV

结果表

RangeID	RangeXNPV
1	-$9.09
2	-$0.00
3	$13.64

示例 - 图表表达式

加载脚本和图表表达式

概述

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

- 名为 RangeTab3 的表中包含的财务数据。
- 使用 **RangeXNPV()** 函数计算净现值。

加载脚本

```
RangeTab3:
LOAD *,
recno() as RangeID,
RangeXNPV(DiscountRate,Value1,Date1,Value2,Date2) as RangeXNPV;
LOAD * INLINE [
DiscountRate|Value1|Date1|Value2|Date2
0.1|-100|2021-01-01|100|2022-01-01|
0.1|-100|2021-01-01|110|2022-01-01|
0.1|-100|2021-01-01|125|2022-01-01|
] (delimiter is '|');
```

结果

执行以下操作：

加载数据并打开工作表。创建新表并添加以下计算作为度量。

```
=RangeXNPV(0.1, -2500,'2008-01-01',2750,'2008-09-01')
```

结果表

=XIRR(Payments, Date)
$80.25

另请参见:

- *XNPV (page 369)*

8.22 关系函数

这是一组函数,使用已聚合的数字计算图表中各个维度值的属性。

函数是关系型的,因为函数输出不仅取决于数据点本身的值,还取决于值与其他数据点的关系。例如,如果不与其他维度值进行比较,就无法计算排名。

这些函数只可用于图表表达式中。它们不能在加载脚本中使用。

图表中需要一个维度,因为它定义了比较所需的其他数据点。因此,关系函数在无量纲图表(例如,KPI 对象)中没有意义。

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]])
```

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 文件 (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 [, ...]])
```

时间序列分解函数

STL_Trend

STL_Trend 是一个时间序列分解函数。与 **STL_Seasonal** 和 **STL_Residual** 一起，此函数用于将时间序列分解为季节、趋势和残差分量。在 STL 算法的背景下，时间序列分解用于在给定输入度量和其他参数的情况下识别重复出现的季节模式和总体趋势。**STL_Trend** 函数将从时间序列数据中识别不受季节模式或周期影响的总体趋势。

```
STL_Trend - 图表函数(target_measure, period_int [,seasonal_smoother [,trend_
smoother]])
```

STL_Seasonal

STL_Seasonal 是一个时间序列分解函数。与 **STL_Trend** 和 **STL_Residual** 一起，此函数用于将时间序列分解为季节、趋势和残差分量。在 STL 算法的背景下，时间序列分解用于在给定输入度量和其他参数的情况下识别重复出现的季节模式和总体趋势。**STL_Seasonal** 函数可以识别时间序列中的季节模式，将其与数据显示的总体趋势相分离。

```
STL_Seasonal - 图表函数(target_measure, period_int [,seasonal_smoother
[,trend_smoother]])
```

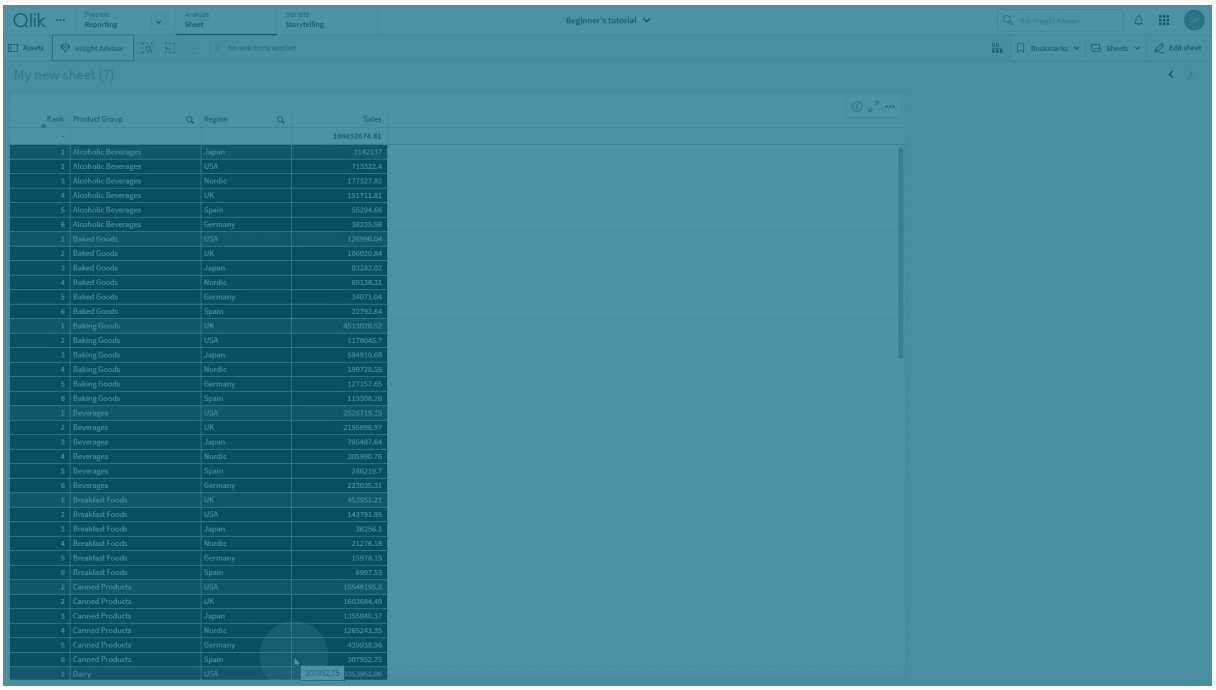
STL_Residual

STL_Residual 是一个时间序列分解函数。与 **STL_Seasonal** 和 **STL_Trend**一起，此函数用于将时间序列分解为季节、趋势和残差分量。在 STL 算法的背景下，时间序列分解用于在给定输入度量和其他参数的情况下识别重复出现的季节模式和总体趋势。在执行这项操作时，输入度量中部分变化既不符合季节成分，也不符合趋势成分，将被定义为残差成分。**STL_Residual** 图表函数捕获计算的这一部分。

```
STL_Residual - 图表函数(target_measure, period_int [,seasonal_smoother
[,trend_smoother]])
```

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

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	描述
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	描述
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:

从维度 Product 和 Sales 创建两个可视化,从 Product 和 UnitSales 创建另一个可视化。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

Customer	Sales	Rank(Sales)
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 334)

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 1330)* 和 *KMeansND - 图表函数 (page 1345)*。**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 1359)* 和 *KMeansCentroidND - 图表函数 (page 1360)*。

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 1328)

标题：DistributorData

记录的总数：118

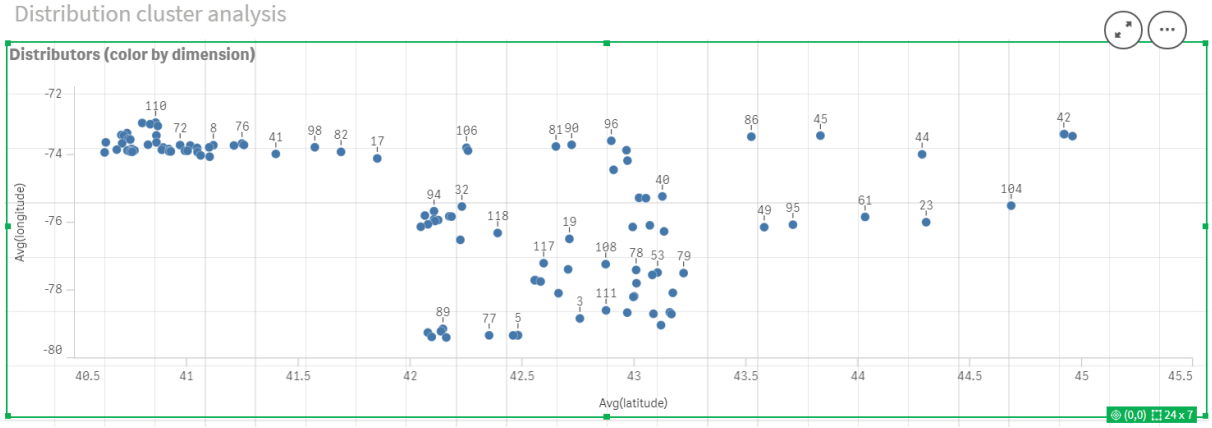
应用 KMeans2D 函数

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。

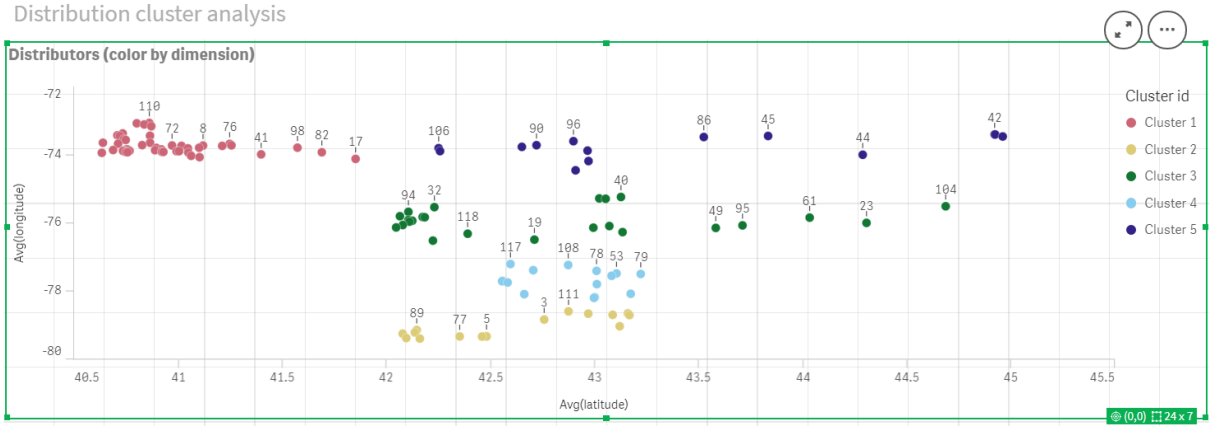
注意 Qlik Sense K 均值函数支持使用名为深度差 (DeD) 的方法支持自动聚合。如果用户为集群数设置 0，则会为该数据集确定最优集群数。但是，在本例中，为 **num_clusters** 参数创建了一个变量（有关语法，请参阅 *KMeans2D - 图表函数 (page 1330)*）。因此，所需的集群数 (k=5) 由一个变量指定。

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

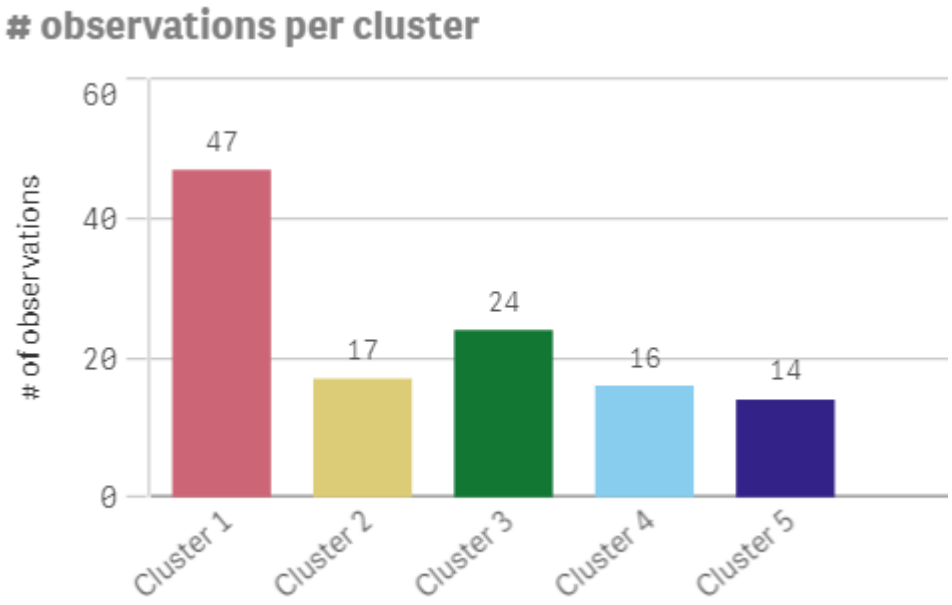
添加**条形图**:*# observations per cluster*

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

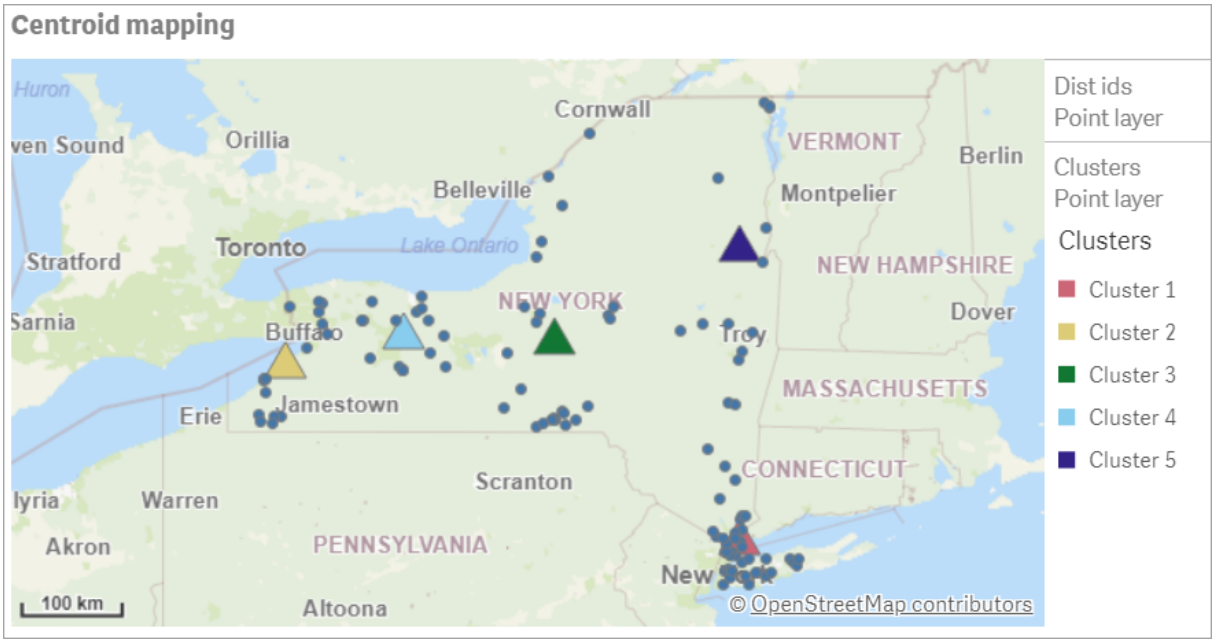
下一步是映射形心。它由应用程序开发人员决定,如果他们喜欢把可视化放置在单独的工作表上,就可以如此。

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**。

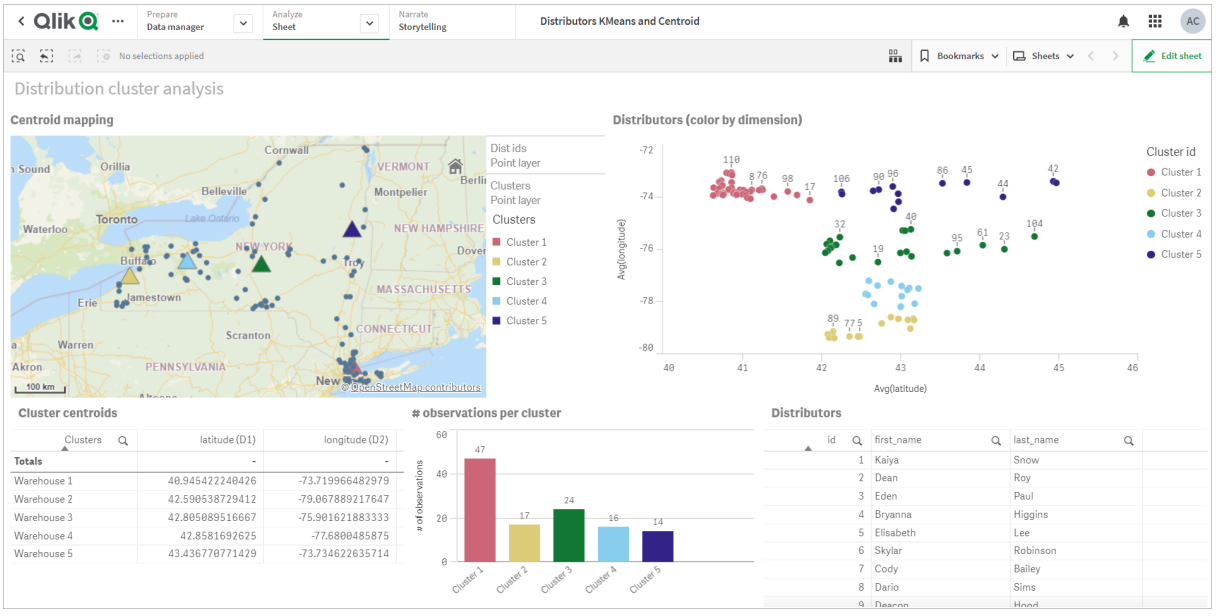
地图按群集映射的形心



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。

参数	说明
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。

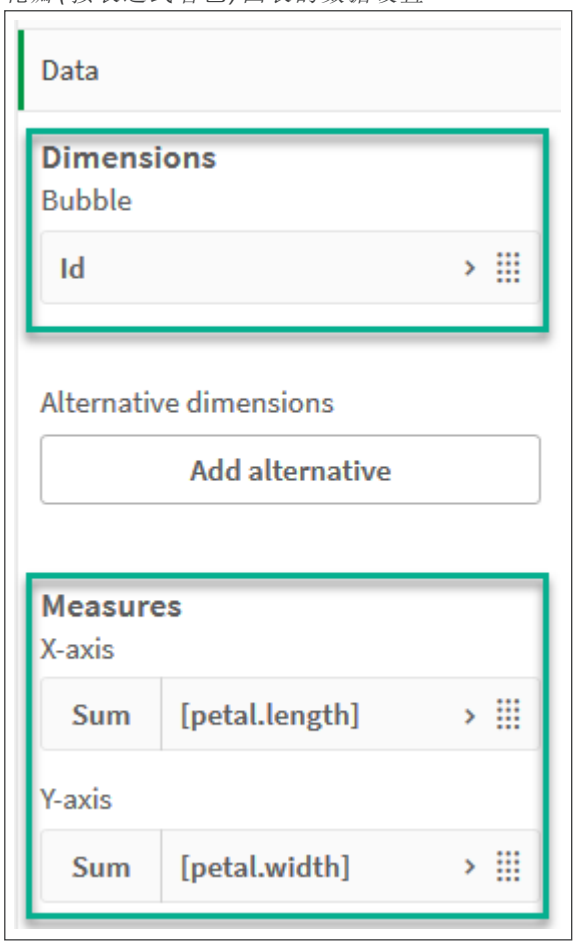
我们还为 *num_clusters* 参数创建一个变量，然后使用变量输入框来更改集群的数量。

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：

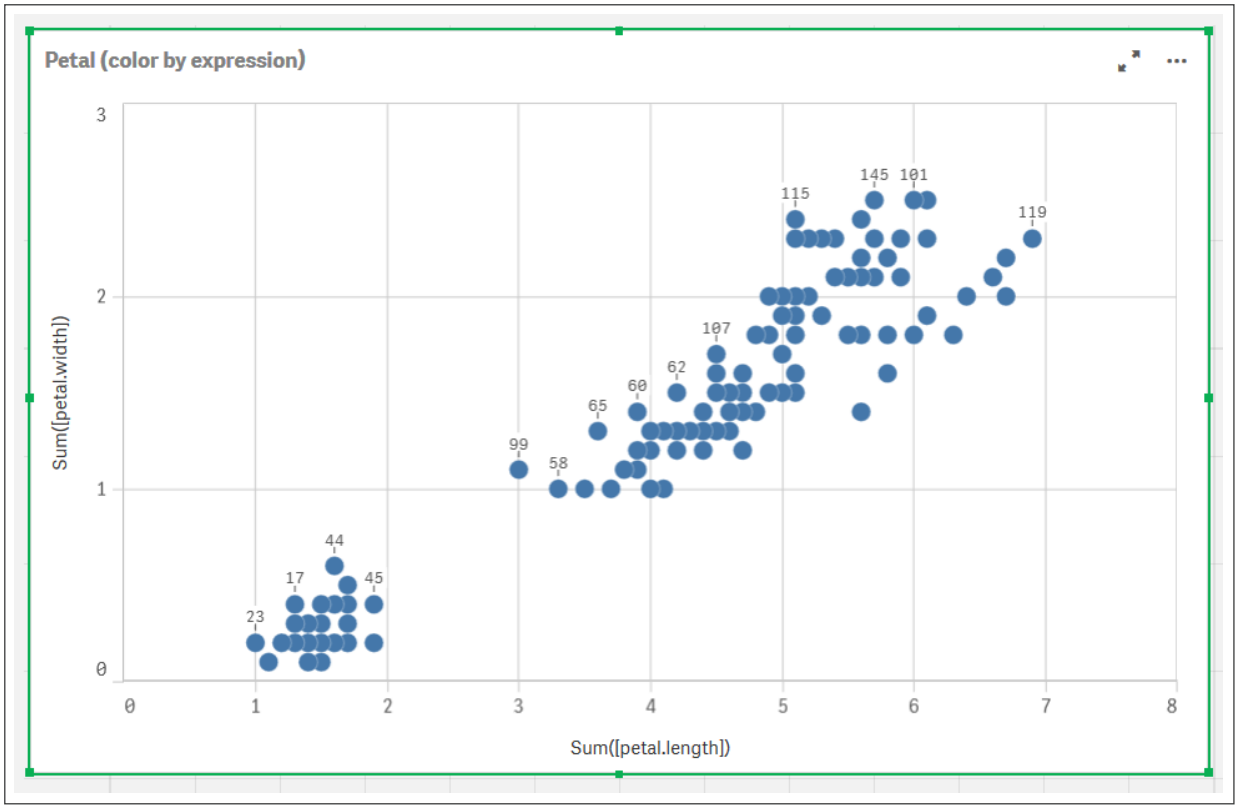
1. 将**散点图**图表拖动至新的工作表。将图表命名为*花瓣(按表达式着色)*。
2. 创建变量以指定集群数。对于变量 **Name**，输入 *KmeansPetalClusters*。对于变量 **Definition**，输入 *=2*。
3. 配置图表的**数据**：
    i. 在**维度**下，选择**气泡**的字段的 *id*。输入标签的集群 Id。
    ii. 在**度量**下，选择 **X 轴**表达式的 *Sum([petal.length])*。
    iii. 在**度量**下，选择 **Y 轴**表达式的 *Sum([petal.width])*。

花瓣(按表达式着色)图表的数据设置



数据点在图表上绘制。

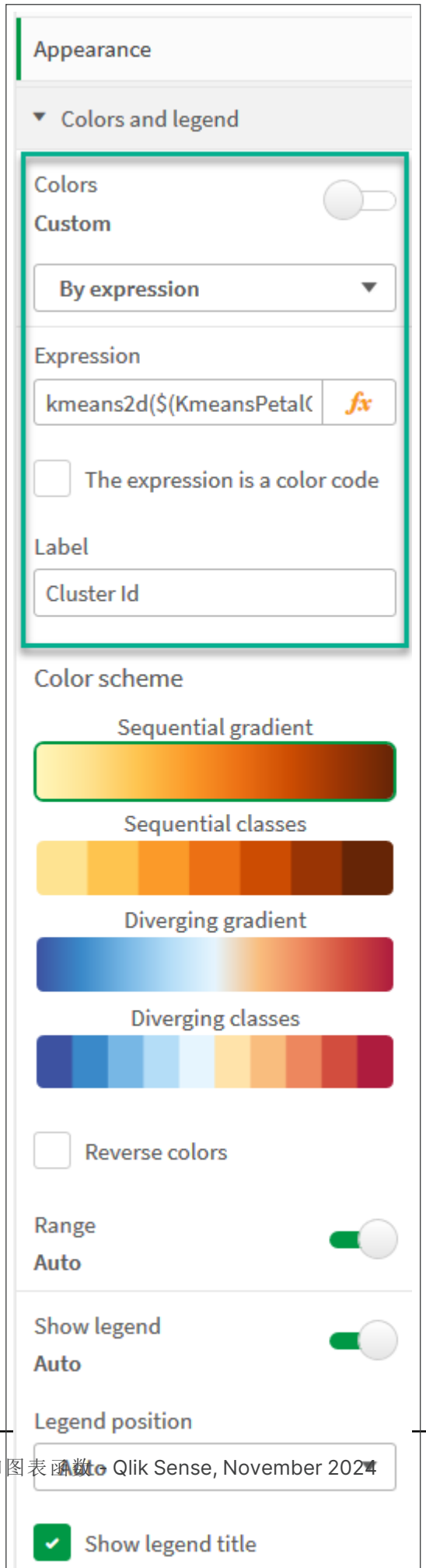
花瓣(按表达式着色)图表上的数据点



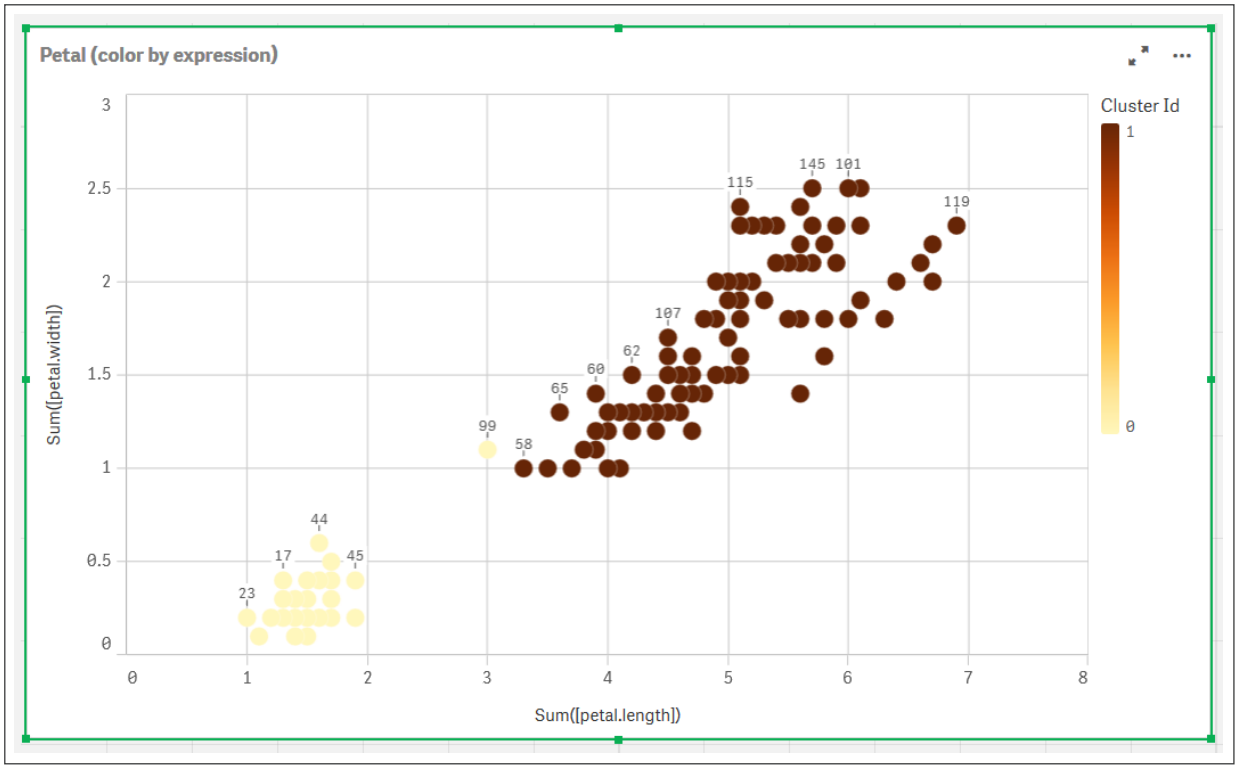
4. 配置图表的**外观**:
   i. 在**颜色和图例**下,为**颜色**选择**自定义**。
   ii. 选择侧向以**按表达式**对图表着色。
   iii. 为**表达式**输入以下内容: *kmeans2d($(KmeansPetalClusters), Sum([petal.length]), Sum([petal.width]))*
   注意 *KmeansPetalClusters* 是我们设置为 *2* 的变量。
   或者输入以下内容:*kmeans2d(2, Sum([petal.length]), Sum([petal.width]))*
   iv. 取消选择**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

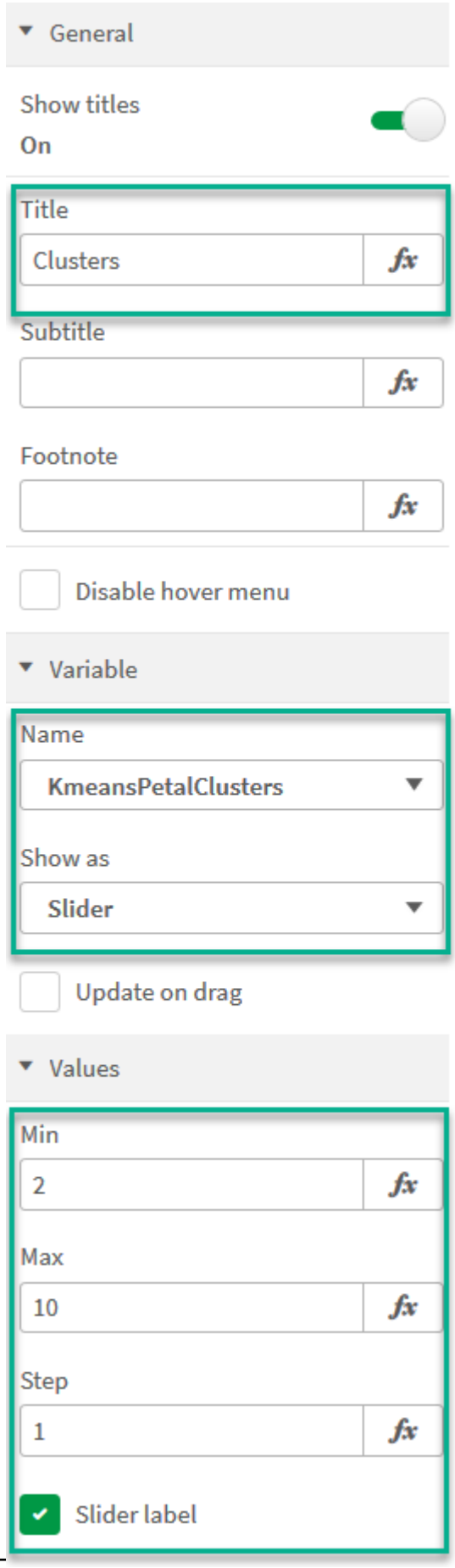
图表上的两个集群按 KMeans 表达式着色。
花瓣(按表达式着色)图表中按表达式着色群集。



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**。

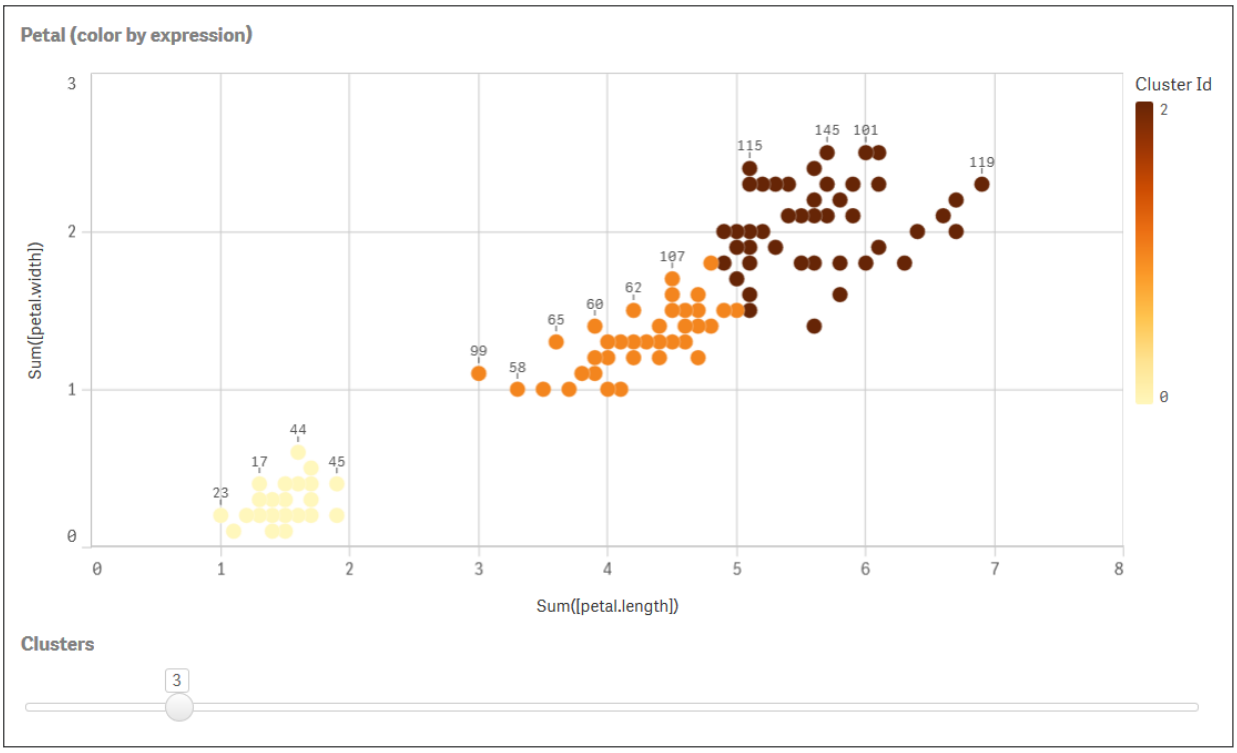
viii. 选择**值**，并根据需要配置设置。

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。

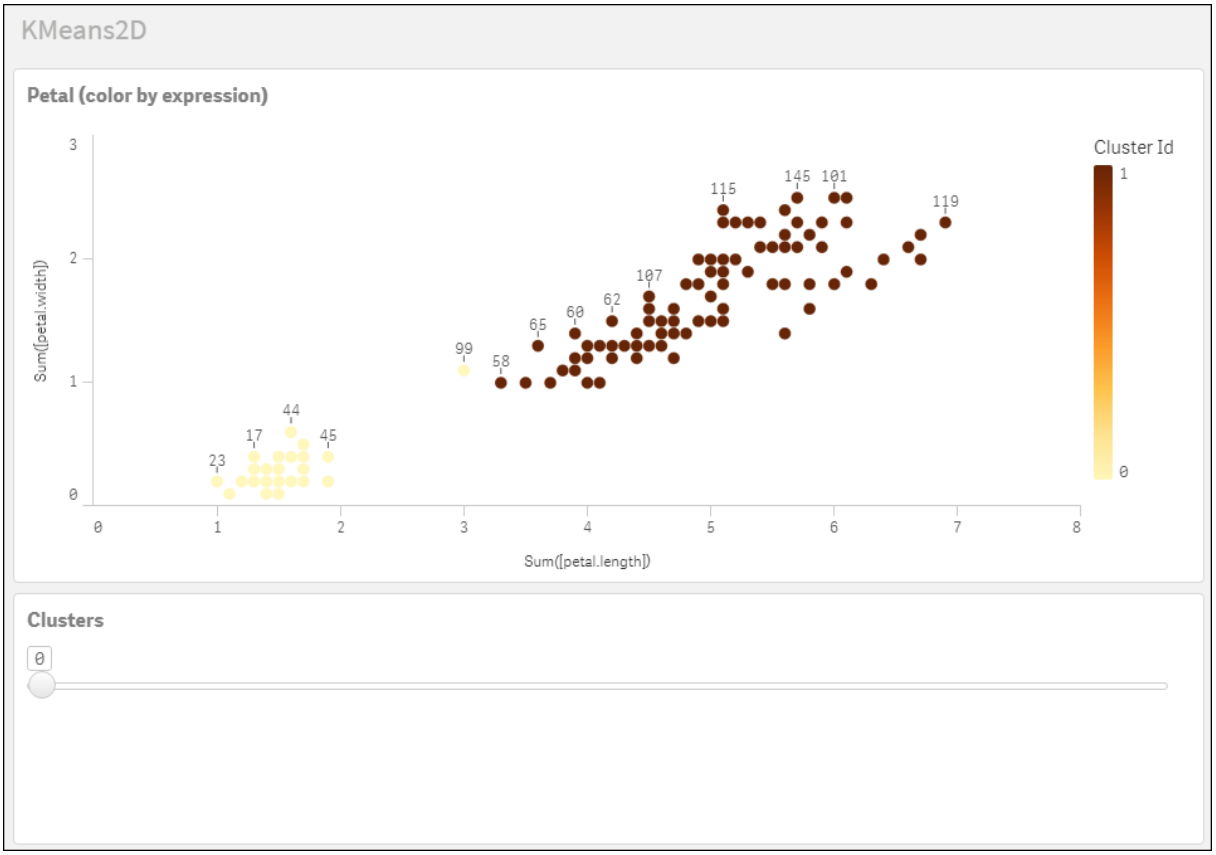
花瓣(按表达式着色)图表中按表达式着色群集。



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。
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。

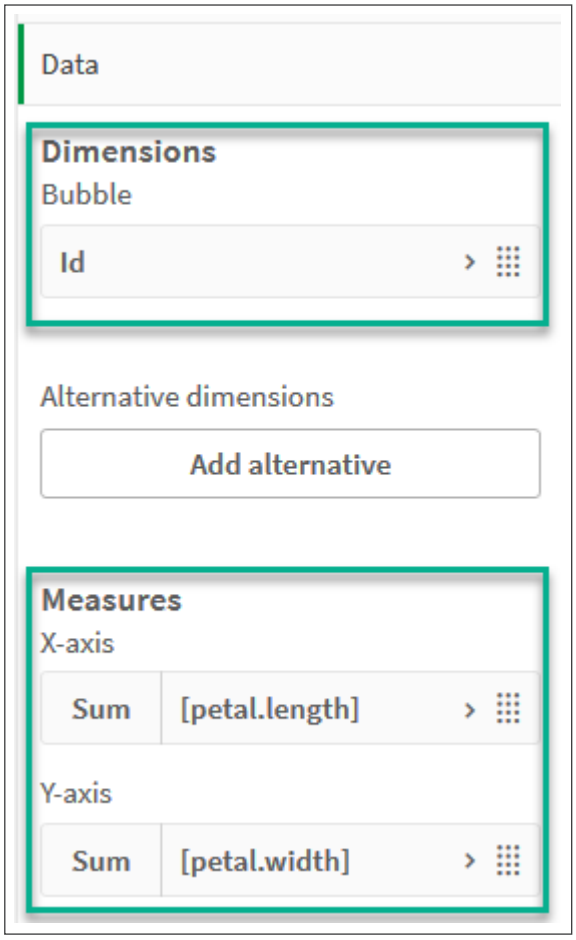
我们还为 *num_iter* 参数创建一个变量，然后使用第二变量输入框来更改迭代的数量。

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：

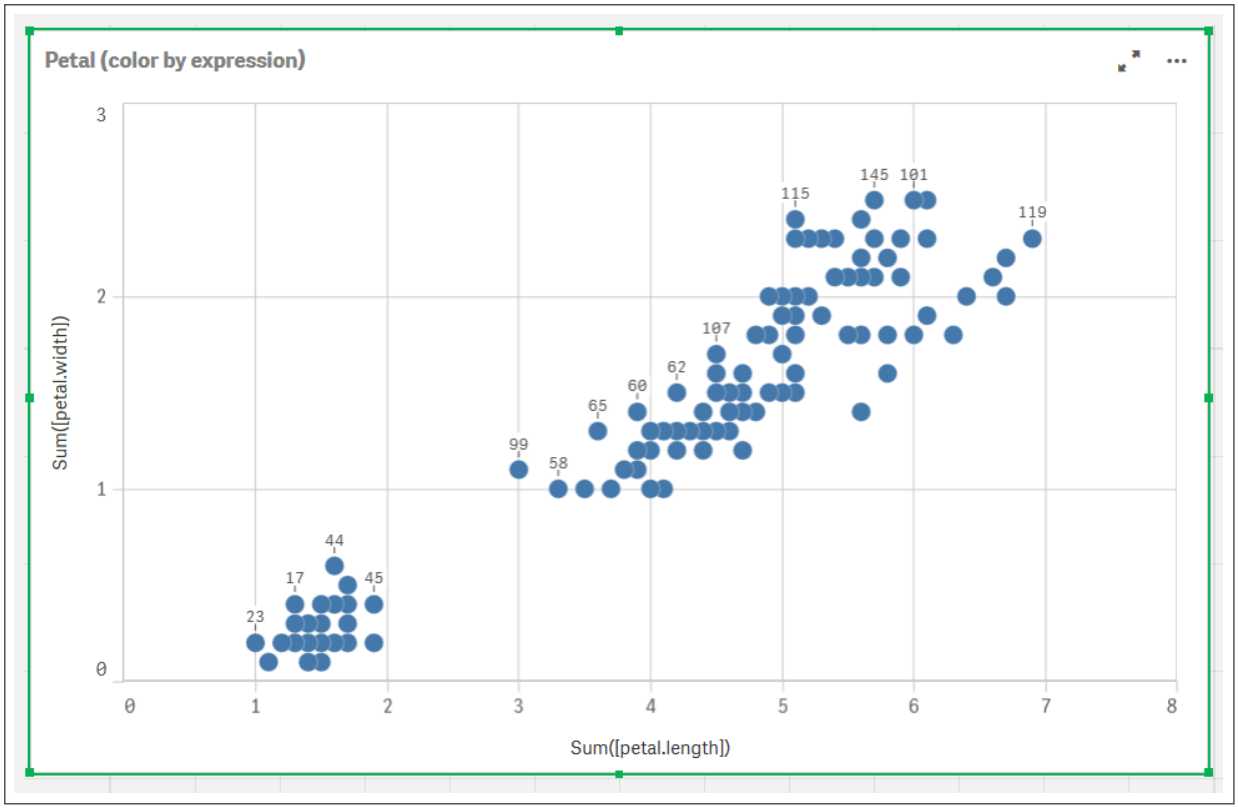
1. 将**散点图**图表拖动至新的工作表。将图表命名为*花瓣(按表达式着色)*。
2. 创建变量以指定集群数。对于变量 **Name**，输入 *KmeansPetalClusters*。对于变量 **Definition**，输入 *=2*。
3. 创建变量以指定迭代数。对于变量 **Name**，输入 *KmeansNumberIterations*。对于变量 **Definition**，输入 *=1*。
4. 配置图表的**数据**：
    i. 在**维度**下，选择**气泡**的字段的 *id*。输入标签的集群 Id。
    ii. 在**度量**下，选择 **X 轴**表达式的 *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。

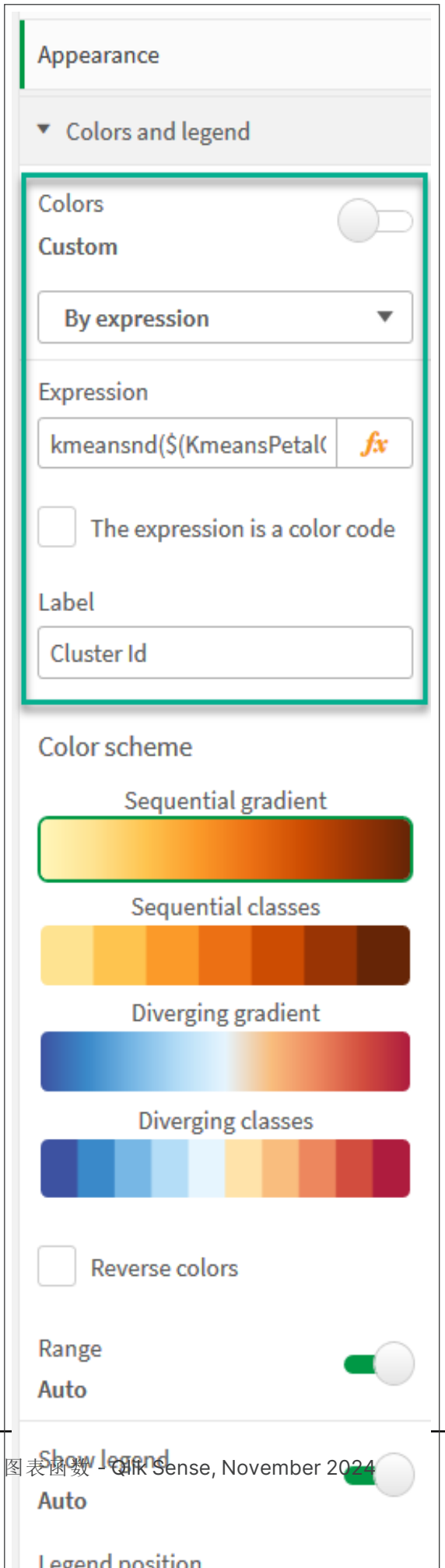
花瓣(按表达式着色)图表上的数据点



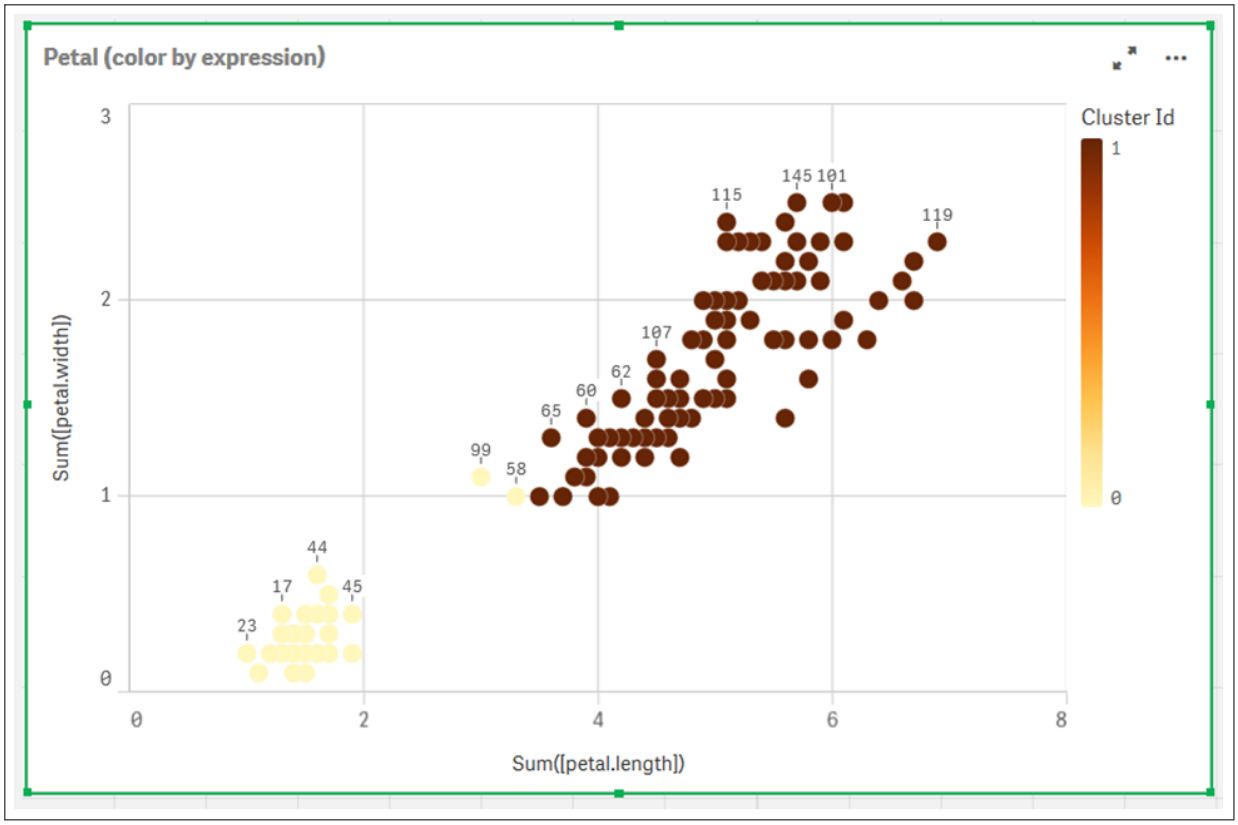
5. 配置图表的**外观**:
   i. 在**颜色和图例**下,为**颜色**选择**自定义**。
   ii. 选择侧向以**按表达式**对图表着色。
   iii. Enter the following for **Expression**: *kmeansnd ($(KmeansPetalClusters),$(KmeansNumberIterations), Sum([petal.length]), Sum ([petal.width]),Sum([sepal.length]), Sum([sepal.width]))*
   注意 *KmeansPetalClusters* 是我们设置为 *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

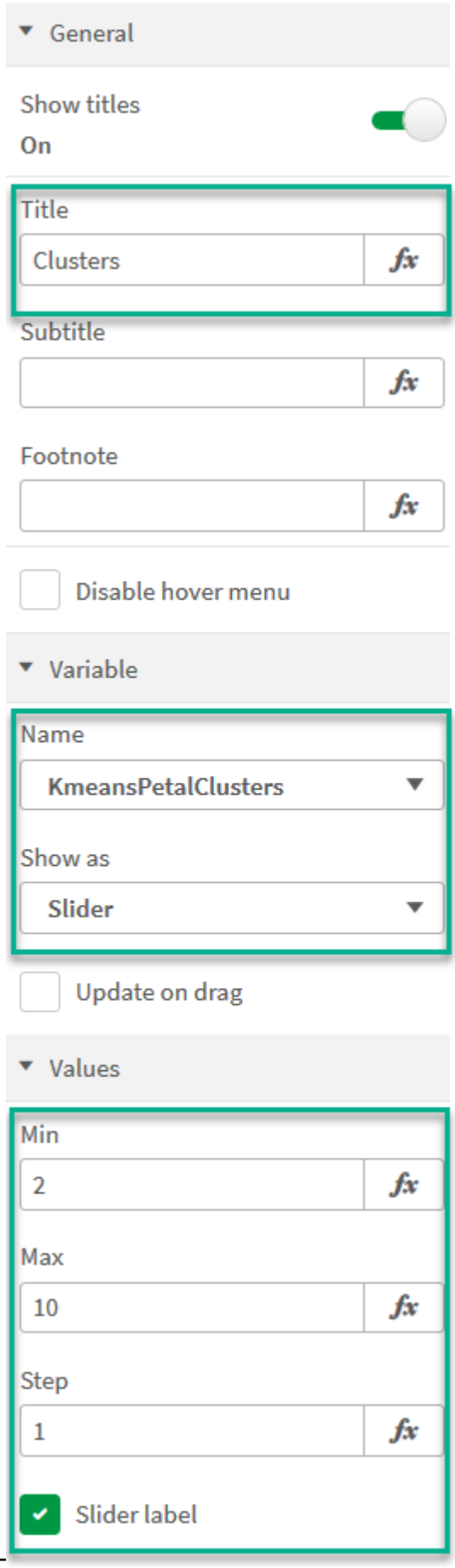
图表上的两个集群按 KMeans 表达式着色。
花瓣(按表达式着色)图表中按表达式着色群集。



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。

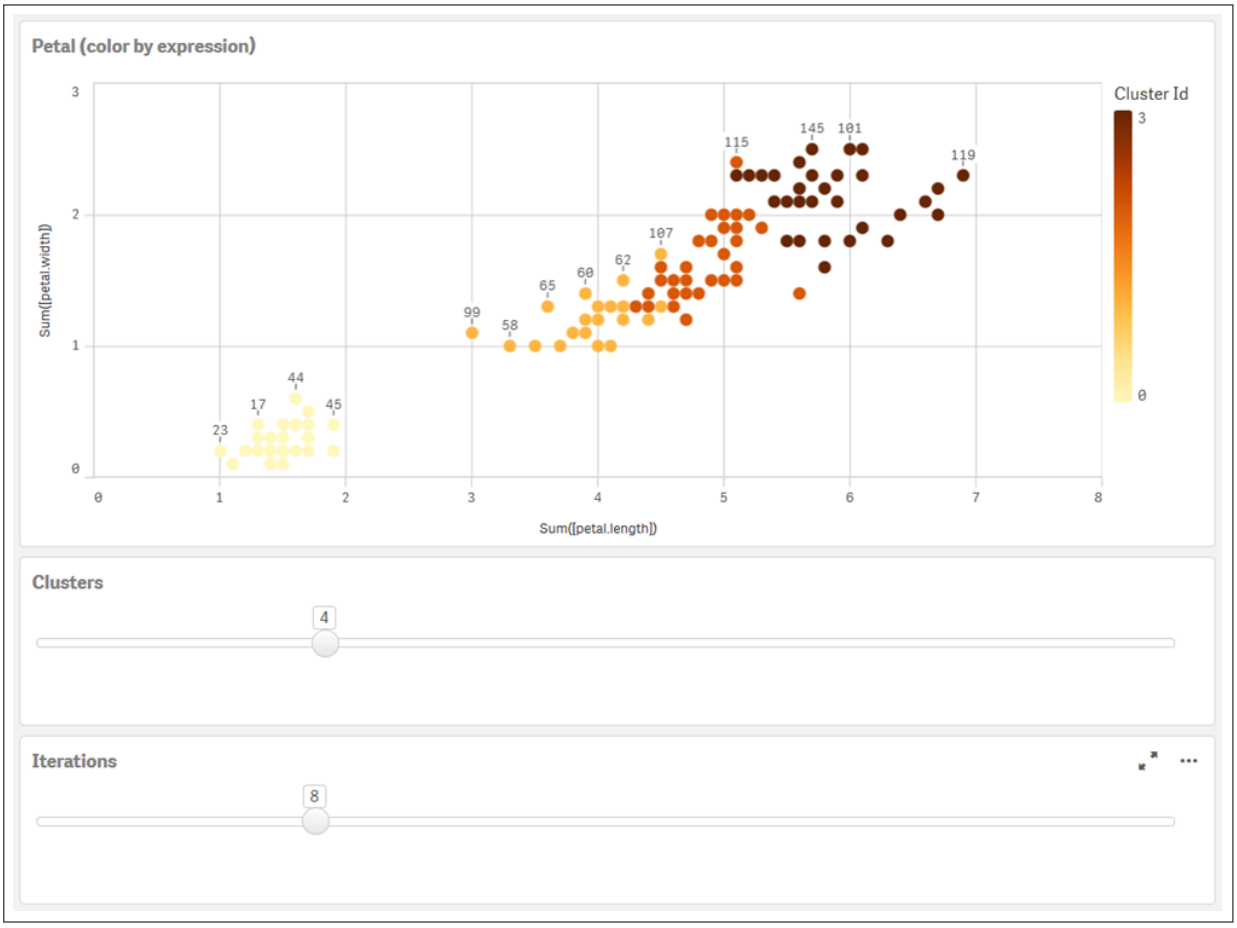
Clusters 变量输入框的外观。


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。

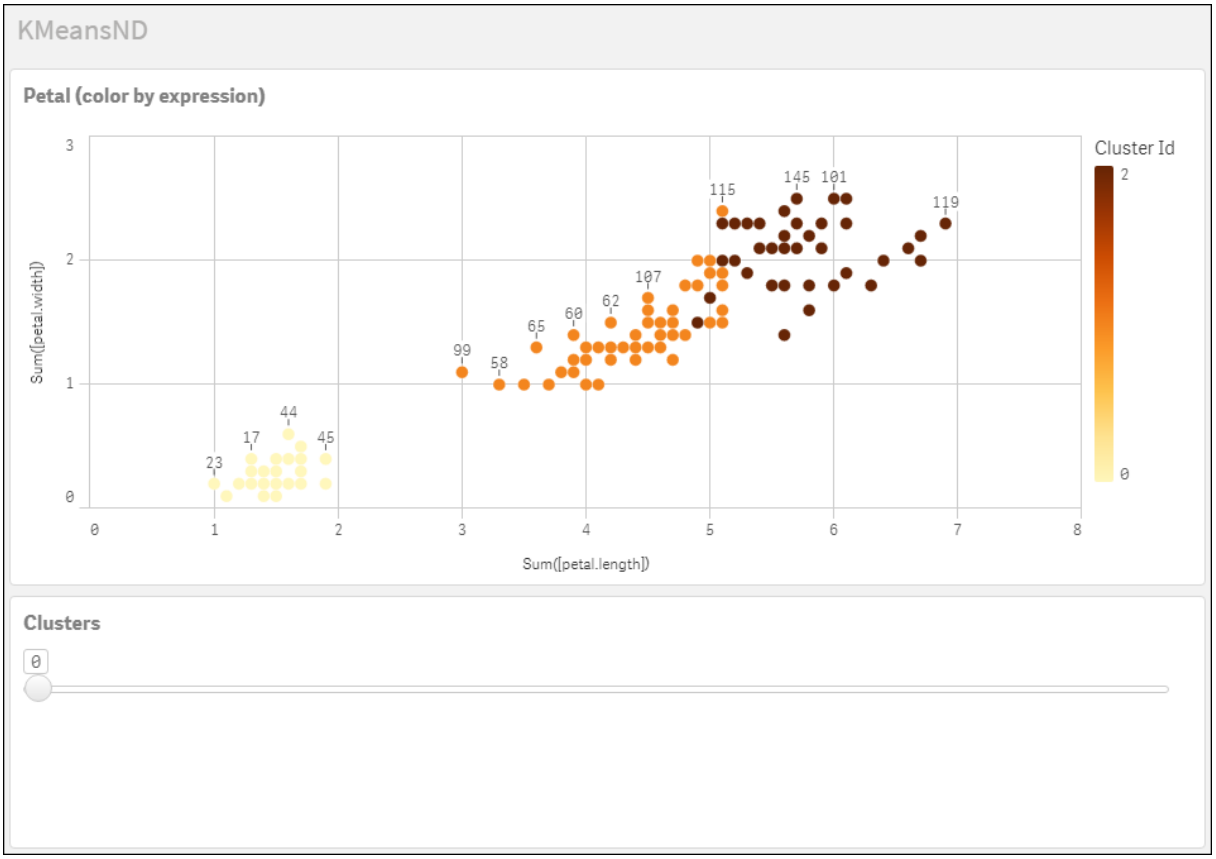
    花瓣（按表达式着色）图表中按表达式着色群集。



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。

参数	说明
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。

STL_Trend - 图表函数

STL_Trend 是一个时间序列分解函数。与 **STL_Seasonal** 和 **STL_Residual** 一起，此函数用于将时间序列分解为季节、趋势和残差分量。在 STL 算法的背景下，时间序列分解用于在给定输入度量和其他参数的情况下识别重复出现的季节模式和总体趋势。**STL_Trend** 函数将从时间序列数据中识别不受季节模式或周期影响的总体趋势。

三个 STL 函数通过简单求和与输入度量相关：

STL_Trend + **STL_Seasonal** + **STL_Residual** = 输入度量

STL(使用损失的季节和趋势分解)采用数据平滑技术，并通过其输入参数，允许用户调整其执行的计算周期。这种周期性决定了在分析中如何分割输入度量(度量)的时间维度。

STL_Trend 至少接受一个输入度量 (`target_measure`) 和其 `period_int` 的一个整数值，并返回一个浮点值。输入度量将以聚合的形式出现，并随时间维度的变化而变化。(可选) 可以包括 `seasonal_smoother` 和 `trend_smoother` 的值，以调整平滑算法。

您可以使用此函数，方法是将其直接输入到图表的表达式编辑器中。

语法：

```
STL_Trend(target_measure, period_int [,seasonal_smoother [,trend_smoother]])
```

返回数据类型： 双

参数

参数	说明
target_measure	分解为季节性和趋势分量的度量。这应该是一个度量，如 Sum(Sales) 或 Sum (Passengers)，随时间维度而变化。 这不得是一个常数值。
period_int	数据集的周期性。此参数是一个整数值，表示构成信号的一个周期或季节周期的离散步数。 例如，如果时间序列在一年的每个季度被分割为一个部分，则必须将 **period_int** 设置为值 4，以将周期定义为“年”。
seasonal_smoother	季节平滑器的长度。这必须是一个奇数。季节平滑器使用季节变化中特定阶段的数据，跨越多个时段。每个期间都使用时间维度的一个离散步骤。季节平滑器指示用于平滑的期间数。 例如，如果时间维度按月份分段，期间为年 (12)，则将计算季节成分，以便根据当年和相邻年份的同一月份数据计算每年的每个特定月份。**seasonal_smoother** 值是用于平滑的年数。
trend_smoother	趋势平滑器的长度。这必须是一个奇数。趋势平滑器使用与 **period_int** 参数相同的时间刻度，其值是用于平滑的颗粒数。 例如，如果一个时间序列按月份分段，趋势平滑器将是用于平滑的月份数。

STL_Trend 图表函数通常与以下函数结合使用：

相关函数

函数	交互
STL_Seasonal - 图表函数 (page 1363)	这是用于计算时间序列季节分量的函数。
STL_Residual - 图表函数 (page 1364)	当将一个输入度量分解为季节和趋势分量时，该度量的部分变化将不适合这两个主要分量中的任何一个。**STL_Residual** 函数计算这部分分解。

如果需要教程，其中有展示如何使用此函数的完整示例，请参见*教程 - Qlik Sense 中的时间序列分解 (page 1366)*。

STL_Seasonal - 图表函数

STL_Seasonal 是一个时间序列分解函数。与 **STL_Trend** 和 **STL_Residual** 一起，此函数用于将时间序列分解为季节、趋势和残差分量。在 STL 算法的背景下，时间序列分解用于在给定输入度量和其他参数的情况下识别重复出现的季节模式和总体趋势。**STL_Seasonal** 函数可以识别时间序列中的季节模式，将其与数据显示的总体趋势相分离。

三个 STL 函数通过简单求和与输入度量相关：

STL_Trend + **STL_Seasonal** + **STL_Residual** = 输入度量

STL(使用损失的季节和趋势分解)采用数据平滑技术，并通过其输入参数，允许用户调整其执行的计算周期。这种周期性决定了在分析中如何分割输入度量(度量)的时间维度。

STL_Seasonal 至少接受一个输入度量 (`target_measure`) 和其 `period_int` 的一个整数值，并返回一个浮点值。输入度量将以聚合的形式出现，并随时间维度的变化而变化。(可选)可以包括 `seasonal_smoother` 和 `trend_smoother` 的值，以调整平滑算法。

您可以使用此函数，方法是将其直接输入到图表的表达式编辑器中。

语法：

```
STL_Seasonal(target_measure, period_int [,seasonal_smoother [,trend_smoother]])
```

返回数据类型：双

参数

参数	说明
target_measure	分解为季节性和趋势分量的度量。这应该是一个度量，如 Sum(Sales) 或 Sum (Passengers)，随时间维度而变化。 这不得是一个常数值。
period_int	数据集的周期性。此参数是一个整数值，表示构成信号的一个周期或季节周期的离散步数。 例如，如果时间序列在一年的每个季度被分割为一个部分，则必须将 **period_int** 设置为值 4，以将周期定义为“年”。
seasonal_smoother	季节平滑器的长度。这必须是一个奇数。季节平滑器使用季节变化中特定阶段的数据，跨越多个时段。每个期间都使用时间维度的一个离散步骤。季节平滑器指示用于平滑的期间数。 例如，如果时间维度按月份分段，期间为年 (12)，则将计算季节成分，以便根据当年和相邻年份的同一月份数据计算每年的每个特定月份。**seasonal_smoother** 值是用于平滑的年数。

参数	说明
trend_ smoother	趋势平滑器的长度。这必须是一个奇数。趋势平滑器使用与 **period_int** 参数相同的时间刻度，其值是用于平滑的颗粒数。 例如，如果一个时间序列按月份分段，趋势平滑器将是用于平滑的月份数。

STL_Seasonal 图表函数通常与以下函数结合使用：

相关函数

函数	交互
STL_Trend - 图表函数 (page 1361)	这是用于计算时间序列趋势分量的函数。
STL_Residual - 图表函数 (page 1364)	当将一个输入度量分解为季节和趋势分量时，该度量的部分变化将不适合这两个主要分量中的任何一个。**STL_ Residual** 函数计算这部分分解。

如果需要教程，其中有展示如何使用此函数的完整示例，请参见*教程 - Qlik Sense 中的时间序列分解 (page 1366)*。

STL_Residual - 图表函数

STL_Residual 是一个时间序列分解函数。与 **STL_Seasonal** 和 **STL_Trend**一起，此函数用于将时间序列分解为季节、趋势和残差分量。在 STL 算法的背景下，时间序列分解用于在给定输入度量和其他参数的情况下识别重复出现的季节模式和总体趋势。在执行这项操作时，输入度量中部分变化既不符合季节成分，也不符合趋势成分，将被定义为残差成分。**STL_Residual** 图表函数捕获计算的这一部分。

三个 STL 函数通过简单求和与输入度量相关：

STL_Trend + **STL_Seasonal** + **STL_Residual** = 输入度量

STL(使用损失的季节和趋势分解)采用数据平滑技术，并通过其输入参数，允许用户调整其执行的计算周期。这种周期性决定了在分析中如何分割输入度量(度量)的时间维度。

由于时间序列分解主要寻找数据的季节性和一般变化，因此残差中的信息被认为是三个组成部分中最不重要的。然而，倾斜或周期性残差分量有助于识别计算中的问题，例如不正确的周期设置。

至少，**STL_Residual** 采用一个输入度量 (target_measure) 和针对其 period_int 的一个整数值，并返回一个浮点值。输入度量将以聚合的形式出现，并随时间维度的变化而变化。(可选) 可以包括 seasonal_smoother 和 trend_smoother 的值，以调整平滑算法。

您可以使用此函数，方法是将其直接输入到图表的表达式编辑器中。

语法：

```
STL_Residual(target_measure, period_int [,seasonal_smoother [,trend_
smoother]])
```

返回数据类型： 双

参数

参数	说明
target_measure	分解为季节性和趋势分量的度量。这应该是一个度量，如 Sum(Sales) 或 Sum (Passengers)，随时间维度而变化。 这不得是一个常数值。
period_int	数据集的周期性。此参数是一个整数值，表示构成信号的一个周期或季节周期的离散步数。 例如，如果时间序列在一年的每个季度被分割为一个部分，则必须将 **period_int** 设置为值 4，以将周期定义为“年”。
seasonal_smoother	季节平滑器的长度。这必须是一个奇数。季节平滑器使用季节变化中特定阶段的数据，跨越多个时段。每个期间都使用时间维度的一个离散步骤。季节平滑器指示用于平滑的期间数。 例如，如果时间维度按月份分段，期间为年 (12)，则将计算季节成分，以便根据当年和相邻年份的同一月份数据计算每年的每个特定月份。**seasonal_smoother** 值是用于平滑的年数。
trend_smoother	趋势平滑器的长度。这必须是一个奇数。趋势平滑器使用与 **period_int** 参数相同的时间刻度，其值是用于平滑的颗粒数。 例如，如果一个时间序列按月份分段，趋势平滑器将是用于平滑的月份数。

STL_Residual 图表函数通常与以下函数结合使用：

相关函数

函数	交互
STL_Seasonal - 图表函数 (page 1363)	这是用于计算时间序列季节分量的函数。
STL_Trend - 图表函数 (page 1361)	这是用于计算时间序列趋势分量的函数。

如果需要教程，其中有展示如何使用此函数的完整示例，请参见*教程 - Qlik Sense 中的时间序列分解 (page 1366)*。

教程 - Qlik Sense 中的时间序列分解

本教程演示使用 STL 算法并使用三个图表函数分解时间序列。

本教程使用航空公司每月乘客数量的时间序列数据来演示 STL 算法的功能。**STL_Trend**、**STL_Seasonal** 和 **STL_Residual** 图表函数将用于创建可视化效果。有关Qlik Sense中时间序列分解的详细信息，请参阅*时间序列分解函数 (page 1315)*。

创建应用程序

首先创建一个新应用程序并将数据集导入其中。

下载此数据集：

教程 - 时间序列分解

该文件包含有关航空公司每月乘客数量的数据。

执行以下操作：

1. 在应用中心中，单击**创建新应用程序**。
2. 打开应用程序并将 *Tutorial - Time series decomposition.csv* 文件拖动到其中。

准备并加载数据

为了 Qlik Sense 正确解释 YearMonth 字段，您可能需要使用数据管理器将该字段识别为日期字段，而不是具有字符串值的字段。通常，此步骤是自动处理的，但在这种情况下，日期以稍微不常见的 *YYYY-MM* 格式显示。

1. 在数据管理器中，选择表并单击 。
2. 选中 *YearMonth* 字段后，单击 并将**字段类型**设置为**日期**。
3. 在**输入格式**下，输入 *YYYY-MM*。
4. 在**显示格式**下，输入 *YYYY-MM*，然后单击**确定**。
   该字段现在应该显示日历图标。
5. 单击**加载数据**。

现在，您可以开始使用 STL 函数直观地表示数据了。

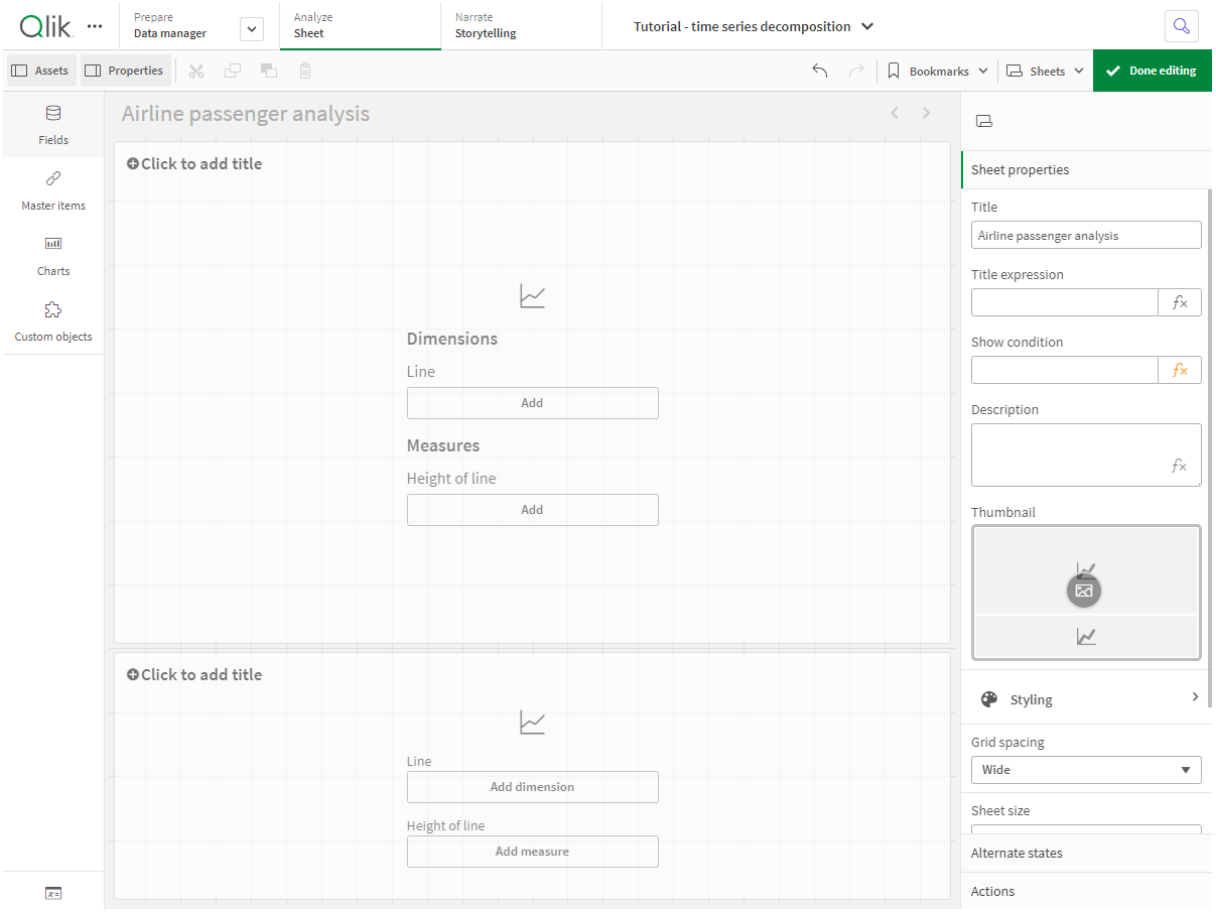
创建可视化

接下来，您将创建两个折线图来演示 **STL_Trend**、**STL_Seasonal** 和 **STL_Residual** 图表函数的功能。

打开一个新工作表并给它提供一个标题。

在工作表中添加两个折线图。调整图表的大小并重新定位，以匹配下图。

Qlik Sense 空白应用程序工作表的网格轮廓



第一个折线图:趋势和季节分量

执行以下操作:

1. 将标题 *Seasonal and Trend* 添加到第一个折线图中。
2. 添加 *YearMonth* 作为维度,并将其标记为*日期*。
3. 添加以下度量并将其标记为*每月乘客*:
   =Sum(Passengers)
4. 在**数据**下,展开*每月乘客*度量,然后单击**添加趋势线**。
5. 将**类型**设置为**线性**。
   您将将此趋势线与趋势分量的平滑输出进行比较。
6. 添加以下度量以绘制趋势分量并将其标记为*趋势*:
   =STL_Trend(SUM(Passengers), 12)
7. 接下来,添加以下度以绘制季节性分量并将其标记为*季节性*:
   =STL_Seasonal(SUM(Passengers), 12)
8. 在**外观 > 演示**下,将**滚动条**设置为**无**。
9. 保留默认颜色,或根据您的喜好进行更改。

第二个直线图:残余分量

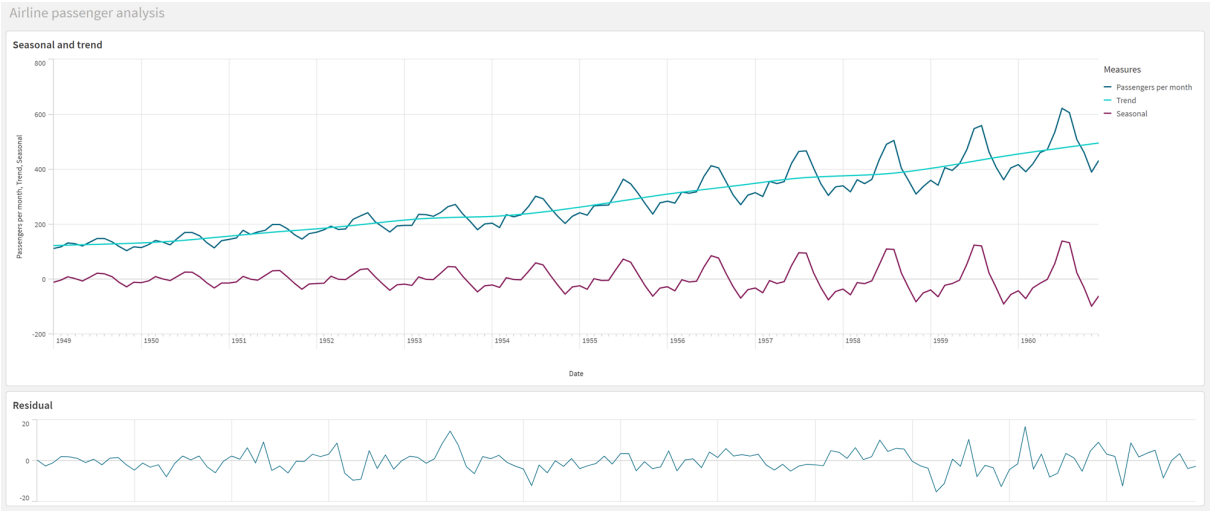
接下来,配置第二个折线图。该可视化将显示时间序列的剩余分量。

执行以下操作:

1. 将折线图拖到工作表上。添加标题 *Residual*。
2. 添加*日期*作为维度。
3. 添加以下度量并将其标记为*残余*:
   =STL_Residual(SUM(Passengers), 12)
4. 在**外观 > 演示**下,将**滚动条**设置为**无**。

您的工作绩效在应该看起来像下面的工作表。

用于航空公司乘客分析的 Qlik Sense 工作表



解读和解释数据

使用 STL 图表函数,我们可以从时间序列数据中获得许多见解。

趋势分量

趋势分量中的统计信息被去季节化。这使得人们更容易看到随时间推移的一般性、非重复性波动。与*每月乘客*的直线、线性趋势线相比,STL 趋势分量确实捕捉到了变化的趋势。它显示了一些明显的偏差,同时仍然以可读的方式显示信息。STL 算法中的平滑行为有助于捕捉这一点。

STL 趋势图中可见的航空公司乘客数量下降可以能部分是因为受到 20 世纪 50 年代发生的经济衰退的经济影响。

季节性分量

去趋势季节性分量隔离了整个时间序列中的反复波动,并从该部分分析中删除了一般趋势信息。我们从一个由年-月聚合组成的数据集开始。有了这些数据,我们将数据分为一个月的粒度。通过定义周期值 12,我们将图表设置为在一年(十二个月)周期内模拟季节模式。

在数据中，夏季的航空乘客会出现反复的季节性激增，冬季则会出现下降。这与夏季通常是度假和旅行的热门时间的想法一致。我们还看到，在时间序列的过程中，这些季节性周期的幅度急剧增加。

残余分量

残差分量的图表显示了趋势和季节分解中未捕获的所有信息。残差分量包括统计噪声，但它也可能指示 STL 趋势和季节函数参数的错误设置。通常，如果信号的残余分量中存在周期性振荡，或者显示的信息明显不是随机的，则通常是时间序列中存在当前未在季节或趋势分量中捕获的信息的信号。在这种情况下，您需要重新检查每个函数参数的定义，并可能更改周期性。

更平滑的值

由于我们没有为趋势和季节平滑器指定任何值，因此函数将使用这些参数的默认值。在 Qlik Sense 中，STL 算法中的默认平滑值会产生有效的结果。因此，在大多数情况下，这些参数可以不包含在表达式中。

> *在三个 STL 函数中的任何一个函数中，将季节性或趋势平滑参数设置为 0 都会使算法使用默认值，而不是值 0。*

趋势平滑值使用图表中指定的维度。由于 *YearMonth* 字段按月份显示数据，因此趋势更平滑的值将是月份数。季节性平滑器将反映定义的周期性。在这种情况下，由于我们将一个时期定义为持续十二个月(一年)，因此季节性更平滑的值是年数。这听起来可能令人困惑，但这确实意味着要找到季节性，我们需要跨越多个季节。这个数字在季节上更平滑。

其他有用信息

考虑到季节性周期的振幅随时间增加，更先进的分析方法可以利用对数函数来创建乘法分解。实际上，可以通过将季节性除以趋势分量以在 Qlik Sense 中创建相对振幅的简单度量。这样做后，我们注意到，随着时间的推移，每个周期的夏季峰值相对幅度会增大。然而，冬季低点的振幅不会随时间增加。

8.23 统计分布函数

统计分布函数返回给定输入变量的不同可能结果的发生概率。可以使用这些函数计算数据点的潜在价值。

以下描述的三组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。

- 概率函数计算分布中由提供的值给出的点上的概率。
  - 频率函数用于离散分布。
  - 密度函数用于连续函数。
- Dist 函数计算分布中由提供的值给出的点上的分布累积概率。
- Inv 函数用于在给定分布的累积概率的情况下计算反值。

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。

BetaDensity

BetaDensity() 返回 Beta 概率分布。

```
BetaDensity (value, alpha, beta)
```

BetaDist

BetaDist() 返回累计 Beta 概率分布。

```
BetaDist (value, alpha, beta)
```

BetaInv

BetaINV() 返回累计 Beta 概率分布的倒数。

```
BetaInv (prob, alpha, beta)
```

BinomDist

BinomDist() 返回累计二项式概率分布。

```
BinomDist (value, trials, trial_probability)
```

BinomFrequency

BinomFrequency() 用于返回二项式概率分布。

```
BinomFrequency (value, trials, trial_probability)
```

BinomInv

BinomInv() 返回累计二项式概率分布的倒数。

```
BinomInv (prob, trials, trial_probability)
```

ChiDensity

ChiDensity()返回 chi^2 分布的单尾概率。chi^2 密度函数与 chi^2 检测相关联。

```
ChiDensity (value, degrees_freedom)
```

ChiDist

ChiDist()返回 chi^2 分布的单尾概率。chi^2 分布与 chi^2 检测相关联。

```
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)
```

FDensity

FDensity() 返回 F 概率分布。

```
FDensity (value, degrees_freedom1, degrees_freedom2)
```

FDist

FDist() 返回累计 F 概率分布。

```
FDist (value, degrees_freedom1, degrees_freedom2)
```

FInv

FInv() 返回累计 F 概率分布的倒数。

```
FInv (prob, degrees_freedom1, degrees_freedom2)
```

GammaDensity

GammaDensity() 返回 Gamma 概率分布。

```
GammaDensity (value, k, θ)
```

GammaDist

GammaDist() 返回累计 Gamma 概率分布。

```
GammaDist (value, k, θ)
```

GammaInv

GammaInv() 返回累计 Gamma 概率分布的倒数。

```
GammaInv (prob, k, θ)
```

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误差的累积正态分布的相反值。

```
NormInv (prob, mean, standard_dev)
```

PoissonDist

PoissonDist() 返回累计泊松概率分布。

```
PoissonDist (value, mean)
```

PoissonFrequency

PoissonFrequency() 用于返回泊松概率分布。

```
PoissonFrequency (value, mean)
```

PoissonInv

PoissonInv() 返回累计泊松概率分布的倒数。

```
PoissonInv (prob, mean)
```

TDensity

TDensity() 返回学生的 t 密度函数的值，其中数值是要对其计算概率的 t的计算值。

```
TDensity (value, degrees_freedom, tails)
```

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)
```

另请参见：

统计聚合函数 (page 377)

BetaDensity

BetaDensity() 返回 Beta 概率分布。

语法：

```
BetaDensity(value, alpha, beta)
```

返回数据类型：数字

参数

参数	说明
value	您想要用于评估分布的值。该值必须介于 0 和 1 之间。
alpha	定义首个形状参数的正数。它是随机变量的指数
beta	定义第二个形状参数的正数。它表示分母自由度的数字。

BetaDist

BetaDist() 返回累计 Beta 概率分布。

语法：

```
BetaDist(value, alpha, beta)
```

返回数据类型： 数字

参数

参数	说明
value	您想要用于评估分布的值。该值必须介于 0 和 1 之间。
alpha	定义首个形状参数的正数。它是随机变量的指数
beta	定义第二个形状参数的正数。它是控制分布形状的指数。

此函数通过以下方式与 BetaInv 函数关联：
If prob = BetaDist(value, alpha, beta), then BetaInv(prob, alpha, beta) = value

BetaInv

BetaINV() 返回累计 Beta 概率分布的倒数。

语法：

```
BetaInv(prob, alpha, beta)
```

返回数据类型： 数字

参数

参数	描述
prob	与 Beta 概率分布相关联的概率。必须为一个介于 0 和 1 之间的值。
alpha	定义首个形状参数的正数。它是随机变量的指数
beta	定义第二个形状参数的正数。它是控制分布形状的指数。

此函数通过以下方式与 BetaDist 函数关联：
If prob = BetaDist(value, alpha, beta), then BetaInv(prob, alpha, beta) = value

BinomDist

BinomDist() 返回累计二项式概率分布。

语法：

```
BinomDist(value, trials, trial_probability)
```

返回数据类型： 数字

参数

参数	说明
value	您想要用于评估分布的值。该值必须是不小于零且不大于试验次数的整数。

参数	说明
trials	说明试验次数的正整数。
trial_probability	每次试验的成功概率。它始终是介于 0 和 1 之间的值。

此函数通过以下方式与 BinomInv 函数关联：
If prob = BinomDIST(value, trials, trial_probability), then BinomInv(prob, trials, trial_probability) = value

BinomFrequency

BinomFrequency() 用于返回二项式概率分布。

语法：

```
BinomFrequency(value, trials, trial_probability)
```

返回数据类型： 数字

参数

参数	说明
value	您想要用于评估分布的值。该值必须是不小于零且不大于试验次数的整数。
trials	说明试验次数的正整数
trial_probability	每次试验的成功概率。它始终是介于 0 和 1 之间的值。

BinomInv

BinomInv() 返回累计二项式概率分布的倒数。

语法：

```
BinomInv(prob, trials, trial_probability)
```

返回数据类型： 数字

参数

参数	描述
prob	与二项概率分布相关联的概率。必须为一个介于 0 和 1 之间的值。
trials	说明试验次数的正整数。
trial_probability	每次试验的成功概率。它始终是介于 0 和 1 之间的值。

此函数通过以下方式与 BinomDist 函数关联：
If prob = BinomDist(value, trials, trial_probability), then BinomInv(prob, trials, trial_probability) = value

ChiDensity

ChiDensity()返回 chi^2 分布的单尾概率。chi^2 密度函数与 chi^2 检测相关联。

语法：

```
ChiDensity(value, degrees_freedom)
```

返回数据类型： 数字

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	表示自由的分子度数的正整数。

ChiDist

ChiDist()返回 chi^2 分布的单尾概率。chi^2 分布与 chi^2 检测相关联。

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

FDensity

`FDensity()` 返回 F 概率分布。

语法：

```
FDensity(value, degrees_freedom1, degrees_freedom2)
```

返回数据类型： 数字

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_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。

FDist

`FDist()` 返回累计 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。值必须不能为负数。
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()` 返回累计 `F` 概率分布的倒数。

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

GammaDensity

GammaDensity() 返回 Gamma 概率分布。

语法：

```
GammaDensity(value, k, θ)
```

返回数据类型： 数字

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。
k	定义形状参数的正数。
θ	定义范围参数的正数。

GammaDist

GammaDist() 返回累计 Gamma 概率分布。

语法：

```
GammaDist(value, k, θ)
```

返回数据类型： 数字

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。
k	定义形状参数的正数。
θ	定义范围参数的正数。

此函数通过以下方式与 GammaINV 函数关联：
If prob = GammaDist(value, k, θ), then GammaInv(prob, k, θ) = value

GammaInv

GammaInv() 返回累计 Gamma 概率分布的倒数。

语法：

```
GammaInv(prob, k, θ)
```

返回数据类型： 数字

参数

参数	描述
prob	与伽马概率分布相关联的概率。必须为一个介于 0 和 1 之间的值。
k	定义形状参数的正数。
θ	定义范围参数的正数。

此函数通过以下方式与 GammaDist 函数关联：
`If prob = GammaDist(value, k, θ), then GammaInv(prob, k, θ) = value`

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。
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误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

PoissonDist

PoissonDist() 返回累计泊松概率分布。

语法：

```
PoissonDist(value, mean)
```

返回数据类型：数字

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。
mean	定义平均结果的正数。

此函数通过以下方式与 PoissonInv 函数关联：
If prob = PoissonDist(value, mean), then PoissonInv(prob, mean) = value

PoissonFrequency

PoissonFrequency() 用于返回泊松概率分布。

语法：

```
PoissonFrequency(value, mean)
```

返回数据类型：数字

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。
mean	定义平均结果的正数。

PoissonInv

PoissonInv() 返回累计泊松概率分布的倒数。

语法：

```
PoissonInv(prob, mean)
```

返回数据类型：数字

参数

参数	描述
prob	与泊松概率分布相关联的概率。必须为一个介于 0 和 1 之间的值。
mean	定义平均结果的正数。

此函数通过以下方式与 PoissonDIST 函数关联：
If prob = PoissonDist(value, mean), then PoissonInv(prob, mean) = value

TDensity

TDensity() 返回学生的 t 密度函数的值，其中数值是要对其计算概率的 t的计算值。

语法：

```
TDensity(value, degrees_freedom)
```

返回数据类型： 数字

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。

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。值必须不能为负数。
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

8.24 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()** 函数将文本字符串中每个单词的第一个字符转换为大写，并将所有其他字符转换为小写。

```
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()** 用于返回组合输入表达式值的 128 位哈希值。结果为 22 个字符的字符串。

```
Hash128 (expr{, expression})
```

Hash160

Hash160() 返回组合输入表达式值的 160 位哈希值。结果为 27 个字符的字符串。**Hash160()** 返回组合输入表达式值的 160 位哈希值。结果为 27 个字符的字符串。

```
Hash160 (expr{, expression})
```

Hash256

Hash256() 返回组合输入表达式值的 256 位哈希值。结果为 43 个字符的字符串。**Hash256()** 返回组合输入表达式值的 256 位哈希值。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])
```

IsJson

IsJson() 测试指定字符串是否包含有效的 JSON(JavaScript 对象表示法)数据。您还可以验证特定的 JSON 数据类型。

```
IsJson (json [, type])
```

JsonGet

JsonGet() 返回 JSON(JavaScript 对象表示法)数据字符串的路径。数据必须是有效的 JSON，但可以包含额外的空格或换行符。

```
JsonGet (json, path)
```

JsonSet

JsonSet() 修改包含 JSON(JavaScript 对象表示法)数据的字符串。它可以使用路径指定的新位置设置或插入 JSON 值。数据必须是有效的 JSON，但可以包含额外的空格或换行符。

```
JsonSet(json, path, value)
```

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()** 返回字符串第一个字符的数字值（ASCII 或 Unicode）。此函数可用于根据字符串的底层字符代码对其进行评估或比较，例如，在对包含非标准字符的字符串进行排序或筛选时。

```
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。**Capitalize()** 函数将文本字符串中每个单词的第一个字符转换为大写，并将所有其他字符转换为小写。

语法：

```
Capitalize(text)
```

返回数据类型： 字符串

示例：加载脚本和图表表达式

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

更多示例和结果

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。

加载脚本和结果

```
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。**Hash128()** 用于返回组合输入表达式值的 128 位哈希值。结果为 22 个字符的字符串。哈希值可用于屏蔽个人身份信息 (PII), 如客户姓名、社会保障号码或帐号。

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+'。

加载脚本和结果

```
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

示例:加载脚本和结果

```
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() 返回组合输入表达式值的 160 位哈希值。结果为 27 个字符的字符串。
Hash160() 返回组合输入表达式值的 160 位哈希值。结果为 27 个字符的字符串。哈希值可用于屏蔽个人身份信息 (PII), 如客户姓名、社会保障号码或帐号。

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?<$'AI.)?U$'。

加载脚本和结果

```
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

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() 返回组合输入表达式值的 256 位哈希值。结果为 43 个字符的字符串。
Hash256() 返回组合输入表达式值的 256 位哈希值。结果为 43 个字符的字符串。哈希值可用于屏蔽个人身份信息 (PII), 如客户姓名、社会保障号码或帐号。

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:'。

加载脚本和结果

```
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。 *如果文本中不存在子串，索引将返回 0。*
count	定义搜索出现的哪个 **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
Index('abc', 'x')	返回 0('abc' 字符串中不存在 'x')。
Index('abc', 'a', 2)	返回 0('a' 没有第二次出现)。

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 ];
```

IsJson

IsJson() 测试指定字符串是否包含有效的 JSON(JavaScript 对象表示法)数据。您还可以验证特定的 JSON 数据类型。

语法：

```
value IsJson(json [, type])
```

返回数据类型： 双

参数

参数	描述
json	要测试的字符串。它可以包含额外的空格或换行符。
type	可选参数，指定要测试的 JSON 数据类型。 • 'value'(默认值) • 'object' • 'array' • 'string' • 'number' • 'Boolean' • 'null'

示例:有效的 JSON 和类型

示例	结果
`IsJson('null')`	返回 -1 (true)
`IsJson('"abc"', 'value')`	返回 -1 (true)
`IsJson('"abc"', 'string')`	返回 -1 (true)
`IsJson(123, 'number')`	返回 -1 (true)

示例:有效的 JSON 或类型

示例	结果	描述
`IsJson('text')`	返回 0 (false)	'text' 不是有效的 JSON 值
`IsJson('"text"', 'number')`	返回 0 (false)	'"text"' 是有效的 JSON 数字
`IsJson('"text"', 'text')`	返回 0 (false)	'text' 不是有效的 JSON 类型

JsonGet

JsonGet() 返回 JSON(JavaScript 对象表示法) 数据字符串的路径。数据必须是有效的 JSON, 但可以包含额外的空格或换行符。

语法:

```
value JsonGet(json, path)
```

返回数据类型: 双

参数

参数	描述
json	包含 JSON 数据的字符串。
path	必须根据 RFC 6901 指定路径。这将允许在 JSON 数据中查找属性, 而无需使用复杂的子字符串或索引函数。

示例:有效的 JSON 和路径

示例	结果
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '')`	返回 '{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}'
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/a')`	返回 '{"foo":"bar"}'
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/a/foo')`	返回 '"bar"'

示例	结果
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b')`	返回 '[123,"abc","ABC"]'
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b/0')`	返回 '123'
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b/1')`	返回 '"abc"'
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b/2')`	返回 '"ABC"'

示例:有效的 JSON 或路径

示例	结果	描述
`JsonGet('{"a":"b"}','/b')`	返回 null	路径未指向 JSON 数据的有效部分。
`JsonGet('{"a"}','/a')`	返回 null	JSON 数据不是有效的 JSON(成员“a”没有值)。

JsonSet

JsonSet() 修改包含 JSON(JavaScript 对象表示法)数据的字符串。它可以使用路径指定的新位置设置或插入 JSON 值。数据必须是有效的 JSON,但可以包含额外的空格或换行符。

语法:

```
value JsonSet(json, path, value)
```

返回数据类型: 双

参数

参数	描述
json	包含 JSON 数据的字符串。
path	必须根据 RFC 6901 指定路径。这允许在 JSON 数据中构建属性,而无需使用复杂的子字符串或索引函数以及串联。
value	JSON 格式的新字符串值。

示例:有效的 JSON、路径和值

示例	结果
`JsonSet('{}','/a','"b"')`	返回 '{"a":"b"}'
`JsonSet('[]','/0','"x"')`	返回 '["x"]'
`JsonSet('"abc"','','123')`	返回 123

示例:有效的 JSON、路径或值

示例	结果	描述
`JsonSet('"abc"','/x','123')`	返回 null	路径未指向 JSON 数据的有效部分。
`JsonSet('{"a":{"b":"c"}}','a/b','"x"')`	返回 null	路径无效。
`JsonSet('{"a":"b"}','/a','abc')`	返回 null	值不是有效 JSON。字符串必须用引号括起来。

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:加载脚本和结果

```
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 1408)

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 - 左侧高级场景

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。

Text	Start	靠左
abcdef	3	abc
2021-07-14	4	2021
2021-07-14	2	20

另请参见*Index (page 1394)*, 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'

图表表达式

概述

打开数据加载编辑器,并将下面的加载脚本添加到新选项卡。

加载脚本包含:

-
- 数据表中有一个名为InputText 的字段。

加载脚本

```
Example:
Load * inline [
InputText
Sliver
SSiver
SSiveer
];
```

结果

加载数据并打开工作表。创建新表并将该字段添加为维度:

- InputText
- =LevenshteinDist('Silver', InputText),用于计算将 InputText 的字符串值更改为单词 'Silver' 所需的最小单字符编辑次数。

结果表

InputText	LevenshteinDist('Silver', InputText)
Sliver	2
SSiveer	3
SSiver	2

LevenshteinDist 函数的输出返回将 InputText 更改为预期文本 'Silver' 所需的更改次数。例如，第一行需要两次更改才能将单词 'Sliver' 修改为 'Silver'。第二行需要 3 个更改：1) 删除多余的字符 'S'。2) 删除多余的字符 'e'。3) 插入新字符 'l'。

图表表达式

概述

此示例整合了来自不同系统的产品名称。由于拼写错误、缩写、间距或其他变化，产品名称并不总是使用相同的拼写。使用 LevenshteinDist 函数，您可以测量两个产品名称之间的相似性，并确定哪些名称可能指代同一产品，即使名称不完全相同。

打开数据加载编辑器，并将下面的加载脚本添加到新选项卡。

加载脚本包含：

-
-
    - ProductA
    - ProductB

加载脚本

```
Example:
Load * inline [
ProductA, ProductB
Coca Cola 330ml, CocaCola 330 ml
Pepsi 500 ml, Pepsi 500ml
Sprite Zero 600 ml, SpriteZero600ml
Red Bull 250ml, Redbull 250ml
];
```

结果

加载数据并打开工作表。创建新表并将这些字段添加为维度：

- ProductA
- ProductB

- =LevenshteinDist(ProductA, ProductB)，用于计算将 ProductB 的字符串值更改以匹配 ProductA 所需的最小单字符编辑次数。

结果表

ProductA	ProductB	LevenshteinDist(ProductA, ProductB)
Coca Cola 330ml	CocaCola 330 ml	2
Pepsi 500 ml	Pepsi 500ml	1
Red Bull 250ml	Redbull 250ml	2
Sprite Zero 600 ml	SpriteZero600ml	3

编辑距离是一种模糊匹配，在客户数据管理、库存系统和文档处理等领域被广泛用作拼写检查器、光学字符识别和校正系统的一部分，在这些领域，文本的细微变化经常发生。

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 200)。

结果

字符串	StringLength	LtrimStringLength
def	6	5
abc	10	7

另请参见：

- RTrim (page 1412)

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。

Text	Start	Mid1	Count	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 1394)

Ord

Ord() 用于返回输入字符串第一个字符的 Unicode 代码点数。**Ord()** 返回字符串第一个字符的数字值(ASCII 或 Unicode)。此函数可用于根据字符串的底层字符代码对其进行评估或比较,例如,在对包含非标准字符的字符串进行排序或筛选时。

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 1398)

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。

Text	Start	靠右
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 200)**。*

结果

字符串	StringLength	RtrimStringLength
def	6	4
abc	10	6

另请参见：

- *LTrim (page 1405)*

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：图表表达式

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。

示例	结果
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。

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:

Load Name,
SubField(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

Instrument	Player	Project
Synth	Jen	Video
Synth	Jen	OST
Synth	Jo	Music
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

Text	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;
Load *, trim(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 200)。

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

示例 – 上部场景

8.25 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。。*

GetSysAttr

此函数返回所选应用程序的租户和空间域属性。此函数均可用于数据加载脚本和图表表达式。

```
GetSysAttr (name)
```

> *如果在 Qlik Sense 客户端托管 中使用此函数，它将只返回空的数据值。*

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 ()
```

InObject

InObject() 图表函数计算当前对象是否包含在具有函数参数中指定 ID 的另一个对象中。对象可以是工作表或可视化。

```
InObject - 图表函数(id_str)
```

ObjectId

ObjectId() 图表函数返回计算表达式的对象的 ID。该函数采用一个可选参数，指定该函数所关注的对象类型。对象可以是工作表或可视化。此函数仅在图表表达式中可用。

```
ObjectId - 图表函数([object_type_str])
```

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 Client-Managed Mobile 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()
```

GetSysAttr

此函数返回所选应用程序的租户和空间域属性。此函数均可用于数据加载脚本和图表表达式。

如果在 Qlik Sense 客户端托管 中使用此函数，它将返回空的数据值。因此，您可以使用该功能在 Qlik Sense 客户端托管 中开发加载脚本，而不会遇到错误，以便稍后将应用程序上载到 Qlik Cloud。

要访问 Qlik Cloud 函数的完整文档，请参阅 GetSysAttr - 脚本和图表函数。

InObject - 图表函数

InObject() 图表函数计算当前对象是否包含在具有函数参数中指定 ID 的另一个对象中。对象可以是工作表或可视化。

此函数可用于显示工作表中对象的层次结构，从顶层工作表对象到嵌套在其他可视化中的可视化。此函数可以与 **if** 和 **ObjectId** 函数一起使用，以在应用程序中创建自定义导航。

语法：

```
InObject(id_str)
```

返回数据类型：布尔值

在 Qlik Sense 中，布尔 true 值由 -1 表示，false 值由 0 表示。

参数

参数	描述
id_str	表示正在计算的对象 ID 的字符串值。

可以从应用程序 URL 获取工作表 ID。对于可视化，请使用**开发者**选项来标识对象 ID 和对象类型的文本字符串。

执行以下操作：

1. 在分析模式下，将以下文本添加到 URL 中：
   /options/developer
2. 右键单击可视化，然后单击 **开发者**。
3. 在**属性**下，从对话框标题获取对象 ID，从**“qType”**属性获取对象类型。

限制：

当在作为主条目的容器内的对象（例如按钮）中调用此函数时，可能会产生意外的结果。此限制也适用于筛选器窗格主条目，这些主条目是多个列表框的容器。这是因为主条目使用对象层次结构的方式。

InObject() 通常与以下函数结合使用：

相关函数

函数	交互
if (page 533)	**if** 和 **ObjectId** 函数可以一起用于创建条件表达式。例如，可视化可以通过使用这些函数的表达式实现条件着色。
ObjectId - 图表函数 (page 1426)	与 **if** 类似，**ObjectId** 也与 **InObject** 一起用于创建条件表达式。

示例 1 – 基本功能

图表表达式和结果

下面的基本示例演示了如何确定一个对象是否包含在另一个对象中。在这种情况下，我们将使用工作表的 ID 作为参数检查**文本和图像**对象是否驻留在工作表对象中。

执行以下操作：

1. 打开新工作表并将**文本和图像**图表拖到工作表上。
2. 在属性面板中，单击**添加度量**。
3. 单击 $f\times$ 以打开表达式编辑器。
4. 将以下表达式粘贴到对话框中：
   `=InObject()`
5. 修改表达式以将工作表的 ID 作为字符串包含在括号之间。
   例如，对于 ID 为 1234-5678 的工作表，可以使用以下内容：
   `=InObject('1234-5678')`
6. 单击**应用**。

值 -1 显示在图表中，表示表达式的计算结果为真。

示例 2 – 具有条件颜色的对象

图表表达式和结果

概述

以下示例演示如何创建显示不同颜色的自定义导航按钮，以指示当前打开的工作表。

首先创建一个新应用程序并打开数据加载编辑器。将以下加载脚本粘贴到新选项卡中。请注意，数据本身是占位符，不会在示例内容中使用。

加载脚本

```
Transactions:

Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'4/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'7/26/2022',45.89
```

```
8198,'8/9/2022',36.23
8199,'9/22/2022',25.66
8200,'11/23/2022',82.77
8201,'12/27/2022',69.98
8202,'1/1/2023',76.11
8203,'2/8/2022',25.12
8204,'3/19/2022',46.23
8205,'6/26/2022',84.21
8206,'9/14/2022',96.24
8207,'11/29/2022',67.67
];
```

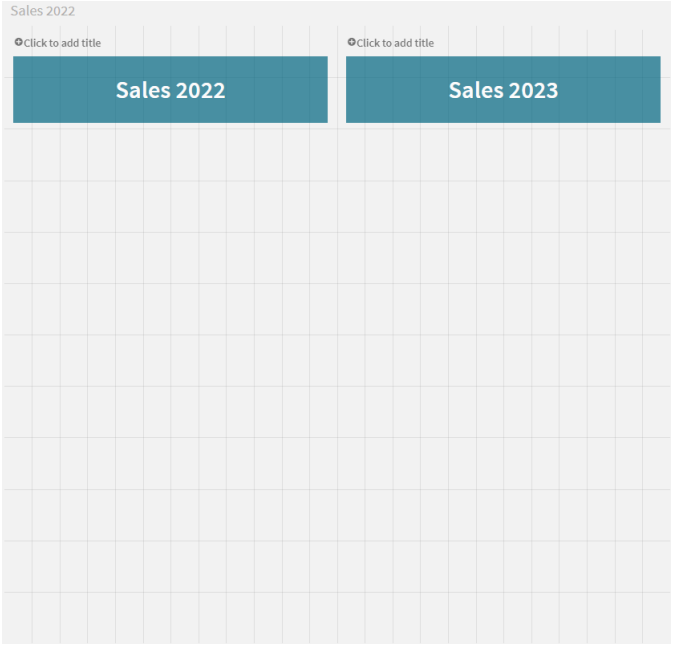
创建可视化

加载数据并创建两个新工作表。将它们的标题分别命名为 *Sales 2022* 和 *Sales 2023*。

接下来，构建两个按钮对象，用于在两个工作表之间导航。

执行以下操作：

1. 向工作表中添加两个**按钮**对象。
2. 在**外观 > 常规**下，将每个按钮的**标签**分别设置为 *Sales 2022* 和 *Sales 2023*。
3. 排列按钮以匹配下图。
   带有两个导航按钮的Sales 2022 工作表排列



4. 选择 *Sales 2022* 按钮，然后在属性面板中展开**操作和导航**。
5. 单击**添加操作**，然后在**导航**下选择**转到工作表**。
6. 在**工作表**下，选择 *Sales 2022*。
7. 重复此按钮操作设置，将 **Sales 2023** 按钮链接到 *Sales 2023* 工作表。
8. 通过右键单击按钮并选择 **添加到主条目**，将按钮转换为主条目。

现在，您可以复制每个按钮并将其粘贴到 *Sales 2023* 工作表中，使用工作表上相同的大小和排列。

创建条件颜色

接下来，配置按钮，使其在链接到当前打开的工作表时为蓝色，在链接到未打开的工作时为浅灰色。

执行以下操作：

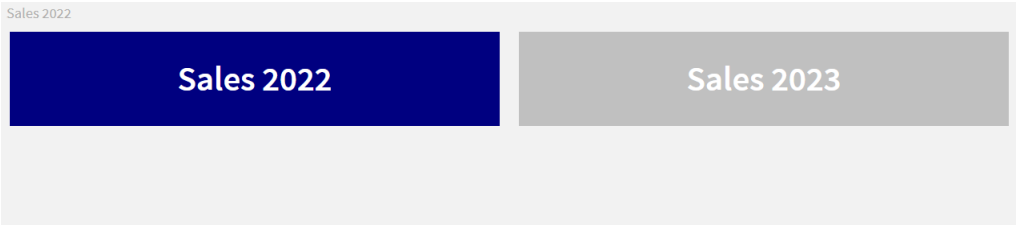
1. 打开 *Sales 2022* 工作表并从 URL 获取工作表 ID。保持 *Sales 2022* 工作表打开。
2. 单击 **Sales 2022** 按钮主条目，然后在属性面板中选择**编辑**。
3. 在**外观 > 背景**下，选择**按表达式**为按钮着色。
4. 在**表达式**中，粘贴以下文本：
   =if(InObject(''), Blue(), LightGray())
5. 在上面表达式中的括号之间，粘贴 *Sales 2022* 工作表的工作表 ID。

现在，该按钮被配置为当 *Sales 2022* 工作表打开时变为蓝色，而当未打开时则变为浅灰色。

对 *Sales 2023* 工作表重复上述说明，将 **Sales 2023** 按钮主条目链接到 *Sales 2023* 工作表 ID。

每个工作表现在应有两个按钮，以蓝色指示当前打开的工作表。

Sales 2022 工作表，蓝色，表示当前显示 2022 年销售额



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()
```

ObjectId - 图表函数

ObjectId() 图表函数返回计算表达式的对象的 ID。该函数采用一个可选参数，指定该函数所关注的对象类型。对象可以是工作表或可视化。此函数仅在图表表达式中可用。

语法：

```
ObjectId([object_type_str])
```

返回数据类型： 字符串

函数的唯一参数 **object_type_str** 是可选的，它引用表示对象类型的字符串值。

参数

参数	描述
object_type_str	表示正在计算的对象类型的字符串值。

如果函数表达式中未指定参数，**ObjectId()** 将返回使用表达式的对象的 ID。要返回显示可视化的工作表对象的 ID，请使用 *ObjectId('sheet')*。

如果可视化对象嵌套在其他可视化对象中，请在函数参数中为不同的结果指定所需的对象类型。例如，对于容器中的**文本和图像**图表，使用 *'text-image'* 返回**文本和图像**对象，使用 *'container'* 返回容器的 ID。

执行以下操作：

1. 在分析模式下，将以下文本添加到 URL 中：
   /options/developer
2. 右键单击可视化，然后单击 **开发者**。
3. 在**属性**下，从对话框标题获取对象 ID，从**"qType"**属性获取对象类型。

限制：

当在作为主条目的容器内的对象（例如按钮）中调用此函数时，可能会产生意外的结果。此限制也适用于筛选器窗格主条目，这些主条目是多个列表框的容器。这是因为主条目使用对象层次结构的方式。

在这些情况下，图表表达式 *ObjectId('sheet')* 将返回一个空字符串，而 *ObjectId('masterobject')* 将显示所属主条目的标识符。

ObjectId() 通常与以下函数结合使用：

相关函数

函数	交互
if (page 533)	**if** 和 **ObjectId** 函数可以一起用于创建条件表达式。例如，可视化可以通过使用这些函数的表达式实现条件着色。
InObject - 图表函数 (page 1422)	与 **if** 类似，**InObject** 也与 **ObjectId** 一起用于创建条件表达式。

示例 1 – 返回图表对象 ID

图表表达式和结果

下面的基本示例演示了如何返回可视化的 ID。

执行以下操作:

1. 打开新工作表并将**文本和图像**图表拖到工作表上。
2. 在属性面板中, 单击**添加度量**。
3. 单击 $f(x)$ 以打开表达式编辑器。
4. 将以下表达式粘贴到对话框中:
   ```
   =ObjectId()
   ```
5. 单击**应用**。

文本和图像对象的 ID 显示在可视化中。

使用以下表达式可以获得相同的结果:

```
=ObjectId('text-image')
```

示例 2 – 返回表 ID

图表表达式和结果

下面的基本示例演示如何返回显示可视化的工作表的 ID。

执行以下操作:

1. 打开新工作表并将**文本和图像**图表拖到工作表上。
2. 在属性面板中, 单击**添加度量**。
3. 单击 $f(x)$ 以打开表达式编辑器。
4. 将以下表达式粘贴到对话框中:
   ```
   =ObjectId('sheet')
   ```
5. 单击**应用**。

工作表的 ID 将显示在可视化中。

示例 3 – 嵌套表达式

图表表达式和结果

以下示例显示了 **ObjectId()** 函数如何嵌套在其他表达式中。

执行以下操作:

1. 打开新工作表并将**文本和图像**图表拖到工作表上。
2. 在属性面板中, 单击**添加度量**。
3. 单击 $f(x)$ 以打开表达式编辑器。

4. 将以下表达式粘贴到对话框中：
   =if(InObject(ObjectId('text-image')), 'In Text & image', 'Not in Text & image')
5. 单击**应用**。

In Text & image 文本显示在图表中，表示表达式中引用的对象是**文本和图像**图表。

有关使用条件着色的更详细示例，请参见*InObject - 图表函数 (page 1422)*。

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)
      )
)
```

8.26 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');
```

8.27 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;
```

8.28 窗口函数

窗口函数使用多行中的值执行计算，以分别为每行生成一个值。只有在读取了整个表之后，才能计算窗口函数。

您可以使用窗口函数执行以下操作：

- 将行中的单个数值与列中的平均值、最大值或最小值进行比较。
- 计算列中或整个表中单个值的秩。

窗口函数不会改变表中记录的数量，但可以执行与聚合函数或关系函数和范围函数类似的任务。

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。

Window

Window 函数从多行执行计算，分别为每行生成一个值。

```
Window (input_expr, [partition1, partition2, ...], [sort_type, [sort_expr]],
[filter_expr], [start_expr,end_expr])[row_window_size])
```

WRank

WRank 函数在 **Window** 内部执行排名计算。

```
WRank ([TOTAL] expr[, mode[, fmt]])
```

Window

Window() 从多行执行计算，分别为每行生成一个值。

您可以使用 **Window** 函数执行以下操作：

- 将行中的单个数值与列中的平均值、最大值或最小值进行比较。
- 计算列中或整个表中单个值的秩。

Window 函数不会更改表中记录的数量，但它仍然可以执行与聚合、关系和范围函数类似的任务。

Window 函数必须在要添加到表中的表的 LOAD 语句中具有缓存。例如：

```
[Transactions]:
Load
       *,
       Window(avg(Expression1),[Num]);
LOAD
       TransLineID,
       TransID,
       "Num",
       Dim1,
       Dim2,
       Dim3,
       Expression1,
       Expression2,
       Expression3
FROM [lib://AttachedFiles/transactions.qvd] (qvd);
```

窗口支持一般函数，如舍入或基本的数值运算。例如：

```
Load *, Round(Window(Sum(Salary),Department)) as SumSalary
Load *, Window(Sum(Salary),Department) + 5 as SumSalary
```

您可以为 **Window** 函数定义一个滑动窗口。这将设置在当前行上应用 **Window** 函数时使用的行数。例如，可以将窗口设置为前 3 行和后 3 行。

语法：

```
Window (input_expr, [partition1, partition2, ...], [sort_type, [sort_expr]],
[filter_expr], [start_expr,end_expr])
```

返回数据类型：添加到 LOAD 语句创建的结果表中的新字段。

参数：

参数

参数	描述

input_expr	函数计算和返回的输入表达式。它必须是基于聚合的任何表达式，例如Median (Salary)。例如：
	`Window(Median(Salary)) as MedianSalary`
	输入也可以是不应用聚合的字段名。在这种情况下，**Window**将其视为 **Only()** 函数应用于该字段。例如：
	`Window(Salary,Department) as WSalary`
	
	您可以选择使用输入表达式定义分区。分区与 **group by** 子句实现的分组相同，不同之处在于将结果作为新列添加到输入表中。分区不会减少输入表的记录数。可以定义多个分区字段。
	示例：
	`LOAD`
	`Window(Max(Sales), City, 'ASC', OrderDate, Sales > 300)`
	`+ AddMonths(OrderDate,-6) as MAX_Sales_City_Last_6_Mos,`
	`Window(Avg(Sales), City, 'ASC', OrderDate, City = 'Portland')`
	`+ AddMonths(OrderDate,-6) as Avg_Sales_Portland_Last_6_Mos,`
	`Window(Max(Sales), City, 'ASC', OrderDate, Sales > 300)`
	`+ AddMonths(OrderDate,-12) as MAX_Sales_City_Last_12_Mos;`
	`LOAD`
	`City,`
	`Sales,`
	`OrderDate`
	`FROM [lib://AttachedFiles/Sales Data.xlsx]`
	`(ooxml, embedded labels, table is [Sales Data]);`

partition1, partition2	input_expr之后，您可以定义任意数量的分区。分区是定义应用聚合的组合的字段。聚合是对每个分区单独应用的。例如： `Window(Avg(Salary), Unit, Department, Country) as AvgSalary` 在上面的例子中，分区是 *Unit*、*Department* 和 *Country*。 分区不是强制性的，但对于字段的正确窗口化是必需的。

参数	描述
sort_type, [sort_expr]]	也可以指定排序类型和排序表达式。sort_type 可以具有两个值之一： • ASC：升序排序。 • DESC：降序排序。 如果定义了 sort_type，则需要定义一个排序表达式。这是一个决定分区中行的顺序的表达式。 例如： `Window(RecNo(), Department, 'ASC', Year)` 在上面的例子中，分区内的结果按 *Year* 字段升序排列。 *排序类型和排序表达式主要只有 **RecNo** 和 **WRank** 函数才需要。*
filter_expr	也可以添加筛选器表达式。这是一个布尔表达式，用于决定记录是否应包含在计算中。 这个参数可以完全省略，结果应该是没有筛选器。 例如： `Window(avg(Salary), Department, 'ASC', Age, EmployeeID=3 Or EmployeeID=7) as wAvgSalary) as wAvgSalaryIfEmpIs3or7`

[start_expr,end_expr]	也可以设置滑动窗口功能的参数。滑动窗口需要两个参数：
	• 开始表达式：要包含在窗口中的当前行之前的行数。
	• 结束表达式：要包含在窗口中的当前行之后的行数。
	例如，如果要包括前面的 3 行、当前行和下面的下一行：
	`Window(concat(Text(Salary),'-'), Department, 'ASC', Age, Year>0,  -3, 1) as WSalaryDepartment`
	要指示所有前面的行或所有之前的行，可以使用 **Unbounded()** 函数。例如，包括所有前面的行、当前行和下面的行：
	`Window(concat(Text(Salary),'-'), Department, 'ASC', Age, Year>0,  UNBOUNDED(), 1) as WSlidingSalaryDepartment`
	例如，要包括当前行的第三行和所有后续行：
	`Window(concat(Text(Salary),'-'), Department, 'ASC', Age, Year>0,  3, UNBOUNDED()) as WSlidingSalaryDepartment`

示例 - 添加包含聚合的字段

示例:添加包含聚合的字段

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。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions:
Load
*,
Window(Avg(transaction_amount),customer_id) as AvgCustTransaction;

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
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

结果

添加包含聚合的字段的结果

transaction_id	transaction_date	transaction_amount	transaction_quantity	CustomerID	大小	color_code	AvgCustTransaction
3750	20180830	23.56	2	2038593	L	红色	103.43
3751	20180907	556.31	6	203521	M	橙色	266.775
3752	20180916	5.75	1	5646471	S	蓝色	42.935
3753	20180922	125.00	7	3036491	L	黑色	64.21
3754	20180922	484.21	13	049681	XS	红色	273.88
3756	20180922	59.18	2	2038593	M	蓝色	103.43

transaction_id	transaction_date	transaction_amount	transaction_quantity	CustomerID	大小	color_code	AvgCustTransaction
3757	20180923	177.42	21	203521	XL	黑色	266.775
3758	20180924	153.42	14	2038593	L	红色	103.43
3759	20180925	7.42	5	203521	M	橙色	266.775
3760	20180925	80.12	18	5646471	M	蓝色	42.935
3761	20180926	3.42	7	3036491	XS	黑色	64.21
3763	20180926	63.55	12	049681	S	红色	273.88
3763	20180927	177.56	10	2038593	L	蓝色	103.43
3764	20180927	325.95	8	203521	XL	黑色	266.775

示例 - 添加包含为特定值筛选的聚合的字段

示例:添加包含为特定值筛选的聚合的字段

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。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions:
Load
*,
Window(Avg(transaction_amount),customer_id, color_code = 'Blue') as AvgCustTransaction;

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
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

结果

添加包含针对特定值筛选的聚合的广告字段的结果

transaction_id	transaction_date	transaction_amount	transaction_quantity	CustomerID	大小	color_code	AvgCustTransaction
3750	20180830	23.56	2	2038593	L	红色	-
3751	20180907	556.31	6	203521	M	橙色	-
3752	20180916	5.75	1	5646471	S	蓝色	42.94
3753	20180922	125.00	7	3036491	L	黑色	-
3754	20180922	484.21	13	049681	XS	红色	-
3756	20180922	59.18	2	2038593	M	蓝色	118.4
3757	20180923	177.42	21	203521	XL	黑色	-
3758	20180924	153.42	14	2038593	L	红色	-
3759	20180925	7.42	5	203521	M	橙色	-
3760	20180925	80.12	18	5646471	M	蓝色	42.94
3761	20180926	3.42	7	3036491	XS	黑色	-
3763	20180926	63.55	12	049681	S	红色	-
3763	20180927	177.56	10	2038593	L	蓝色	118.4
3764	20180927	325.95	8	203521	XL	黑色	-

示例 - 添加带有滑动窗口的字段

示例:添加带有滑动窗口的字段

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。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions:
Load
*,
Window(Avg(transaction_amount),customer_id, 'ASC', -1, 1, 0, 1) as AvgCustTransaction;

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
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

结果

添加包含针对特定值筛选的聚合的广告字段的结果

transaction_id	transaction_date	transaction_amount	transaction_quantity	CustomerID	大小	color_code	AvgCustTransaction
3750	20180830	23.56	2	2038593	L	红色	41.37
3751	20180907	556.31	6	203521	M	橙色	366.865
3752	20180916	5.75	1	5646471	S	蓝色	42.935
3753	20180922	125.00	7	3036491	L	黑色	64.21
3754	20180922	484.21	13	049681	XS	红色	273.88
3756	20180922	59.18	2	2038593	M	蓝色	106.3
3757	20180923	177.42	21	203521	XL	黑色	92.42
3758	20180924	153.42	14	2038593	L	红色	165.49
3759	20180925	7.42	5	203521	M	橙色	166.685
3760	20180925	80.12	18	5646471	M	蓝色	80.12
3761	20180926	3.42	7	3036491	XS	黑色	3.42
3763	20180926	63.55	12	049681	S	红色	177.56
3763	20180927	177.56	10	2038593	L	蓝色	63.55
3764	20180927	325.95	8	203521	XL	黑色	325.95

限制

Window 存在以下限制：

- **Window** 是一个资源密集型函数，特别是在内存消耗方面。
- **Window** 在 Qlik Sense Mobile 中不受支持。
- 图表表达式不支持**Window**。

- 不能在其他 **Window** 函数中嵌套 **Window** 函数。
- **Window** 不能在聚合函数内部使用。
- **Window** 需要能够扫描整个表格。
- **WRank()**、**RecNo()** 和 **RowNo()** 使用滑动窗口功能时无法与 **Window** 一起使用.

WRank

WRank() 评估加载脚本中表的行，并针对每一行显示加载脚本中评估的字段值的相对位置。在评估表时，函数将结果与包含当前分区的其他行的结果进行比较，并返回当前行在段内的排名。

表中的分区

WRank 只能在 **Window** 函数中使用。**Window** 函数必须包含排序类型和排序表达式。排序将应用于排序表达式。

语法：

```
WRank ([mode[, fmt]])
```

返回数据类型：双

参数：

参数

参数	描述
mode	也指定函数结果的数字表示形式。
fmt	也指定函数结果的文本表示形式。
TOTAL	如果表格是一维或如果脚本前面有 **TOTAL** 限定符，则该函数用于评估整列。如果表或等效表具有多个垂直维度，则除显示字段间排序顺序中最后一个维度的列外，当前分区将在所有维度列中仅包括与当前行具有相同值的行。

排名作为一个双值返回，在每一行都有唯一排名的情况下，该值是当前分区中 1 和行数之间的整数。

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

第一个参数 **mode** 可获取以下值：

mode 值

值	描述
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。 如果共享组内的排名跨越整个排名的中间值，则所有行都会获得与整个分区中最高排名和最低排名的平均值相对应的值。
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

第二个参数 **fmt** 可获取以下值：

fmt 值

值	描述
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 的行顺序由表字段的加载顺序决定。

示例 - 添加排序字段

示例：添加排序字段

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。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions:
Load
*,
Window(WRank(0),customer_id, 'Desc', transaction_amount) as TransactionRanking;

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
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

结果

添加排序字段的结果

transaction_id	transaction_date	transaction_ amount	transaction_ quantity	CustomerID	大小	color_ code	TransactionRanking
3750	20180830	23.56	2	2038593	L	红色	4-4
3751	20180907	556.31	6	203521	M	橙色	1-1
3752	20180916	5.75	1	5646471	S	蓝色	2-2
3754	20180922	484.21	13	049681	XS	红色	1-1
3756	20180922	59.18	2	2038593	M	蓝色	3-3
3753	20180922	125.00	7	3036491	L	黑色	1-1
3757	20180923	177.42	21	203521	XL	黑色	3-3
3758	20180924	153.42	14	2038593	L	红色	2-2
3759	20180925	7.42	5	203521	M	橙色	4-4
3760	20180925	80.12	18	5646471	M	蓝色	1-1
3763	20180926	63.55	12	049681	S	红色	2-2
3761	20180926	3.42	7	3036491	XS	黑色	2-2
3764	20180927	325.95	8	203521	XL	黑色	2-2
3763	20180927	177.56	10	2038593	L	蓝色	1-1

示例 - 使用 fmt 添加排名字段以获得单个位数的结果

示例:使用 fmt 添加排名字段以获得单个位数的结果

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。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions:
Load
*,Window(WRank(0,1),customer_id, 'Desc', transaction_amount) as TransactionRanking;

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
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

结果

使用 fmt 添加排名字段以获得单个位数的结果所得的结果

transaction_id	transaction_date	transaction_ amount	transaction_ quantity	CustomerID	大小	color_ code	TransactionRanking
3750	20180830	23.56	2	2038593	L	红色	4
3751	20180907	556.31	6	203521	M	橙色	1
3752	20180916	5.75	1	5646471	S	蓝色	2
3754	20180922	484.21	13	049681	XS	红色	1
3756	20180922	59.18	2	2038593	M	蓝色	3
3753	20180922	125.00	7	3036491	L	黑色	1
3757	20180923	177.42	21	203521	XL	黑色	3

transaction_id	transaction_date	transaction_amount	transaction_quantity	CustomerID	大小	color_code	TransactionRanking
3758	20180924	153.42	14	2038593	L	红色	2
3759	20180925	7.42	5	203521	M	橙色	4
3760	20180925	80.12	18	5646471	M	蓝色	1
3763	20180926	63.55	12	049681	S	红色	2
3761	20180926	3.42	7	3036491	XS	黑色	2
3764	20180927	325.95	8	203521	XL	黑色	2
3763	20180927	177.56	10	2038593	L	蓝色	1

示例 - 添加具有多个分区的已排序字段

示例:添加具有多个分区的已排序字段

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。在 Qlik Sense 中创建下面的表格以查看结果。

```
Transactions:
Load
*,Window(WRank(0,1),customer_id, size, color_code, 'Desc', transaction_amount) as
TransactionRanking;

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
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

结果

使用 fmt 添加排名字段以获得单个位数的结果所得的结果

transaction_id	transaction_date	transaction_amount	transaction_quantity	CustomerID	大小	color_code	TransactionRanking
3750	20180830	23.56	2	2038593	L	红色	2
3751	20180907	556.31	6	203521	M	橙色	1
3752	20180916	5.75	1	5646471	S	蓝色	1
3754	20180922	484.21	13	049681	XS	红色	1
3756	20180922	59.18	2	2038593	M	蓝色	1
3753	20180922	125.00	7	3036491	L	黑色	1
3757	20180923	177.42	21	203521	XL	黑色	2
3758	20180924	153.42	14	2038593	L	红色	1
3759	20180925	7.42	5	203521	M	橙色	2
3760	20180925	80.12	18	5646471	M	蓝色	1
3763	20180926	63.55	12	049681	S	红色	1
3761	20180926	3.42	7	3036491	XS	黑色	1
3764	20180927	325.95	8	203521	XL	黑色	1
3763	20180927	177.56	10	2038593	L	蓝色	1

限制

WRank 存在以下限制：

- 如果您的 fmt 值为 0，并且希望使用 **WRank** 偶结果的文本部分于，则必须将 **Text()** 与 **Window(WRank)** 一起使用。例如：`Text(Window(WRank(0), Unit, 'DESC', Age)) as UnitWRankedByAgeText`。

9 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 1454)

9.1 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)。

9.2 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、Parquet、CSV 或 TXT 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** 函数以 UTC 格式返回指定文件上次修改的时间戳。如果未指定文件,函数将返回当前读取的表文件的最后修改的 UTC 时间戳。

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。在时间戳中,时间以 UTC 表示。
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。

9.3 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)

10 图表级别脚本

修改图表数据时，使用 Qlik Sense脚本的子集，该子集由许多语句组成。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。

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。

在本节中，您可以在修改图表数据时使用的脚本子集中找到按字母顺序排列的所有脚本语句、控制语句和前缀列表。

10.1 控制语句

修改图表数据时，使用 Qlik Sense脚本的子集，该子集由许多语句组成。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。

前缀从不应用于控制语句。

所有脚本关键字可以大小写字符的任意组合输入。

图表修改器控制语句概述

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 ]
```

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。

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

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]
```

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]
```

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
```

Next

Next 脚本关键字用于结束 **For** 循环。

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
```

To

To 脚本关键字可用于多个脚本语句。

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。

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。

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。

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。*
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。

10.2 前缀

前缀可用于常规语句，但不可用以控制语句。

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

图表修改器前缀概述

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
```

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
```

Add

在图表修改上下文中，**Add**前缀结合 **LOAD** 将值附加到 *HC1* 表，表示由 Qlik 关联引擎 计算的超立方体。可以指定一列或多列。缺少的值由 Qlik 关联引擎 自动填充。

语法：

```
Add loadstatement
```

示例：

此示例在内联语句的 *Dates* 和 *Sales* 列中添加了两行

```
Add Load
x as Dates,
y as Sales
Inline
[
Dates,Sales
2001/09/1,1000
2001/09/10,-300
]
```

Replace

在图表修改上下文中，**Replace** 前缀使用脚本定义的计算值更改 *HC1* 表的所有值。

语法：

```
Replace loadstatement
```

示例：

此示例使用 x 和 y 的和覆盖列 z 中的所有值。

```
Replace Load
x+y as z
Resident HC1;
```

10.3 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。

图表修改器常规语句概述

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。

LOAD

在图表修改上下文中，**LOAD** 语句从脚本中定义的数据或以前加载的表中向超立方体加载其他数据。还可从分析连接加载数据。

> ***LOAD** 语句必须具有 **Replace** 或 **Add** 前缀，否则将被拒绝。*

```
Add | Replace Load [ distinct ] fieldlist

(

inline data [ format-spec ] |

resident table-label

) | extension pluginname.functionname([script] tabledescription)]

[ where criterion | while criterion ]

[ group by groupbyfieldlist ]

[order by orderbyfieldlist ]
```

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
```

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
```

Put
Put 语句用于在超立方体中设置一些数值。

HCValue
用于检索指定列的行中的值的 **HCValue** 语句。

Load

在图表修改上下文中，**LOAD** 语句从脚本中定义的数据或以前加载的表中向超立方体加载其他数据。还可从分析连接加载数据。

> ***LOAD** 语句必须具有 **Replace** 或 **Add** 前缀，否则将被拒绝。*

语法：

```
Add | Replace LOAD fieldlist

(

inline data [ format-spec ] |

resident table-label

) | extension pluginname.functionname([script] tabledescription)]

[ where criterion | while criterion ]

[ group by groupbyfieldlist ]

[order by orderbyfieldlist ]
```

参数：

参数

参数	描述
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。
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）。有关更多信息，请参阅*格式规格项目 (page 159)*。

参数	描述
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* 是加在创建原始表格的 **LOAD** 语句之前的标签。该标签需要在末尾加上冒号。
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**扩展**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。
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	描述
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。

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。

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;
```

Put

put 语句用于在超立方体中设置一些数值。

可以通过标签访问列。您还可以按声明顺序访问列和行。有关更多详细信息，请参见下面的示例。

语法：

```
put column(position)=value
```

Example 1:

可以通过标签访问列。

此示例将在标记为 *Sales* 的列的第一个位置设置 1 的值。

```
Put Sales(1) = 1;
```

Example 2:

可以使用度量的 `#hc1.measure` 格式按声明顺序访问度量值列。

本例将设置最终排序后的超多维数据集第十个位置的值 1000。

```
Put #hc1.measure.2(10) = 1000;
```

Example 3:

可以使用维度的 `#hc1.dimension` 格式按声明顺序访问维度行。

本例将常数 Pi 的值放在第三个声明维度的第五行。

```
Put #hc1.dimension.3(5) = Pi();
```

> *如果值或标签中没有此类维度或表达式，则返回一个错误，指示未找到该列。如果列的索引超出界限，则不会显示错误。*

HCValue

用于检索指定列的行中的值的 **HCValue** 函数。

语法：

```
HCValue(column,position)
```

Example 1:

此示例返回标签为“Sales”的列的第一个位置处的值。

```
HCValue(Sales,1)
```

Example 2:

本例返回排序后的超多维数据集第十个位置的值。

```
HCValue(#hc1.measure2,10)
```

Example 3:

此示例返回第三维中第五行的值。

```
HCValue(#hc1.dimension.3,5)
```

如果值或标签中没有此类维度或表达式，则返回一个错误，指示未找到该列。如果列的索引超出界限，则返回 NULL。

11 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。

11.1 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	

11.2 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**

11.3 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**

12 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。

12.1 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**。

12.2 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**

12.3 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 1275)*
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 522)*
QlikViewVersion	使用 **EngineVersion**。 *EngineVersion (page 1422)*
ProductVersion	使用 **EngineVersion**。 *EngineVersion (page 1422)*
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。